

Documentation

Welcome to IT4Innovations documentation pages. The IT4Innovations national supercomputing center operates supercomputers Salomon and Anselm. The supercomputers are available to academic community within the Czech Republic and Europe and industrial community worldwide. The purpose of these pages is to provide a comprehensive documentation on hardware, software and usage of the computers.

!!! Note “Note” What’s New


!!! warning “Reminder” History of Downtimes


How to read the documentation

1. Read the list in the left column. Select the subject of interest. Alternatively, use the Search box in the upper right corner.
2. Read the CONTENTS in the upper right corner.
3. Scan for all the yellow bulb call-outs on the page.
4. Read the details if still more information is needed. **Look for examples** illustrating the concepts.

!!! Note “Note” The call-out. Focus on the call-outs before reading full details.



Getting Help and Support

!!! Note “Note” Contact support [at] it4i.cz for help and support regarding the cluster technology at IT4Innovations. Please use **Czech**, **Slovak** or **English** language for communication with us. Follow the status of your request to IT4Innovations at support.it4i.cz/rt .


Use your IT4Innotations username and password to log in to the support  portal.

Required Proficiency

!!! Note “Note” You need basic proficiency in Linux environment.

In order to use the system for your calculations, you need basic proficiency in Linux environment. To gain the proficiency, we recommend you reading the introduction to Linux  operating system environment and installing a Linux distribution on your personal computer. A good choice might be the Fedora  distribution, as it is similar to systems on the clusters at IT4Innovations. It’s easy to install and use. In fact, any distribution would do.

!!! Note “Note” Learn how to parallelize your code!

In many cases, you will run your own code on the cluster. In order to fully exploit the cluster, you will need to carefully consider how to utilize all the cores available on the node and how to use multiple nodes at the same time. You need to **parallelize** your code. Proficiency in MPI, OpenMP, CUDA, UPC or GPI2 programming may be gained via the training provided by IT4Innovations. .

Terminology Frequently Used on These Pages

- **node:** a computer, interconnected by network to other computers - Computational nodes are powerful computers, designed and dedicated for executing demanding scientific computations.

- **core:** processor core, a unit of processor, executing computations
- **corehours:** wall clock hours of processor core time - Each node is equipped with **X** processor cores, provides **X** corehours per 1 wall clock hour.
- **job:** a calculation running on the supercomputer - The job allocates and utilizes resources of the supercomputer for certain time.
- **HPC:** High Performance Computing
- **HPC (computational) resources:** corehours, storage capacity, software licences
- **code:** a program
- **primary investigator (PI):** a person responsible for execution of computational project and utilization of computational resources allocated to that project
- **collaborator:** a person participating on execution of computational project and utilization of computational resources allocated to that project
- **project:** a computational project under investigation by the PI - The project is identified by the project ID. The computational resources are allocated and charged per project.
- **jobscript:** a script to be executed by the PBS Professional workload manager

Conventions

In this documentation, you will find a number of pages containing examples. We use the following conventions:

Cluster command prompt

\$

Your local linux host command prompt

local \$

Errata

Although we have taken every care to ensure the accuracy of our content, mistakes do happen. If you find a mistake in the text or the code we would be grateful if you would report this to us. By doing so, you can save other readers from frustration and help us improve subsequent versions of this documentation. If you find any errata, please report them by visiting <http://support.it4i.cz/rt>, creating a new ticket, and entering the details of your errata. Once your errata are verified, your submission will be accepted and the errata will be uploaded on our website.

Applying for Resources

Computational resources may be allocated by any of the following Computing resources allocation mechanisms.

Academic researchers can apply for computational resources via Open Access Competitions.

Anyone is welcomed to apply via the Directors Discretion.

Foreign (mostly European) users can obtain computational resources via the PRACE (DECI) program.

In all cases, IT4Innovations' access mechanisms are aimed at distributing computational resources while taking into account the development and application of supercomputing methods and their benefits and usefulness for society. The applicants are expected to submit a proposal. In the proposal, the applicants **apply for a particular amount of core-hours** of computational resources. The requested core-hours should be substantiated by scientific excellence of the proposal, its computational maturity and expected impacts. Proposals do undergo a scientific, technical and economic evaluation. The allocation decisions are based on this evaluation. More information at Computing resources allocation and Obtaining Login Credentials page.

Certificates FAQ

FAQ about certificates in general

Q: What are certificates?

IT4Innovations employs X.509 certificates for secure communication (e. g. credentials exchange) and for grid services related to PRACE, as they present a single method of authentication for all PRACE services, where only one password is required.

There are different kinds of certificates, each with a different scope of use. We mention here:

- User (Private) certificates
- Certificate Authority (CA) certificates
- Host certificates
- Service certificates

However, users need only manage User and CA certificates. Note that your user certificate is protected by an associated private key, and this private key must never be disclosed.

Q: Which X.509 certificates are recognised by IT4Innovations?

Any certificate that has been issued by a Certification Authority (CA) from a member of the IGTF (<http://www.igtf.net>) is recognised by IT4Innovations: European certificates are issued by members of the EUGridPMA (<https://www.eugridmpa.org>), which is part of the IGTF and coordinates the trust fabric for e-Science Grid authentication within Europe. Further the Czech “*Qualified certificate*” (*Kvalifikovaný certifikát*) (provided by <http://www.postsignum.cz/> or <http://www.ica.cz/Kvalifikovany-certifikat.aspx>), that is used in electronic contact with Czech public authorities is accepted.

Q: How do I get a User Certificate that can be used with IT4Innovations?

To get a certificate, you must make a request to your local, IGTF approved, Certificate Authority (CA). Usually you then must visit, in person, your nearest Registration Authority (RA) to verify your affiliation and identity (photo identification is required). Usually, you will then be emailed details on how to retrieve your certificate, although procedures can vary between CAs. If you are in Europe, you can locate your trusted CA via <http://www.eugridpma.org/members/worldmap>.

In some countries certificates can also be retrieved using the TERENA Certificate Service, see the FAQ below for the link.

Q: Does IT4Innovations support short lived certificates (SLCS)?

Yes, provided that the CA which provides this service is also a member of IGTF.

Q: Does IT4Innovations support the TERENA certificate service?

Yes, ITInnovations supports TERENA eScience personal certificates. For more information, please visit <https://tcs-esience-portal.terena.org>, where you also can find if your organisation/country can use this service

Q: What format should my certificate take?

User Certificates come in many formats, the three most common being the 'PKCS12', 'PEM' and the JKS formats.

The PKCS12 (often abbreviated to 'p12') format stores your user certificate, along with your associated private key, in a single file. This form of your certificate is typically employed by web browsers, mail clients, and grid services like UNICORE, DART, gsissh-term and Globus toolkit (GSI-SSH, GridFTP and GRAM5).

The PEM format (*.pem) stores your user certificate and your associated private key in two separate files. This form of your certificate can be used by PRACE's gsissh-term and with the grid related services like Globus toolkit (GSI-SSH, GridFTP and GRAM5).

To convert your Certificate from PEM to p12 formats, and *vice versa*, IT4Innovations recommends using the openssl tool (see separate FAQ entry).

JKS is the Java KeyStore and may contain both your personal certificate with your private key and a list of your trusted CA certificates. This form of your certificate can be used by grid services like DART and UNICORE6.

To convert your Certificate from p12 to JKS, IT4Innovations recommends using the keytool utility (see separate FAQ entry).

Q: What are CA certificates?

Certification Authority (CA) certificates are used to verify the link between your user certificate and the authority which issued it. They are also used to verify the link between the host certificate of a IT4Innovations server and the CA which issued that certificate. In essence they establish a chain of trust between you and the target server. Thus, for some grid services, users must have a copy of all the CA certificates.


To assist users, SURFsara (a member of PRACE) provides a complete and up-to-date bundle of all the CA certificates that any PRACE user (or IT4Innovations grid services user) will require. Bundle of certificates, in either p12, PEM or JKS formats, are available from <http://winnetou.sara.nl/prace/certs/>.

It is worth noting that gsissh-term and DART automatically updates their CA certificates from this SURFsara website. In other cases, if you receive a warning that a server's certificate can not be validated (not trusted), then please update your CA certificates via the SURFsara website. If this fails, then please contact the IT4Innovations helpdesk.

Lastly, if you need the CA certificates for a personal Globus 5 installation, then you can install the CA certificates from a MyProxy server with the following command.

```
myproxy-get-trustroots -s myproxy-prace.lrz.de
```

If you run this command as 'root', then it will install the certificates into /etc/grid-security/certificates. If you run this not as 'root', then the certificates will be installed into

\$HOME/.globus/certificates. For Globus, you can download the globuscerts.tar.gz packet from <http://winnetou.sara.nl/prace/certs/>.

Q: What is a DN and how do I find mine?

DN stands for Distinguished Name and is part of your user certificate. IT4Innovations needs to know your DN to enable your account to use the grid services. You may use openssl (see below) to determine your DN or, if your browser contains your user certificate, you can extract your DN from your browser.

For Internet Explorer users, the DN is referred to as the “subject” of your certificate. Tools->Internet Options->Content->Certificates->View->Details->Subject.

For users running Firefox under Windows, the DN is referred to as the “subject” of your certificate. Tools->Options->Advanced->Encryption->View Certificates. Highlight your name and then Click View->Details->Subject.

Q: How do I use the openssl tool?

The following examples are for Unix/Linux operating systems only.

To convert from PEM to p12, enter the following command:

```
openssl pkcs12 -export -in usercert.pem -inkey userkey.pem -out
username.p12
```

To convert from p12 to PEM, type the following *four* commands:


```
openssl pkcs12 -in username.p12 -out usercert.pem -clcerts -nokeys
openssl pkcs12 -in username.p12 -out userkey.pem -nocerts
chmod 444 usercert.pem
chmod 400 userkey.pem
```

To check your Distinguished Name (DN), enter the following command:

```
openssl x509 -in usercert.pem -noout -subject -nameopt
RFC2253
```

To check your certificate (e.g., DN, validity, issuer, public key algorithm, etc.), enter the following command:

```
openssl x509 -in usercert.pem -text -noout
```

To download openssl for both Linux and Windows, please visit <http://www.openssl.org/related/binaries.html>. On Macintosh Mac OS X computers openssl is already pre-installed and can be used immediately.

Q: How do I create and then manage a keystore?

IT4innovations recommends the java based keytool utility to create and manage keystores, which themselves are stores of keys and certificates. For example if you want to convert your pkcs12 formatted key pair into a java keystore you can use the following command.


```
keytool -importkeystore -srckeystore $my_p12_cert -destkeystore
$my_keystore -srcstoretype pkcs12 -deststoretype jks -alias
$my_nickname -destalias $my_nickname
```

where `$my_p12_cert` is the name of your p12 (pkcs12) certificate, `$my_keystore` is the name that you give to your new java keystore and `$my_nickname` is the alias name that the p12 certificate was given and is used also for the new keystore.

You also can import CA certificates into your java keystore with the tool, e.g.:

```
keytool -import -trustcacerts -alias $mydomain -file $mydomain.crt -keystore $my_keys
```


where `$mydomain.crt` is the certificate of a trusted signing authority (CA) and `$mydomain` is the alias name that you give to the entry.

More information on the tool can be found at: <http://docs.oracle.com/javase/7/docs/technotes/tools/solaris/keytool.html> 

Q: How do I use my certificate to access the different grid Services?

Most grid services require the use of your certificate; however, the format of your certificate depends on the grid Service you wish to employ.

If employing the PRACE version of GSISSH-term (also a Java Web Start Application), you may use either the PEM or p12 formats. Note that this service automatically installs up-to-date PRACE CA certificates.

If the grid service is UNICORE, then you bind your certificate, in either the p12 format or JKS, to UNICORE during the installation of the client on your local machine. For more information, please visit UNICORE6 in PRACE 

If the grid service is part of Globus, such as GSI-SSH, GriFTP or GRAM5, then the certificates can be in either p12 or PEM format and must reside in the “`$HOME/.globus`” directory for Linux and Mac users or `%HOMEPATH%.globus` for Windows users. (Windows users will have to use the DOS command ‘`cmd`’ to create a directory which starts with a ‘`’`’). Further, user certificates should be named either “`usercred.p12`” or “`usercert.pem`” and “`userkey.pem`”, and the CA certificates must be kept in a pre-specified directory as follows. For Linux and Mac users, this directory is either `$HOME/.globus/certificates` or `/etc/grid-security/certificates`. For Windows users, this directory is `%HOMEPATH%.globuscertificates`. (If you are using GSISSH-Term from `prace-ri.eu` then you do not have to create the `.globus` directory nor install CA certificates to use this tool alone).

Q: How do I manually import my certificate into my browser?

If you employ the Firefox browser, then you can import your certificate by first choosing the “Preferences” window. For Windows, this is Tools->Options. For Linux, this is Edit->Preferences. For Mac, this is Firefox->Preferences. Then, choose the “Advanced” button; followed by the “Encryption” tab. Then, choose the “Certificates” panel; select the option “Select one automatically” if you have only one certificate, or “Ask me every time” if you have more than one. Then click on the “View Certificates” button to open the “Certificate Manager” window. You can then select the “Your Certificates” tab and click on button “Import”. Then locate the PKCS12 (.p12) certificate you wish to import, and employ its associated password.

If you are a Safari user, then simply open the “Keychain Access” application and follow “File->Import items”.

If you are an Internet Explorer user, click Start->Settings->Control Panel and then double-click on Internet. On the Content tab, click Personal, and then click Import. In the Password box, type your password. NB you may be prompted multiple times for your password. In the “Certificate

File To Import” box, type the filename of the certificate you wish to import, and then click OK. Click Close, and then click OK.

Q: What is a proxy certificate?

A proxy certificate is a short-lived certificate which may be employed by UNICORE and the Globus services. The proxy certificate consists of a new user certificate and a newly generated proxy private key. This proxy typically has a rather short lifetime (normally 12 hours) and often only allows a limited delegation of rights. Its default location, for Unix/Linux, is `/tmp/x509_uuid` but can be set via the `$X509_USER_PROXY` environment variable.

Q: What is the MyProxy service?

The MyProxy Service [\[1\]](#), can be employed by `gssh-term` and Globus tools, and is an online repository that allows users to store long lived proxy certificates remotely, which can then be retrieved for use at a later date. Each proxy is protected by a password provided by the user at the time of storage. This is beneficial to Globus users as they do not have to carry their private keys and certificates when travelling; nor do users have to install private keys and certificates on possibly insecure computers.

Q: Someone may have copied or had access to the private key of my certificate either in a separate file or in the browser. What should I do?

Please ask the CA that issued your certificate to revoke this certificate and to supply you with a new one. In addition, please report this to IT4Innovations by contacting the support team [\[2\]](#).

Obtaining Login Credentials

Obtaining Authorization

The computational resources of IT4I are allocated by the Allocation Committee to a Project, investigated by a Primary Investigator. By allocating the computational resources, the Allocation Committee is authorizing the PI to access and use the clusters. The PI may decide to authorize a number of her/his Collaborators to access and use the clusters, to consume the resources allocated to her/his Project. These collaborators will be associated to the Project. The Figure below is depicting the authorization chain:

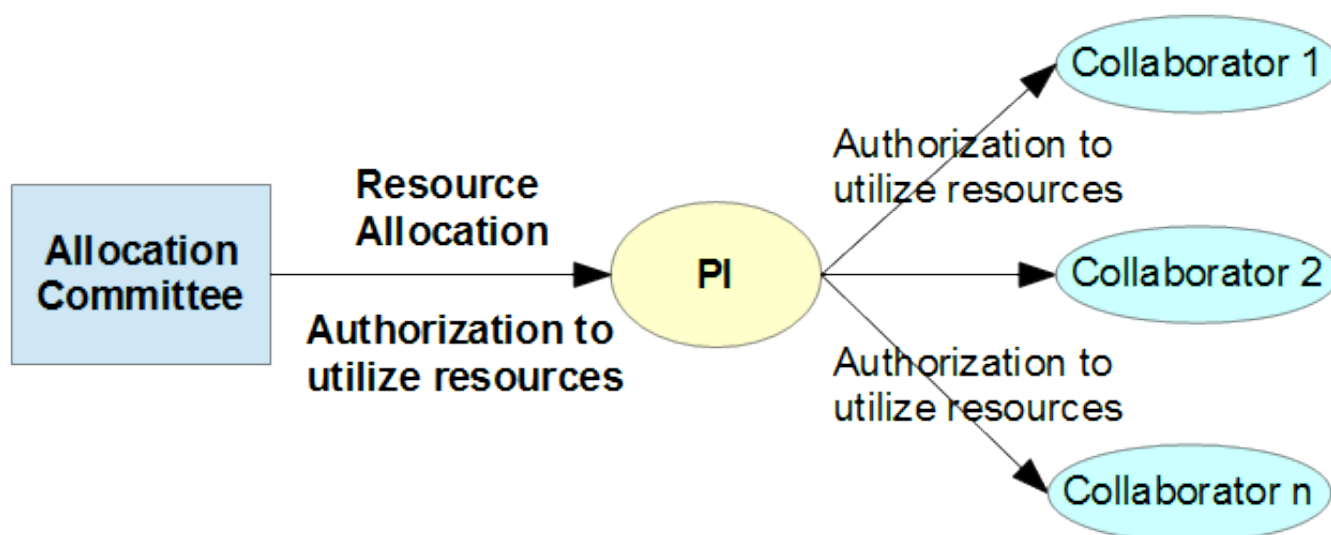


Figure 1:

You need to either become the PI or be named as a collaborator by a PI in order to access and use the clusters.

Head of Supercomputing Services acts as a PI of a project DD-13-5. Joining this project, you may **access and explore the clusters**, use software, development environment and computers via the qexp and qfree queues. You may use these resources for own education/research, no paperwork is required. All IT4I employees may contact the Head of Supercomputing Services in order to obtain **free access to the clusters**.

Authorization of PI by Allocation Committee

The PI is authorized to use the clusters by the allocation decision issued by the Allocation Committee. The PI will be informed by IT4I about the Allocation Committee decision.

Authorization by web

This is a preferred way of granting access to project resources. Please, use this method whenever it's possible.

Log in to the IT4I Extranet portal [🔗](#) using IT4I credentials and go to the **Projects** section.

- **Users:** Please, submit your requests for becoming a project member.
- **Primary Investigators:** Please, approve or deny users' requests in the same section.

Authorization by e-mail (an alternative approach)

In order to authorize a Collaborator to utilize the allocated resources, the PI should contact the IT4I support [📧](#) (E-mail: support [at] it4i.cz) and provide following information:

1. Identify your project by project ID
2. Provide list of people, including himself, who are authorized to use the resources allocated to the project. The list must include full name, e-mail and affiliation. Provide usernames as well, if collaborator login access already exists on the IT4I systems.
3. Include “Authorization to IT4Innovations” into the subject line.

Example (except the subject line which must be in English, you may use Czech or Slovak language for communication with us):

Subject: Authorization to IT4Innovations

Dear support,

Please include my collaborators to project OPEN-0-0.

John Smith, john.smith@myemail.com, Department of Chemistry, MIT, US

Jonas Johansson, jjohansson@otheremail.se, Department of Physics, Royal Institute of Technology

Luisa Fibonacci, lf@emailitalia.it, Department of Mathematics, National Research Council

Thank you,

PI

(**Digitally** signed)

Should the above information be provided by e-mail, the e-mail **must be** digitally signed. Read more on digital signatures below.

The Login Credentials

Once authorized by PI, every person (PI or Collaborator) wishing to access the clusters, should contact the IT4I support [📧](#) (E-mail: support [at] it4i.cz) providing following information:

1. Project ID
2. Full name and affiliation
3. Statement that you have read and accepted the Acceptable use policy document [📄](#) (AUP).
4. Attach the AUP file.
5. Your preferred username, max 8 characters long. The preferred username must associate your surname and name or be otherwise derived from it. Only alphanumeric sequences, dash and underscore signs are allowed.
6. In case you choose Alternative way to personal certificate, a **scan of photo ID** (personal ID or passport or driver license) is required

Example (except the subject line which must be in English, you may use Czech or Slovak language for communication with us):

Subject: Access to IT4Innovations

Dear support,

Please open the user account for me and attach the account to OPEN-0-0

Name and affiliation: John Smith, john.smith@myemail.com, Department of Chemistry, MIT

I have read and accept the Acceptable use policy document (attached)

Preferred username: johnsm

Thank you,
John Smith
(Digitally signed)

Should the above information be provided by e-mail, the e-mail **must be** digitally signed. To sign an e-mail, you need digital certificate. Read more on digital signatures below.

Digital signature allows us to confirm your identity in remote electronic communication and provides an encrypted channel to exchange sensitive information such as login credentials. After receiving your signed e-mail with the requested information, we will send you your login credentials (user name, key, passphrase and password) to access the IT4I systems.

We accept certificates issued by any widely respected certification authority.

For various reasons we do not accept PGP keys.** Please, use only X.509 PKI certificates for communication with us.**

You will receive your personal login credentials by protected e-mail. The login credentials include:

1. username
2. ssh private key and private key passphrase
3. system password

The clusters are accessed by the private key and username. Username and password is used for login to the information systems listed on <http://support.it4i.cz/>.

Change Passphrase

On Linux, use

```
local $ ssh-keygen -f id_rsa -p
```

On Windows, use PuTTY Key Generator.

Change Password

Change password in your user profile at <https://extranet.it4i.cz/user/>.

The Certificates for Digital Signatures

We accept personal certificates issued by any widely respected certification authority (CA). This includes certificates by CAs organized in International Grid Trust Federation (<http://www.igtf.net/>), its European branch EUGridPMA - <https://www.eugridpma.org/> and its member organizations, e.g. the CESNET certification authority - <https://tcs-p.cesnet.cz/confusa/>. The Czech “*Qualified certificate*” (*Kvalifikovaný certifikát*) (provided by <http://www.postsignum.cz/> or <http://www.ica.cz/Kvalifikovany-certifikat.aspx>), that is used in electronic contact with Czech authorities is accepted as well.


Certificate generation process is well-described here:

- How to generate a personal TCS certificate in Mozilla Firefox web browser (in Czech)

A FAQ about certificates can be found here: [Certificates FAQ](#).





Alternative Way to Personal Certificate

Follow these steps **only** if you can not obtain your certificate in a standard way. In case you choose this procedure, please attach a **scan of photo ID** (personal ID or passport or drivers license) when applying for login credentials.

1. Go to <https://www.cacert.org/>
 - If there's a security warning, just acknowledge it.
2. Click *Join*.
3. Fill in the form and submit it by the *Next* button.
 - Type in the e-mail address which you use for communication with us.
 - Don't forget your chosen *Pass Phrase*.
4. You will receive an e-mail verification link. Follow it.
5. After verifying, go to the CAcert's homepage and login using *Password Login*.
6. Go to *Client Certificates* -> *New*.
7. Tick *Add* for your e-mail address and click the *Next* button.
8. Click the *Create Certificate Request* button.
9. You'll be redirected to a page from where you can download/install your certificate.
 - Simultaneously you'll get an e-mail with a link to the certificate.

Installation of the Certificate Into Your Mail Client

The procedure is similar to the following guides:

- MS Outlook 2010
 - How to Remove, Import, and Export Digital certificates
 - Importing a PKCS #12 certificate (in Czech)
- Mozilla Thudnerbird
 - Installing an SMIME certificate
 - Importing a PKCS #12 certificate (in Czech)

End of User Account Lifecycle

User accounts are supported by membership in active Project(s) or by affiliation to IT4Innovations. User accounts, that loose the support (meaning, are not attached to an active project and are not affiliated with IT4I), will be deleted 1 year after the last project to which they were attached expires.

User will get 3 automatically generated warning e-mail messages of the pending removal:.

- First message will be sent 3 months before the removal
- Second message will be sent 1 month before the removal
- Third message will be sent 1 week before the removal.

The messages will inform about the projected removal date and will challenge the user to migrate her/his data




Graphical User Interface

X Window System

The X Window system is a principal way to get GUI access to the clusters.

Read more about configuring **X Window System**.

VNC

The **Virtual Network Computing (VNC)** is a graphical desktop sharing  system that uses the Remote Frame Buffer protocol (RFB)  to remotely control another computer .


Read more about configuring **VNC**.

Cygwin and X11 forwarding

If no able to forward X11 using PuTTY to CygwinX

```
[username@login1.anselm ~]$ gnome-session &  
[1] 23691  
[username@login1.anselm ~]$ PuTTY X11 proxy: unable to connect to forwarded X server: Netw  
PuTTY X11 proxy: unable to connect to forwarded X server: Network error: Connection refus
```

(gnome-session:23691): WARNING **: Cannot open display:**

1. Locate and modify Cygwin shortcut that uses startxwin  locate C:\cygwin64\bin\XWin.exe change it to C:\cygwin64\bin\XWin.exe -listen tcp

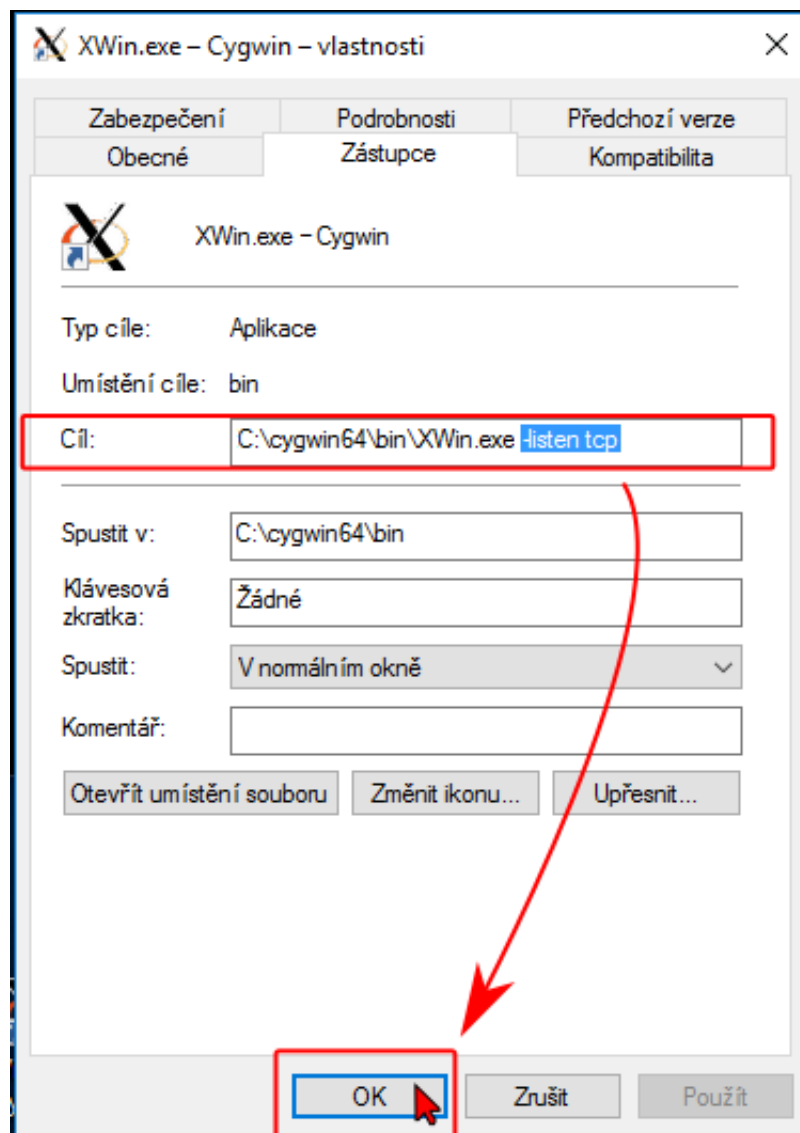


Figure 1: XWin-listen-tcp.png

2. Check Putty settings: Enable X11 forwarding

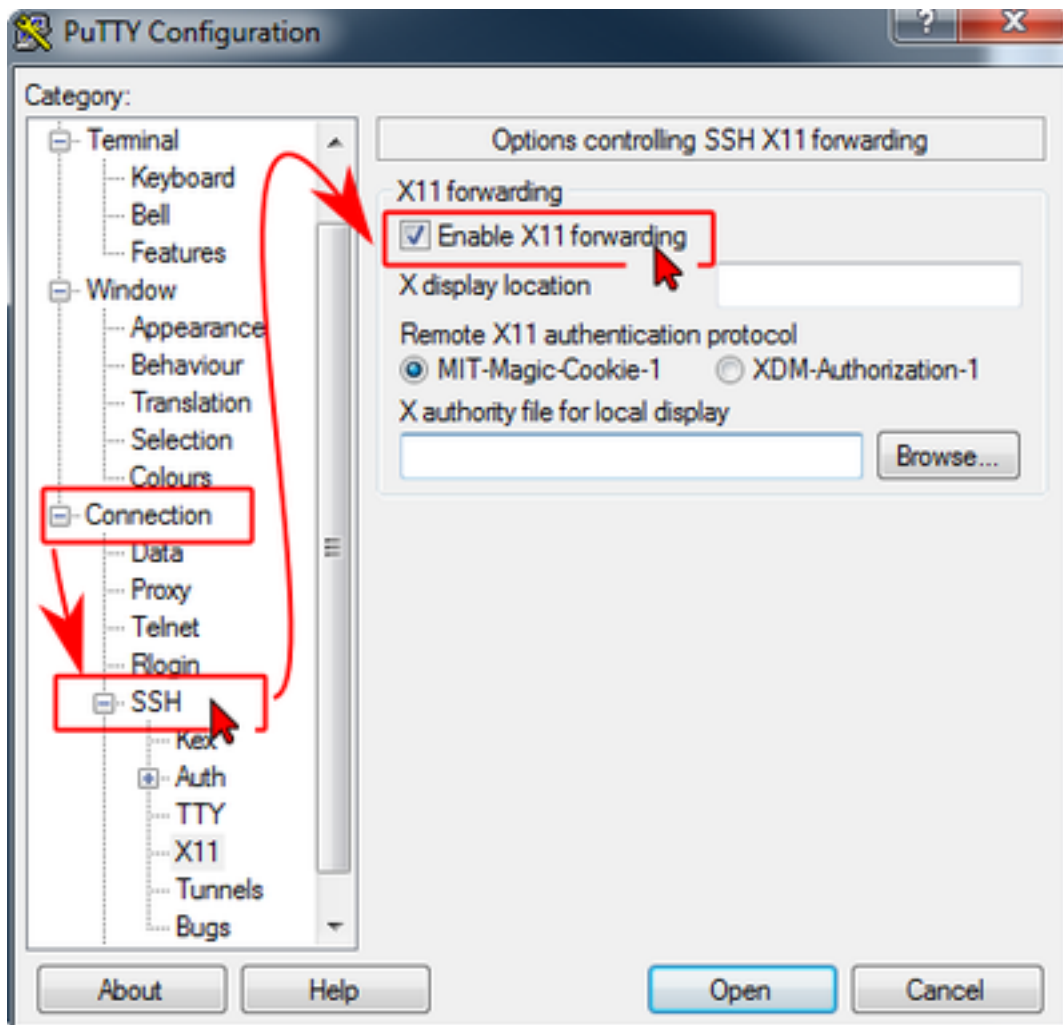


Figure 2:

VNC

The **Virtual Network Computing (VNC)** is a graphical desktop sharing system that uses the Remote Frame Buffer protocol (RFB) to remotely control another computer. It transmits the keyboard and mouse events from one computer to another, relaying the graphical screen updates back in the other direction, over a network.

The recommended clients are TightVNC or TigerVNC (free, open source, available for almost any platform).

Create VNC password

!!! Note “Note” Local VNC password should be set before the first login. Do use a strong password.

```
[username@login2 ~]$ vncpasswd
Password:
Verify:
```

Start vncserver

!!! Note “Note” To access VNC a local vncserver must be started first and also a tunnel using SSH port forwarding must be established.

[See below] (vnc.md#linux-example-of-creating-a-tunnel) for the details on SSH tunnels. In the

You can find ports which are already occupied. Here you can see that ports ” /usr/bin/Xvnc :79” and ” /usr/bin/Xvnc :60” are occupied.

```
[username@login2 ~]$ ps aux | grep Xvnc
username      5971  0.0  0.0 201072 92564 ?        SN   Sep22   4:19 /usr/bin/Xvnc :79 -des
username      10296 0.0  0.0 131772 21076 pts/29   SN   13:01   0:01 /usr/bin/Xvnc :60 -de
.....
```

Choose free port e.g. 61 and start your VNC server:

```
[username@login2 ~]$ vncserver :61 -geometry 1600x900 -depth 16
```

New 'login2:1 (username)' desktop is login2:1

Starting applications specified in /home/username/.vnc/xstartup

Log file is /home/username/.vnc/login2:1.log

Check if VNC server is started on the port (in this example 61):

```
[username@login2 .vnc]$ vncserver -list
```

TigerVNC server sessions:

```
X DISPLAY #      PROCESS ID
:61        18437
```

Another command:

```
[username@login2 .vnc]$ ps aux | grep Xvnc

username      10296 0.0  0.0 131772 21076 pts/29   SN   13:01   0:01 /usr/bin/Xvnc :61 -de
```

To access the VNC server you have to create a tunnel between the login node using TCP **port 5961** and your machine using a free TCP port (for simplicity the very same, in this case).

!!! Note “Note” The tunnel must point to the same login node where you launched the VNC server, eg. login2. If you use just cluster-name.it4i.cz, the tunnel might point to a different node due to DNS round robin.

Linux/Mac OS example of creating a tunnel

At your machine, create the tunnel:

```
local $ ssh -TN -f username@login2.cluster-name.it4i.cz -L 5961:localhost:5961
```

Issue the following command to check the tunnel is established (please note the PID 2022 in the last column, you'll need it for closing the tunnel):

```
local $ netstat -natp | grep 5961
(Not all processes could be identified, non-owned process info
will not be shown, you would have to be root to see it all.)
tcp        0      0 127.0.0.1:5961          0.0.0.0:*               LISTEN      2022/sshd
tcp6       0      0 :::5961                 :::*                     LISTEN      2022/sshd
```

Or on Mac OS use this command:

```
local-mac $ lsof -n -iTCP:5961 | grep LISTEN
ssh 75890 sta545 7u IPv4 0xf6062b5c15a56a3b 0t0 TCP 127.0.0.1:5961 (LISTEN)
```

Connect with the VNC client:

```
local $ vncviewer 127.0.0.1:5961
```

In this example, we connect to VNC server on port 5961, via the ssh tunnel. The connection is encrypted and secured. The VNC server listening on port 5961 provides screen of 1600x900 pixels.

You have to destroy the SSH tunnel which is still running at the background after you finish the work. Use the following command (PID 2022 in this case, see the netstat command above):

```
kill 2022
```

Windows example of creating a tunnel

Use PuTTY to log in on cluster.

Start vncserver using command vncserver described above.

Search for the localhost and port number (in this case 127.0.0.1:5961).

```
[username@login2 .vnc]$ netstat -tanp | grep Xvnc
(Not all processes could be identified, non-owned process info
will not be shown, you would have to be root to see it all.)
tcp        0      0 127.0.0.1:5961          0.0.0.0:*               LISTEN      2400/Xvnc
```

On the PuTTY Configuration screen go to Connection->SSH->Tunnels to set up the tunnel.

Fill the Source port and Destination fields. **Do not forget to click the Add button.**

Run the VNC client of your choice, select VNC server 127.0.0.1, port 5961 and connect using VNC password.

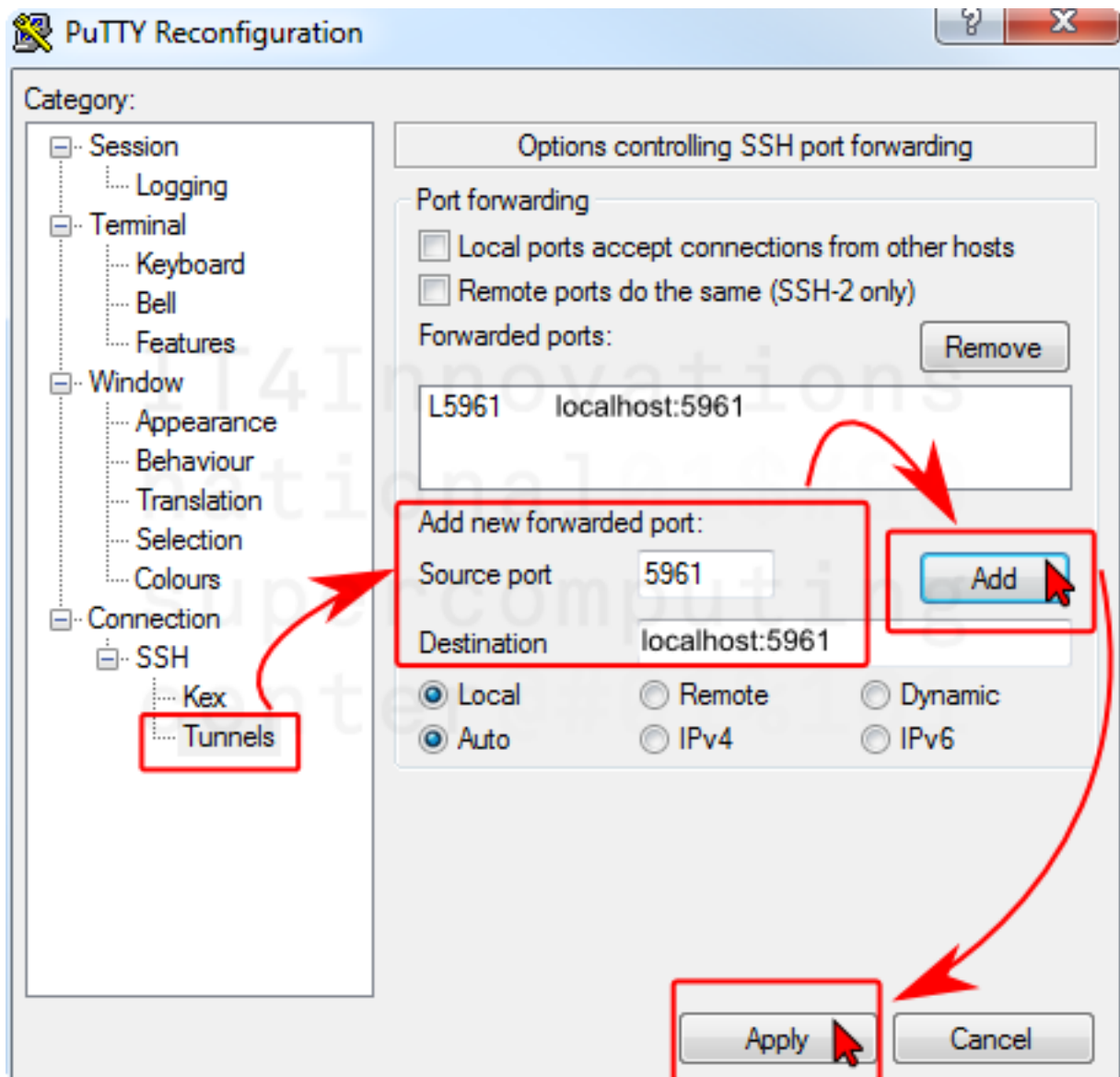


Figure 1:

Example of starting TigerVNC viewer

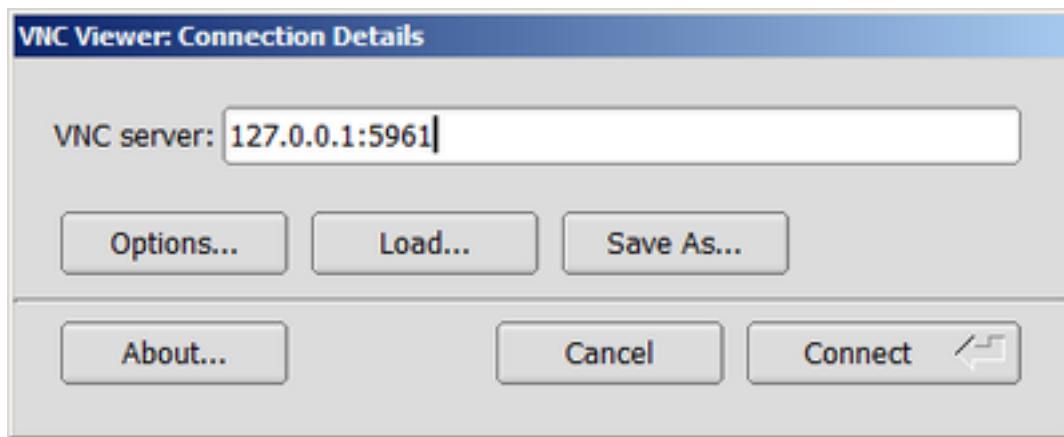


Figure 2:

In this example, we connect to VNC server on port 5961, via the ssh tunnel, using TigerVNC viewer. The connection is encrypted and secured. The VNC server listening on port 5961 provides screen of 1600x900 pixels.

Example of starting TightVNC Viewer

Use your VNC password to log using TightVNC Viewer and start a Gnome Session on the login node.

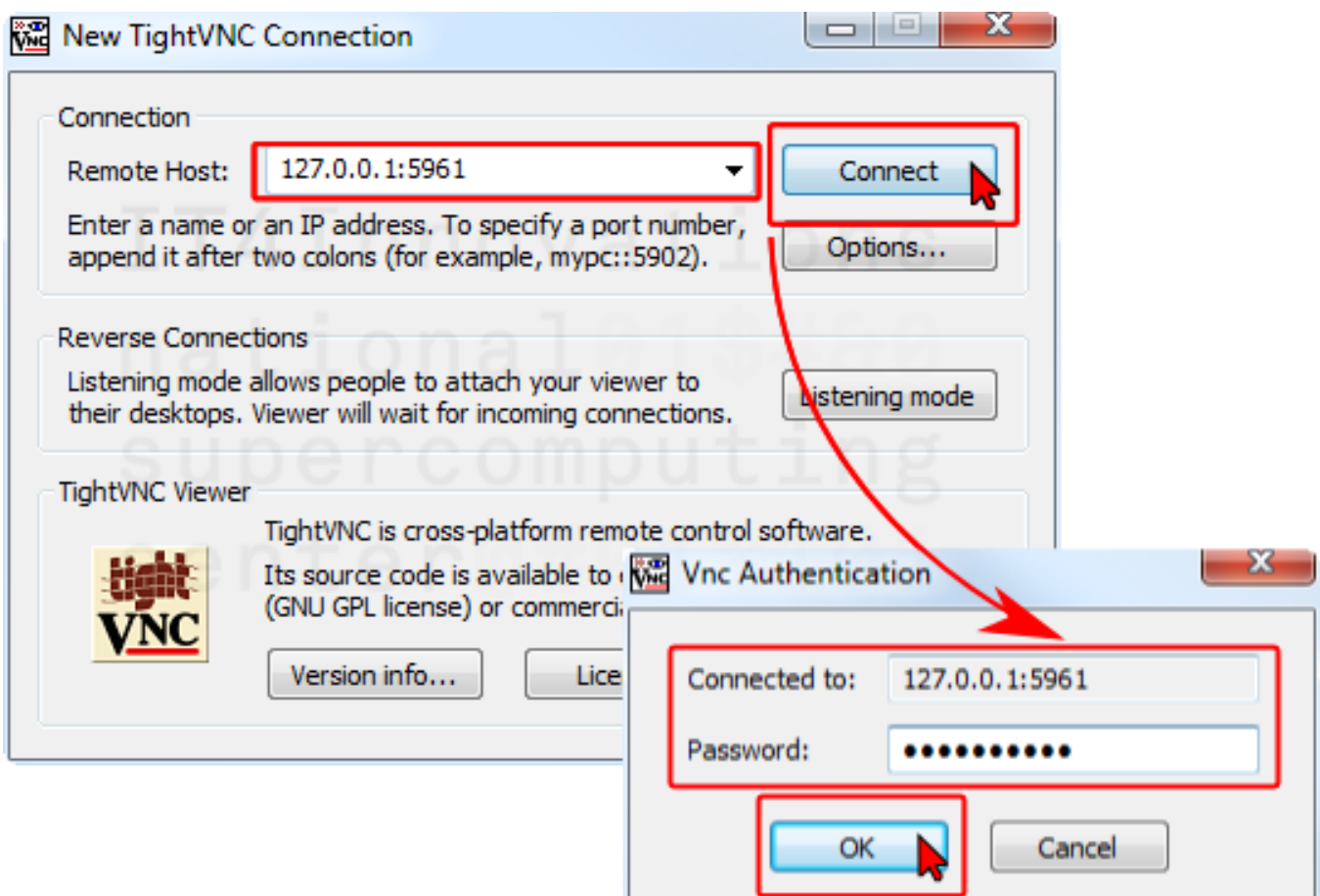


Figure 3:

Gnome session

You should see after the successful login.

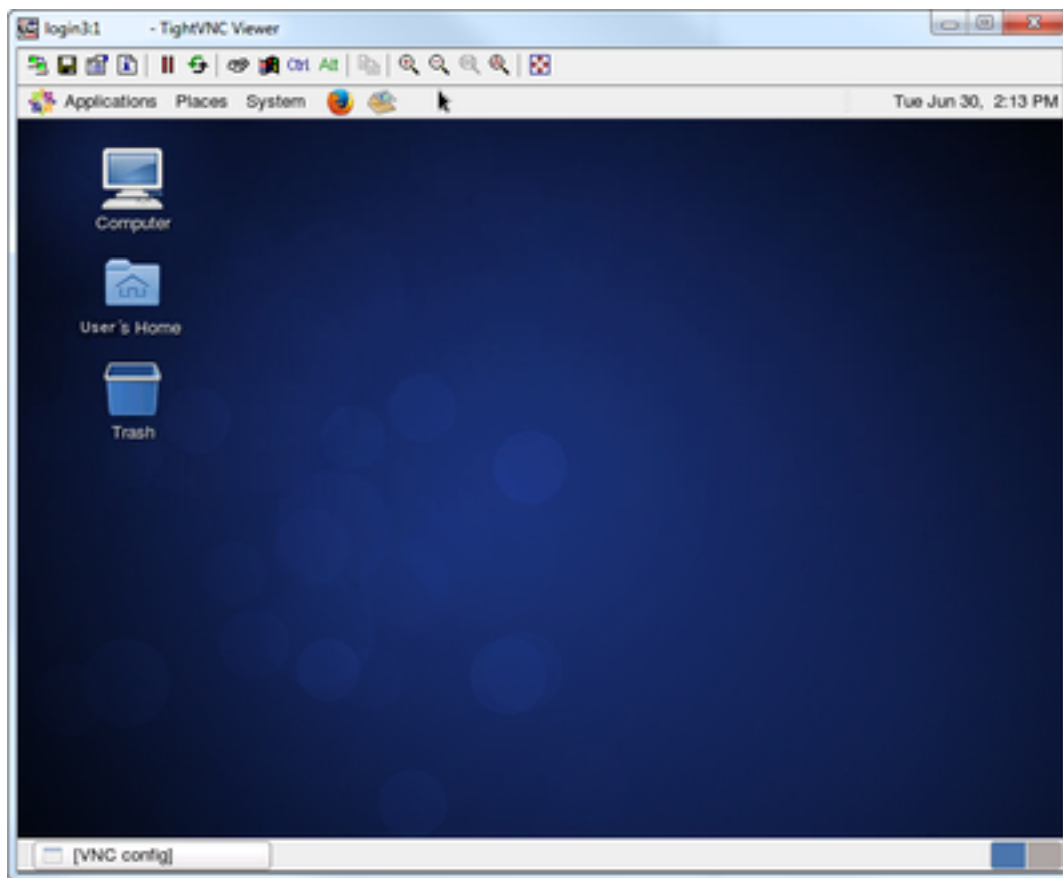


Figure 4:

Disable your Gnome session screensaver

Open Screensaver preferences dialog:

Uncheck both options below the slider:

Kill screensaver if locked screen

If the screen gets locked you have to kill the screensaver. Do not to forget to disable the screensaver then.

```
[username@login2 .vnc]$ ps aux | grep screen
username      1503  0.0  0.0 103244   892 pts/4    S+   14:37   0:00 grep screen
username      24316 0.0  0.0 270564   3528 ?        Ss   14:12   0:00 gnome-screensaver

[username@login2 .vnc]$ kill 24316
```

Kill vncserver after finished work

You should kill your VNC server using command:

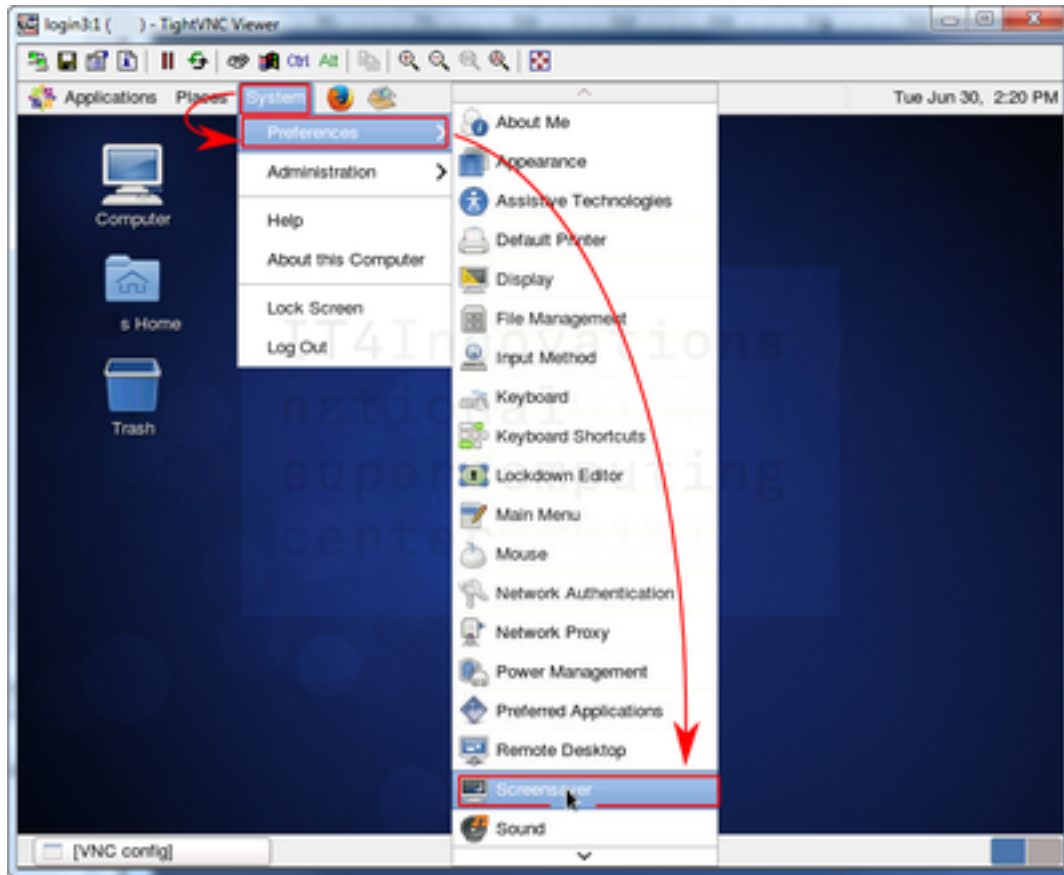


Figure 5:

```
[username@login2 .vnc]$ vncserver -kill :61
Killing Xvnc process ID 7074
Xvnc process ID 7074 already killed
```

Or this way:

```
[username@login2 .vnc]$ pkill vnc
```

GUI applications on compute nodes over VNC

The very same methods as described above, may be used to run the GUI applications on compute nodes. However, for maximum performance, proceed following these steps:

Open a Terminal (Applications -> System Tools -> Terminal). Run all the next commands in the terminal.

Allow incoming X11 graphics from the compute nodes at the login node:

```
$ xhost +
```

Get an interactive session on a compute node (for more detailed info look here). Use the **-v DISPLAY** option to propagate the DISPLAY on the compute node. In this example, we want a complete node (24 cores in this example) from the production queue:

```
$ qsub -I -v DISPLAY=$(uname -n):$(echo $DISPLAY | cut -d ':' -f 2) -A PROJECT_ID -q qpro
```

Test that the DISPLAY redirection into your VNC session works, by running a X11 application (e. g. XTerm) on the assigned compute node:

```
$ xterm
```

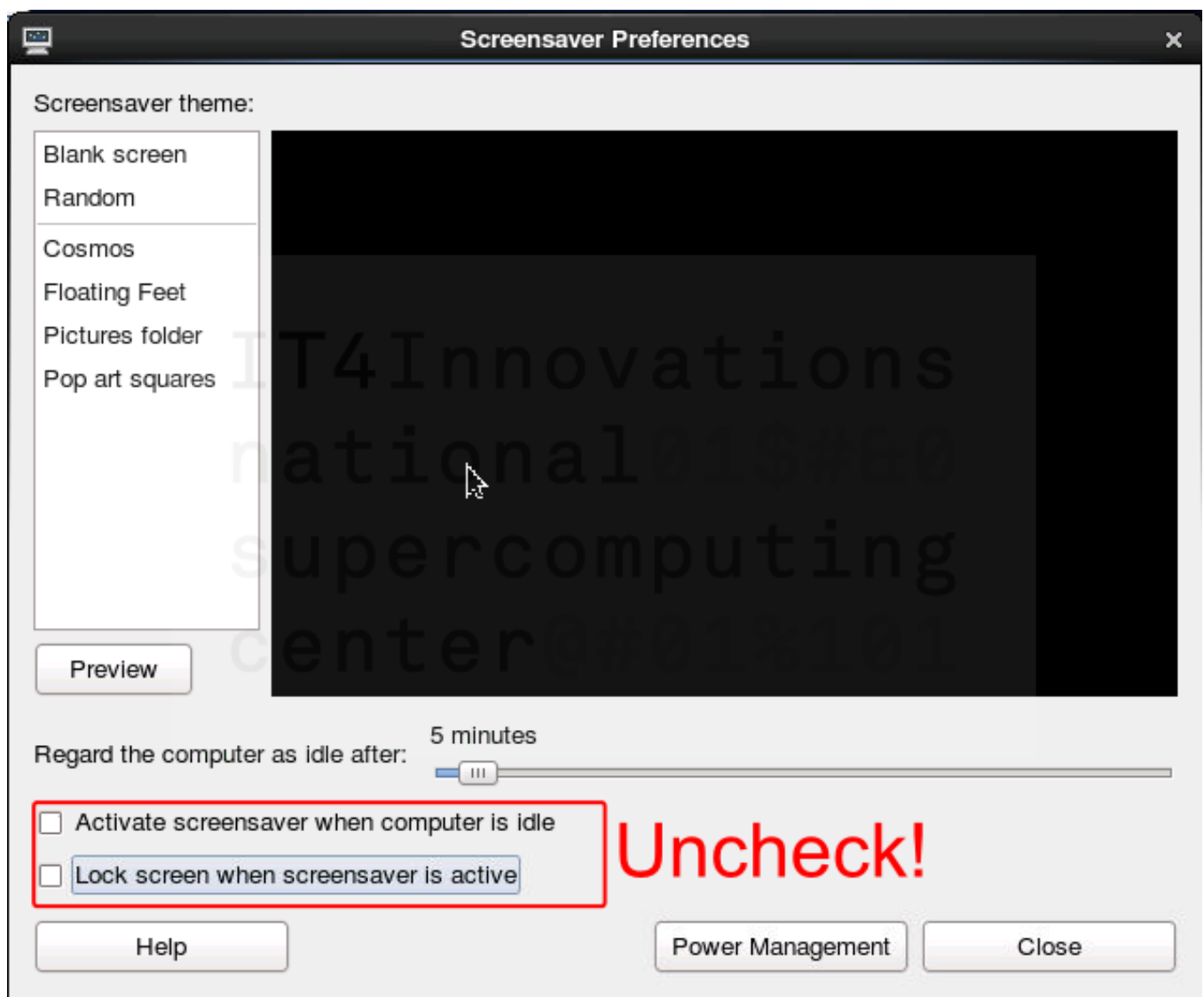


Figure 6:

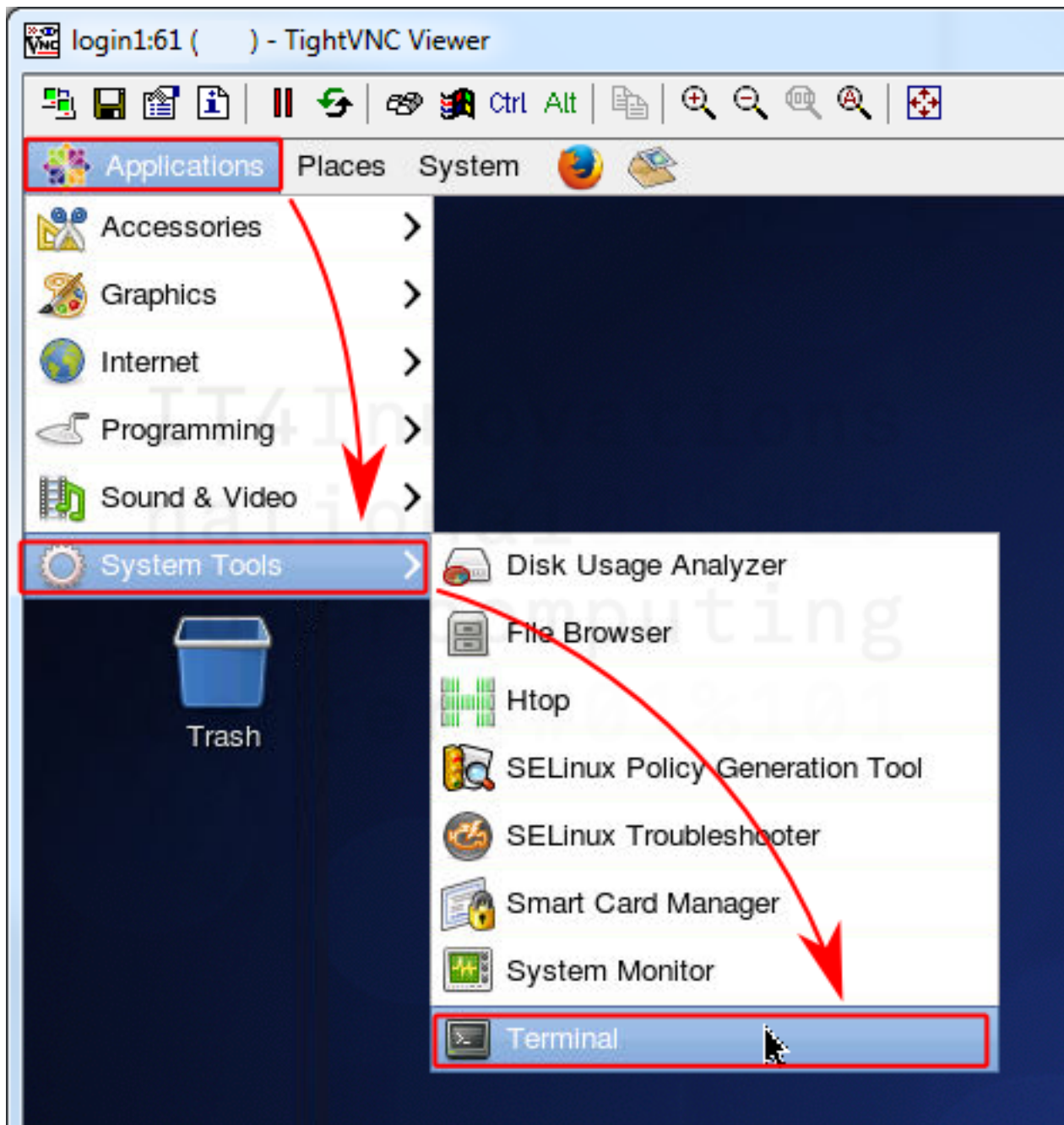


Figure 7:

Example described above:

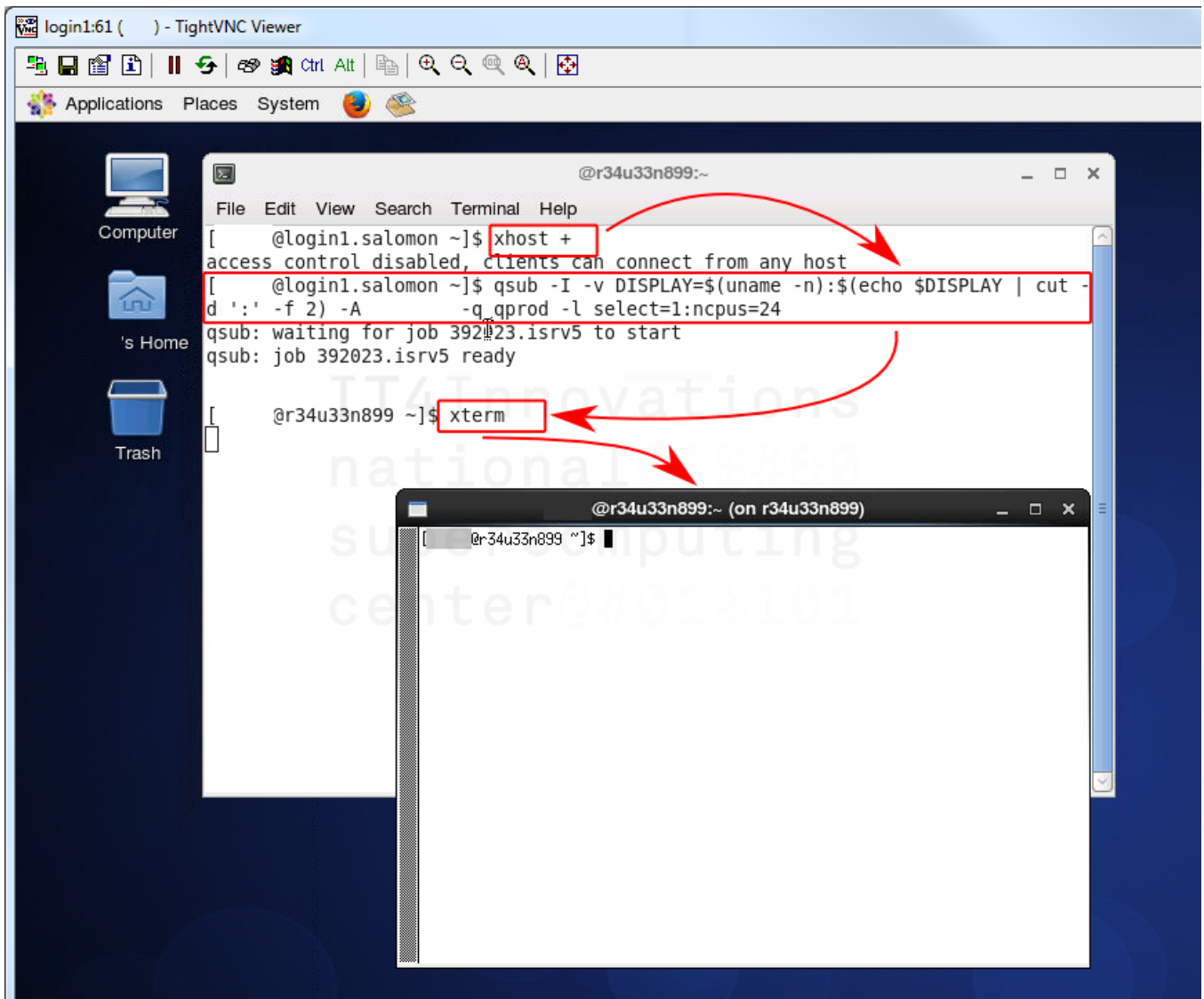


Figure 8:

X Window System

The X Window system is a principal way to get GUI access to the clusters. The **X Window System** (commonly known as **X11**, based on its current major version being 11, or shortened to simply **X**, and sometimes informally **X-Windows**) is a computer software system and network protocol [\[1\]](#) that provides a basis for graphical user interfaces [\[2\]](#) (GUIs) and rich input device capability for networked computers [\[3\]](#).

!!! Note “Note” The X display forwarding must be activated and the X server running on client side

X display

In order to display graphical user interface GUI of various software tools, you need to enable the X display forwarding. On Linux and Mac, log in using the -X option tho ssh client:

```
local $ ssh -X username@cluster-name.it4i.cz
```

X Display Forwarding on Windows

On Windows use the PuTTY client to enable X11 forwarding. In PuTTY menu, go to Connection->SSH->X11, mark the Enable X11 forwarding checkbox before logging in. Then log in as usual.

To verify the forwarding, type

```
$ echo $DISPLAY
```

if you receive something like

```
localhost:10.0
```

then the X11 forwarding is enabled.

X Server

In order to display graphical user interface GUI of various software tools, you need running X server on your desktop computer. For Linux users, no action is required as the X server is the default GUI environment on most Linux distributions. Mac and Windows users need to install and run the X server on their workstations.

X Server on OS X

Mac OS users need to install XQuartz server [\[4\]](#).

X Server on Windows

There are variety of X servers available for Windows environment. The commercial Xwin32 is very stable and rich featured. The Cygwin environment provides fully featured open-source XWin X server. For simplicity, we recommend open-source X server by the Xming project [\[5\]](#). For stability and full features we recommend the XWin [\[6\]](#) X server by Cygwin

|How to use Xwin |How to use Xming | | — | — | |Install Cygwin[↗](#) Find and execute XWin.exe to start the X server on Windows desktop computer. If not able to forward X11 using PuTTY to CygwinX |

Use Xlaunch to configure the Xming.

Run Xming to start the X server on Windows desktop computer.

Read more on http://www.math.umn.edu/systems_guide/putty_xwin32.html[↗](#)

Running GUI Enabled Applications

!!! Note “Note” Make sure that X forwarding is activated and the X server is running.

Then launch the application as usual. Use the & to run the application in background.

```
$ module load intel (idb and gvim not installed yet)
$ gvim &
$ xterm
```

In this example, we activate the intel programming environment tools, then start the graphical gvim editor.

GUI Applications on Compute Nodes

Allocate the compute nodes using -X option on the qsub command

```
$ qsub -q qexp -l select=2:ncpus=24 -X -I
```

In this example, we allocate 2 nodes via qexp queue, interactively. We request X11 forwarding with the -X option. It will be possible to run the GUI enabled applications directly on the first compute node.

Better performance is obtained by logging on the allocated compute node via ssh, using the -X option.

```
$ ssh -X r24u35n680
```

In this example, we log in on the r24u35n680 compute node, with the X11 forwarding enabled.

The Gnome GUI Environment

The Gnome 2.28 GUI environment is available on the clusters. We recommend to use separate X server window for displaying the Gnome environment.

Gnome on Linux and OS X

To run the remote Gnome session in a window on Linux/OS X computer, you need to install Xephyr. Ubuntu package is xserver-xephyr, on OS X it is part of XQuartz[↗](#). First, launch Xephyr on local machine:

```
local $ Xephyr -ac -screen 1024x768 -br -reset -terminate :1 &
```

This will open a new X window with size 1024x768 at DISPLAY :1. Next, ssh to the cluster with DISPLAY environment variable set and launch gnome-session

```
local $ DISPLAY=:1.0 ssh -XC yourname@cluster-name.it4i.cz -i ~/.ssh/path_to_your_key  
... cluster-name MOTD...  
yourname@login1.cluster-namen.it4i.cz $ gnome-session &
```

On older systems where Xephyr is not available, you may also try Xnest instead of Xephyr. Another option is to launch a new X server in a separate console, via:

```
xinit /usr/bin/ssh -XT -i .ssh/path_to_your_key yourname@cluster-namen.it4i.cz gnome-sess
```

However this method does not seem to work with recent Linux distributions and you will need to manually source `/etc/profile` to properly set environment variables for PBS.

Gnome on Windows

Use Xlaunch to start the Xming server or run the XWin.exe. Select the “One window” mode.

Log in to the cluster, using PuTTY. On the cluster, run the `gnome-session` command.

```
$ gnome-session &
```

In this way, we run remote gnome session on the cluster, displaying it in the local X server

Use System->Log Out to close the gnome-session

Accessing the Clusters

The IT4Innovations clusters are accessed by SSH protocol via login nodes.

!!! Note “Note” Read more on [Accessing the Salomon Clusterr](#) or [Accessing the Anselm Cluster](#) pages.

PuTTY

On **Windows**, use PuTTY ssh client.

SSH keys

Read more about SSH keys management.

Pageant SSH agent

Pageant holds your private key in memory without needing to retype a passphrase on every login.

- Run Pageant.
- On Pageant Key List press *Add key* and select your private key (id_rsa.ppk).
- Enter your passphrase.
- Now you have your private key in memory without needing to retype a passphrase on every login.

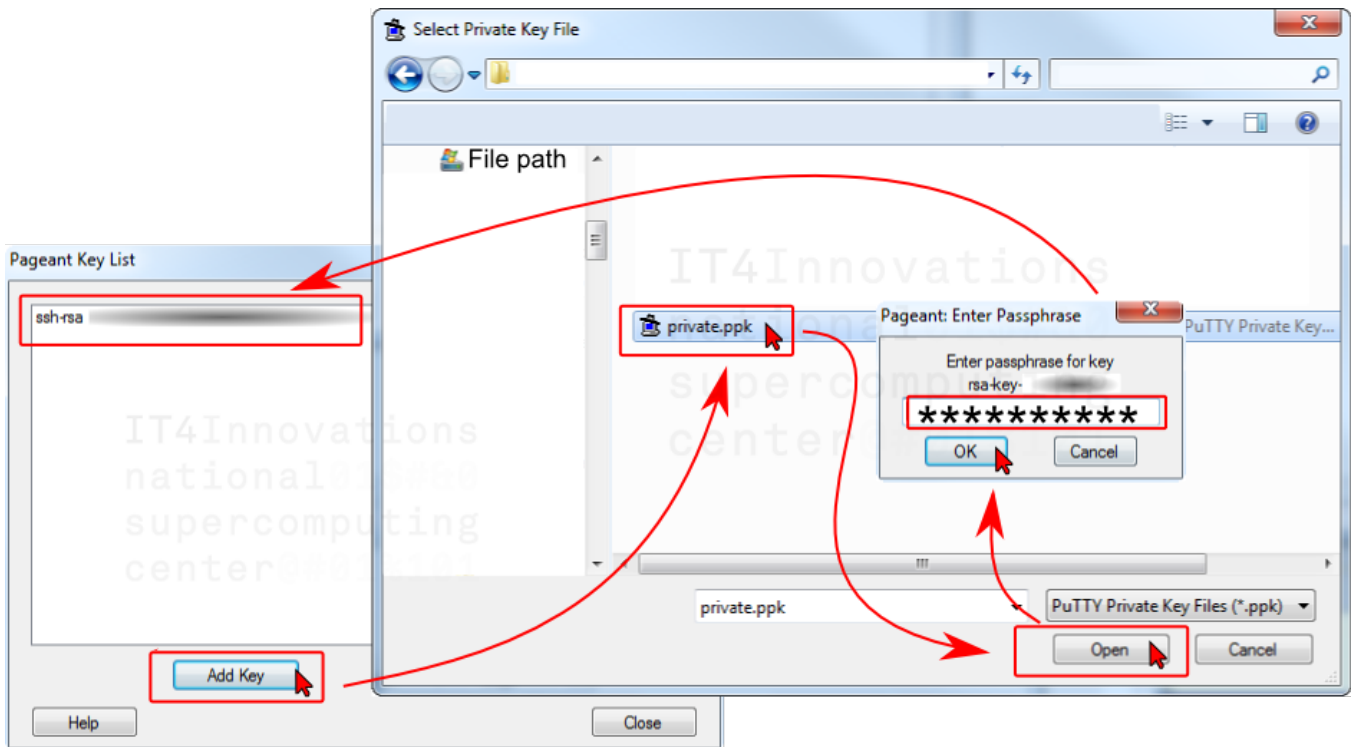


Figure 1:

VPN - Connection fail in Win 8.1

Failed to initialize connection subsystem Win 8.1 - 02-10-15 MS patch

AnyConnect users on Windows 8.1 will receive a “Failed to initialize connection subsystem” error after installing the Windows 8.1 02/10/15 security patch. This OS defect introduced with the 02/10/15 patch update will also impact Windows 7 users with IE11. Windows Server 2008/2012 are also impacted by this defect, but neither is a supported OS for AnyConnect.

Workaround:

- Close the Cisco AnyConnect Window and the taskbar mini-icon
- Right click vpnui.exe in the ‘Cisco AnyConnect Secure Mobility Client’ folder. (C:\Program Files (x86)\Cisco\Cisco AnyConnect Secure Mobility Client)
- Click on the ‘Run compatibility troubleshooter’ button
- Choose ‘Try recommended settings’
- The wizard suggests Windows 8 compatibility.
- Click ‘Test Program’. This will open the program.
- Close

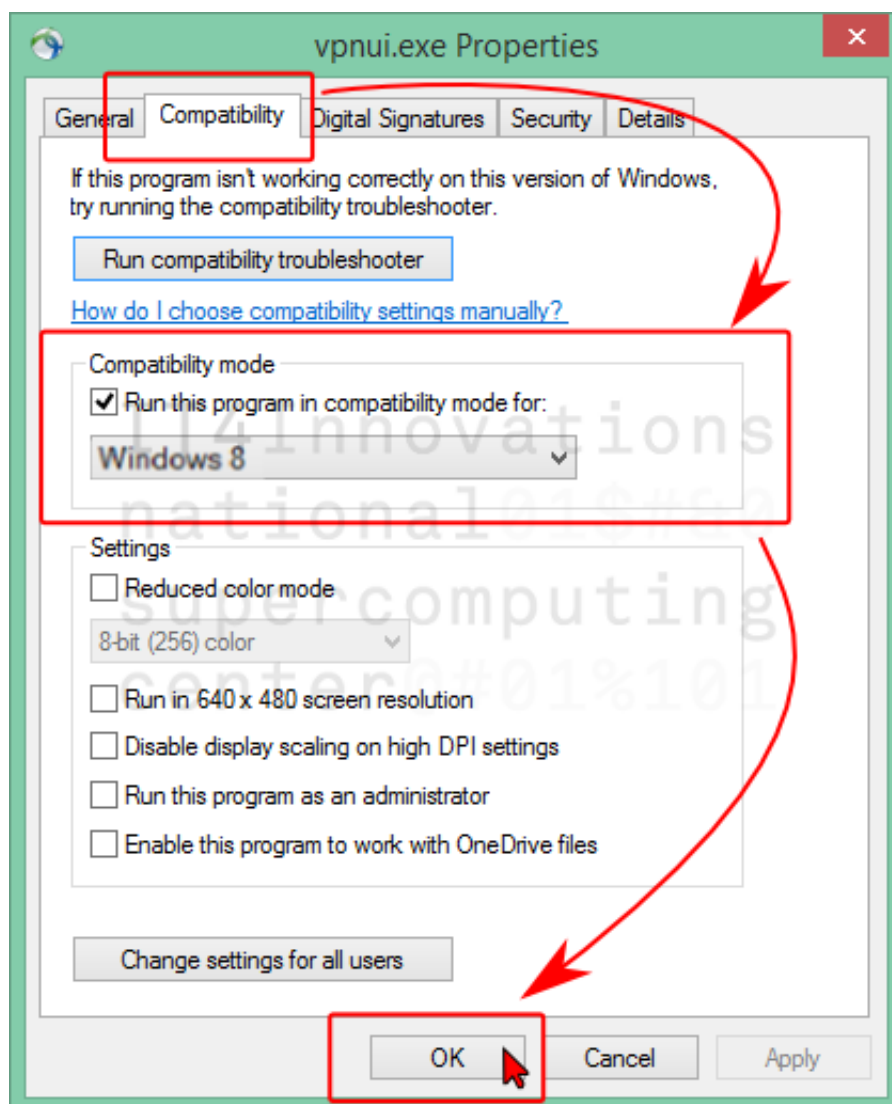


Figure 1:

PuTTY key generator

PuTTYgen is the PuTTY key generator. You can load in an existing private key and change your passphrase or generate a new public/private key pair.

Change Password for Existing Private Key

You can change the password of your SSH key with “PuTTY Key Generator”. Make sure to backup the key.

- Load your private key file with *Load* button.
- Enter your current passphrase.
- Change key passphrase.
- Confirm key passphrase.
- Save your private key with *Save private key* button.

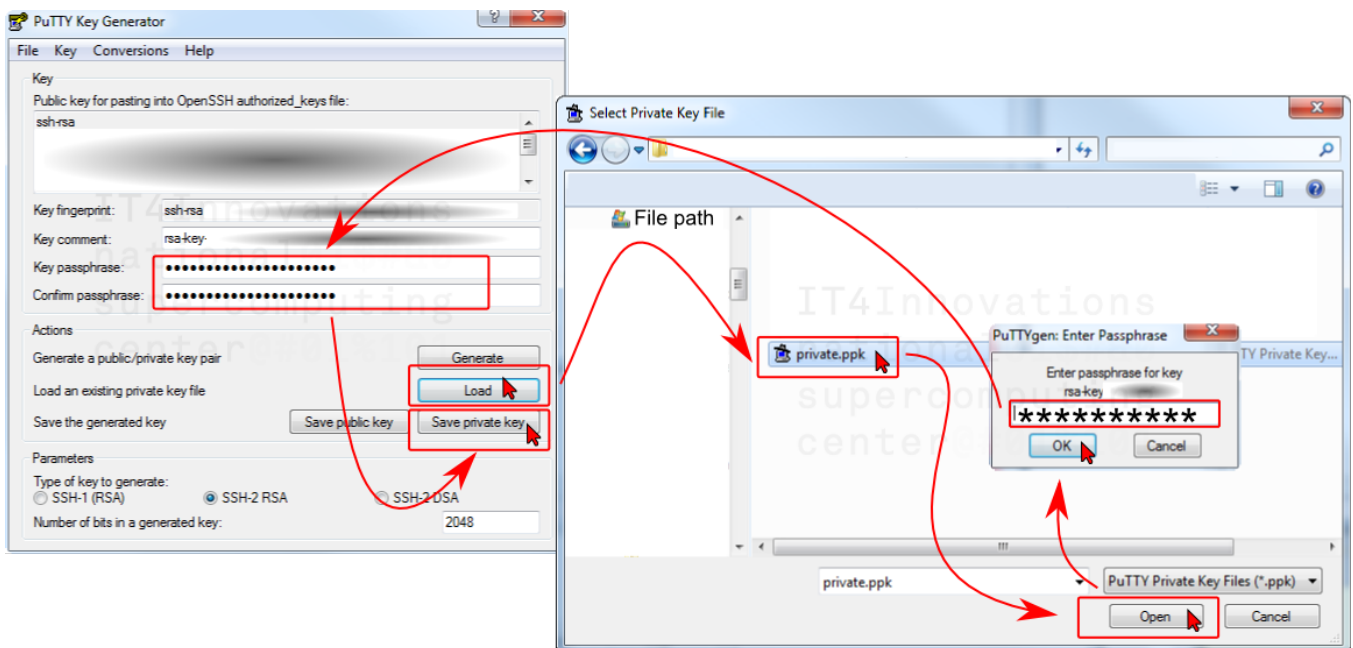


Figure 1:

Generate a New Public/Private key

You can generate an additional public/private key pair and insert public key into `authorized_keys` file for authentication with your own private key.

- Start with *Generate* button.
- Generate some randomness.
- Wait.
- Enter a *comment* for your key using format 'username@organization.example.com'. Enter key passphrase. Confirm key passphrase. Save your new private key in `"*.ppk"` format with *Save private key* button.
- Save the public key with *Save public key* button. You can copy public key out of the 'Public key for pasting into authorized_keys file' box.
- Export private key in OpenSSH format `"id_rsa"` using Conversion -> Export OpenSSH key

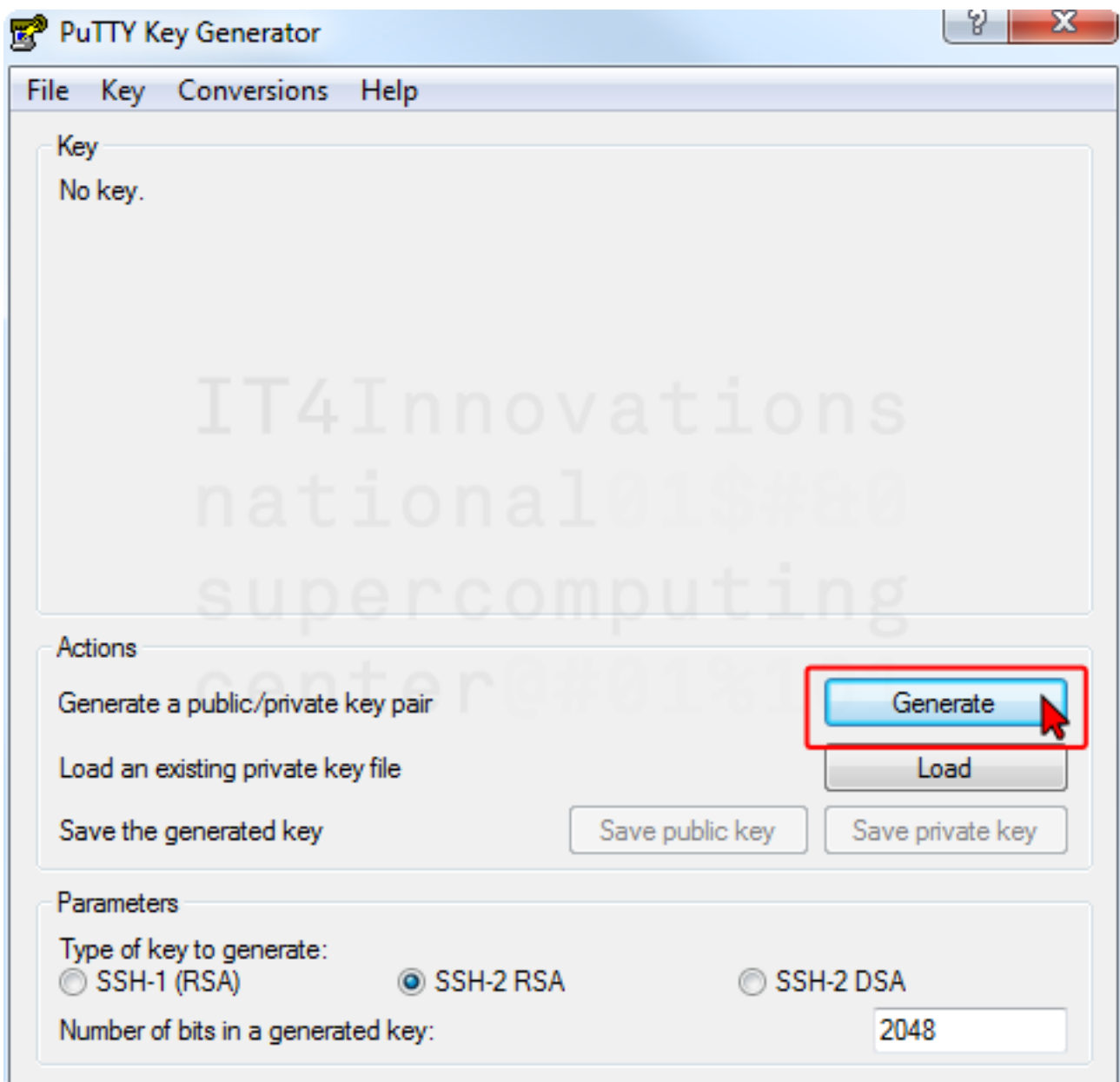


Figure 2:

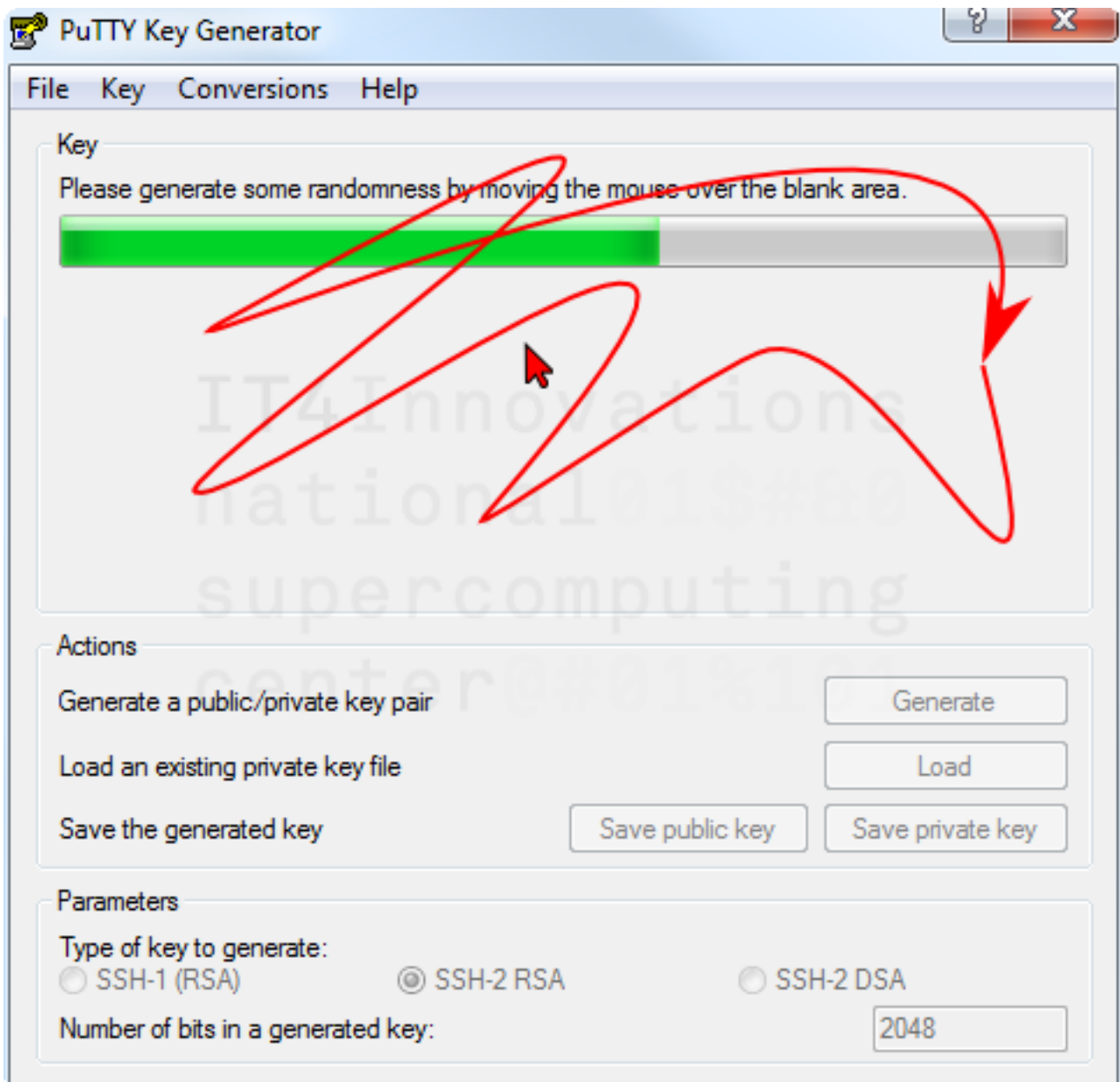


Figure 3:

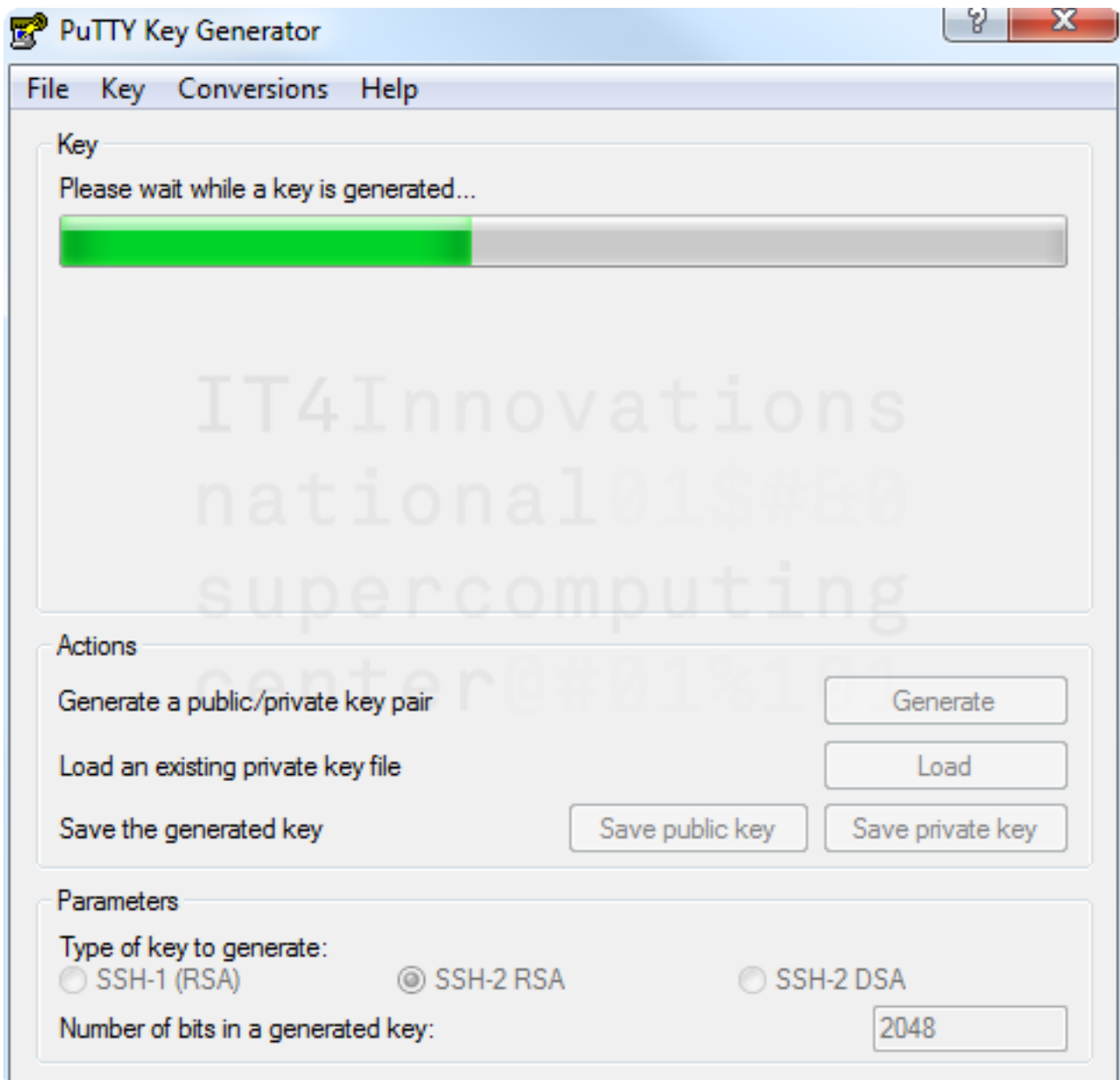


Figure 4:

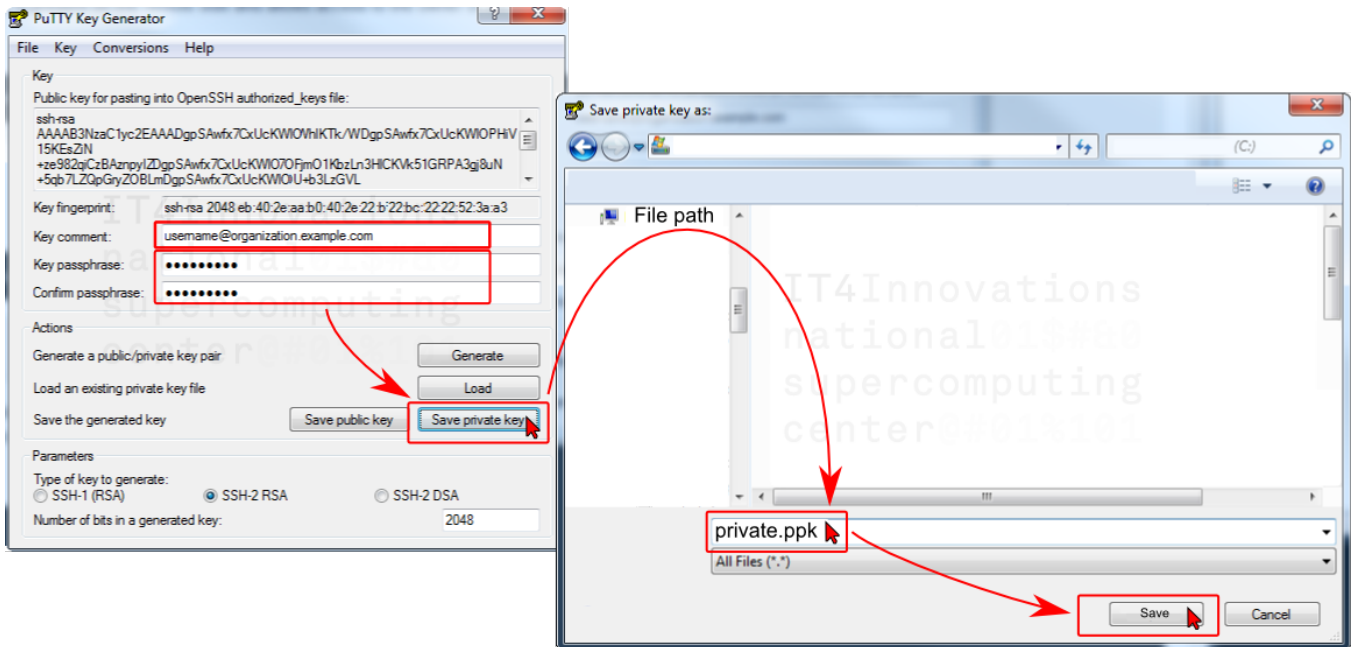


Figure 5:

- Now you can insert additional public key into `authorized_keys` file for authentication with your own private key. You must log in using ssh key received after registration. Then proceed to How to add your own key.

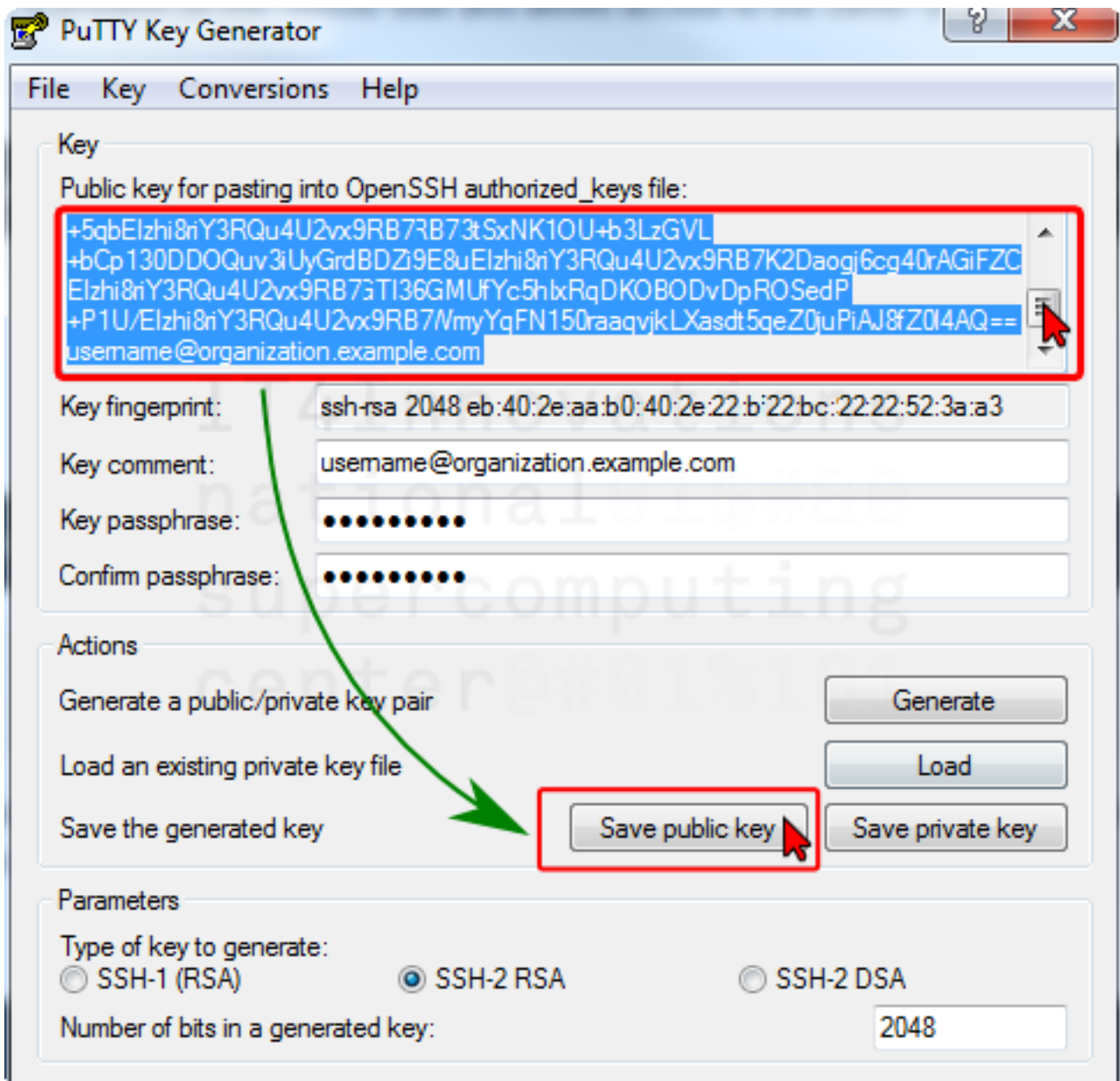


Figure 6:

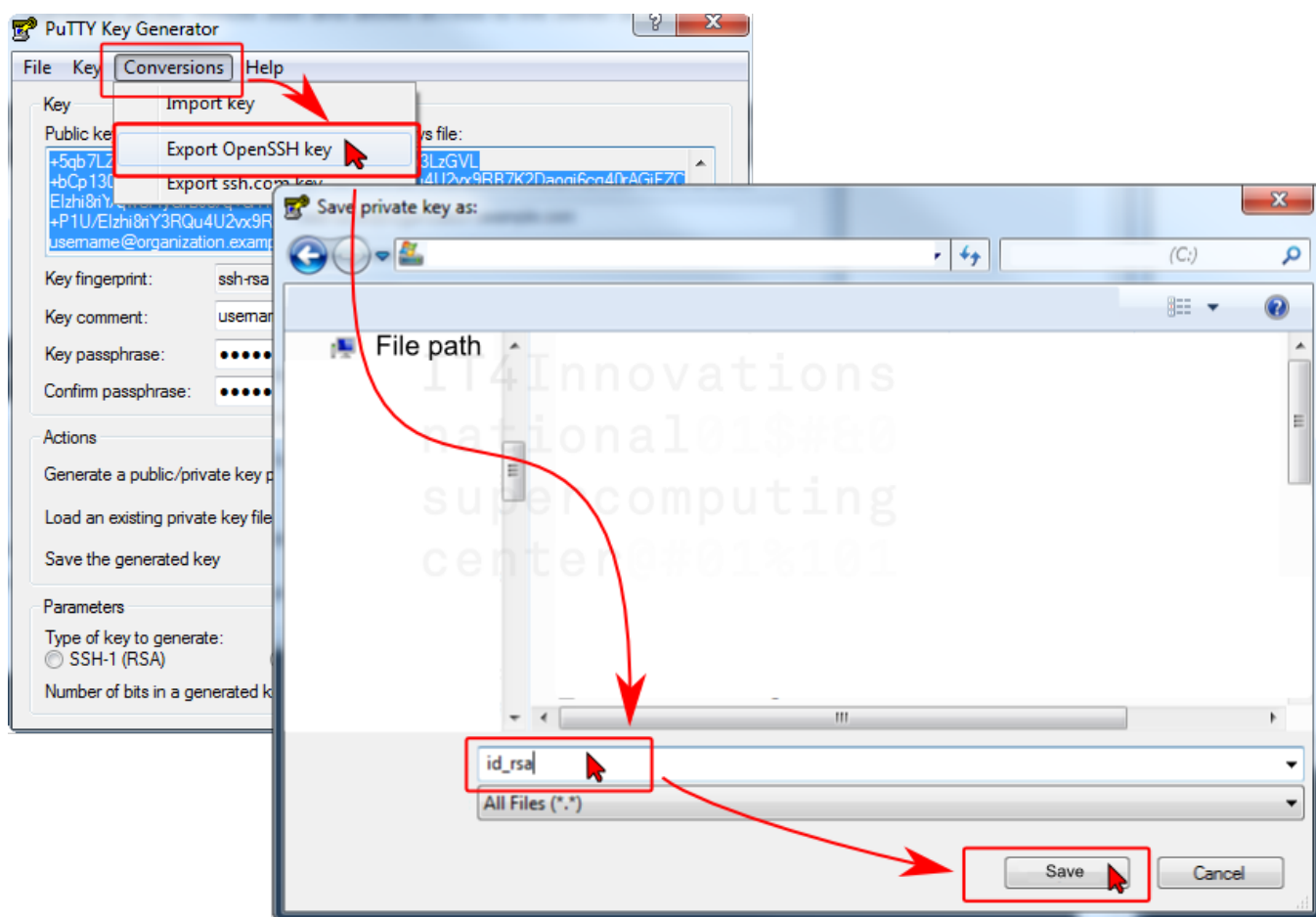


Figure 7:

PuTTY

PuTTY - before we start SSH connection

Windows PuTTY Installer

We recommend you to download “**A Windows installer for everything except PuTTYtel**” with **Pageant** (SSH authentication agent) and **PuTTYgen** (PuTTY key generator) which is available here [🔗](#).

!!! Note “Note” After installation you can proceed directly to private keys authentication using “Putty”.

"Change Password for Existing Private Key" is optional.

"Generate a New Public/Private key pair" is intended for users without Public/Private key in t

"Pageant" is optional.

PuTTYgen

PuTTYgen is the PuTTY key generator. Read more how to load in an existing private key and change your passphrase or generate a new public/private key pair using PuTTYgen if needed.

Pageant SSH agent

Pageant holds your private key in memory without needing to retype a passphrase on every login. We recommend its usage.

PuTTY - how to connect to the IT4Innovations cluster

- Run PuTTY
- Enter Host name and Save session fields with Login address and browse Connection - > SSH -> Auth menu. The *Host Name* input may be in the format “**username@clustername.it4i.cz**” so you don’t have to type your login each time. In this example we will connect to the Salomon cluster using “**salomon.it4i.cz**”.
- Category -> Connection - > SSH -> Auth: Select Attempt authentication using Pageant. Select Allow agent forwarding. Browse and select your private key file.
- Return to Session page and Save selected configuration with *Save* button.
- Now you can log in using *Open* button.
- Enter your username if the *Host Name* input is not in the format “username@salomon.it4i.cz”.
- Enter passphrase for selected private key file if Pageant **SSH authentication agent is not used**.

Another PuTTY Settings

- Category -> Windows -> Translation -> Remote character set and select **UTF-8**.

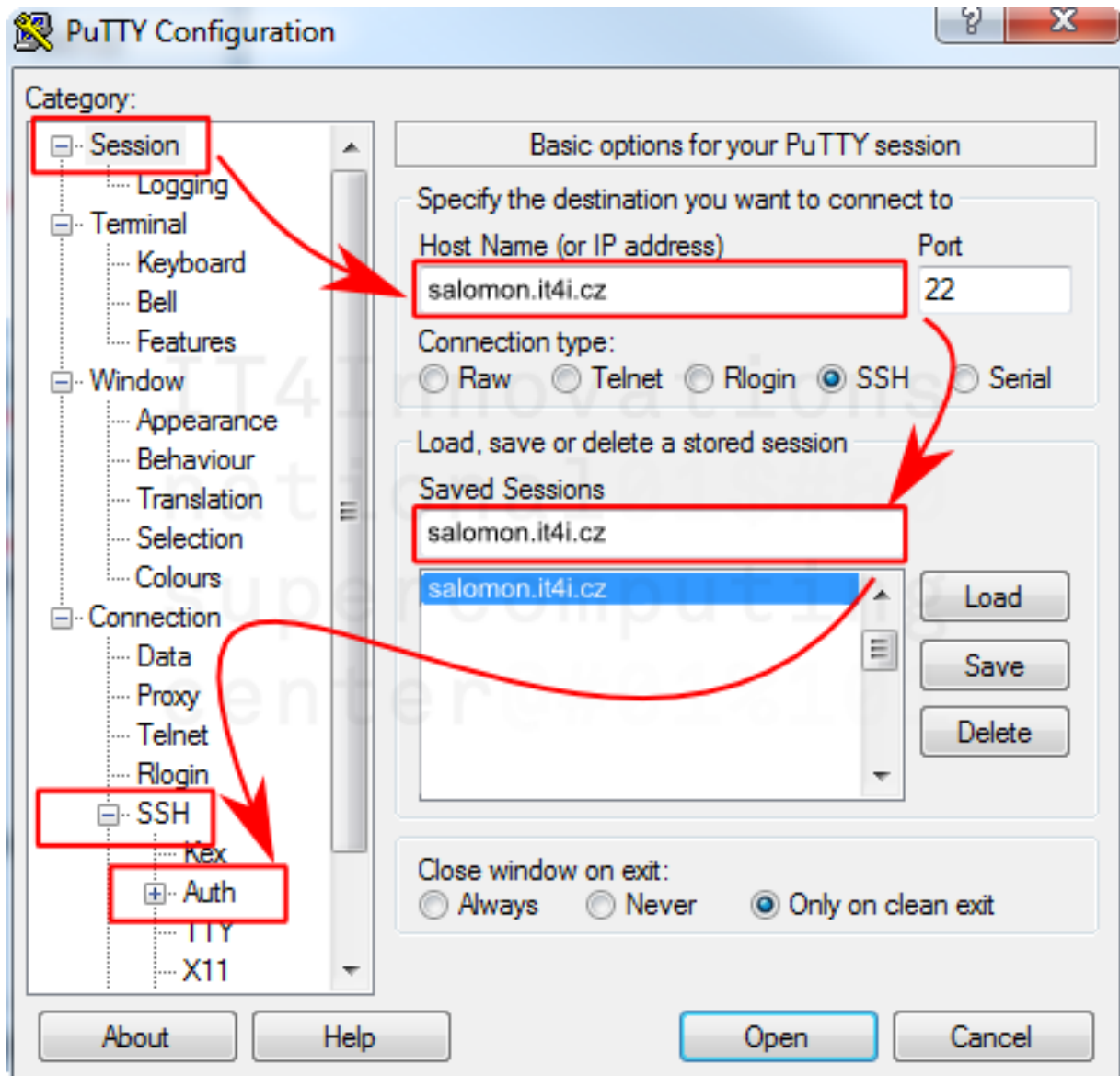


Figure 1:

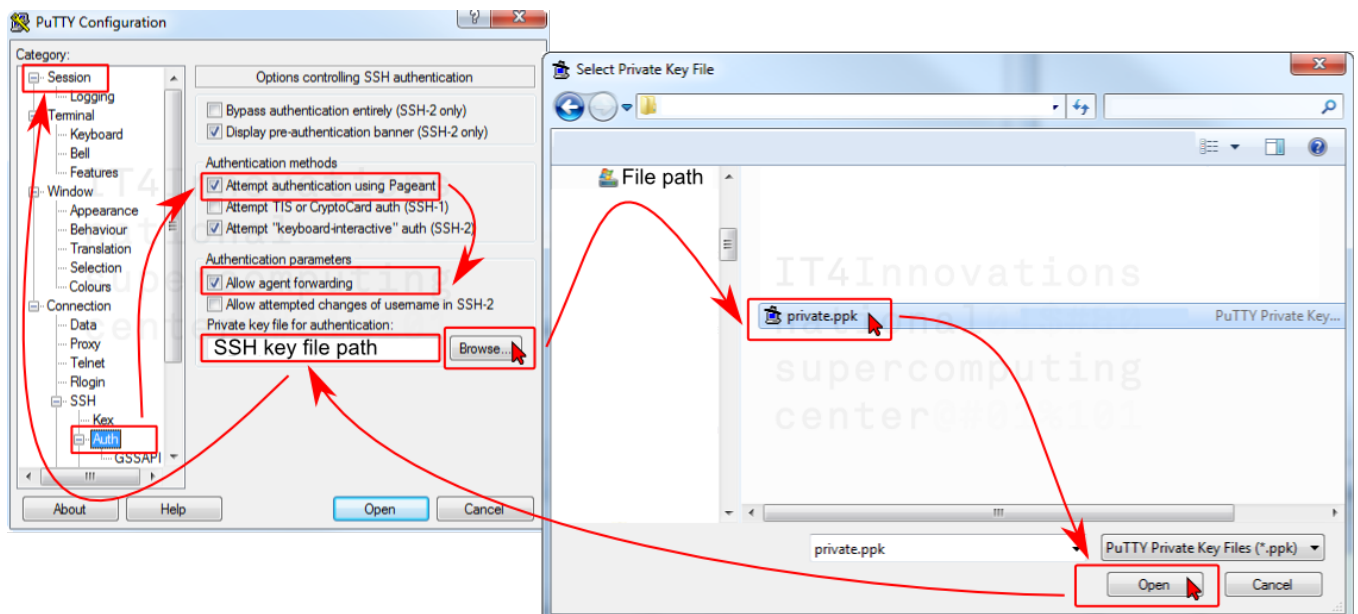


Figure 2:

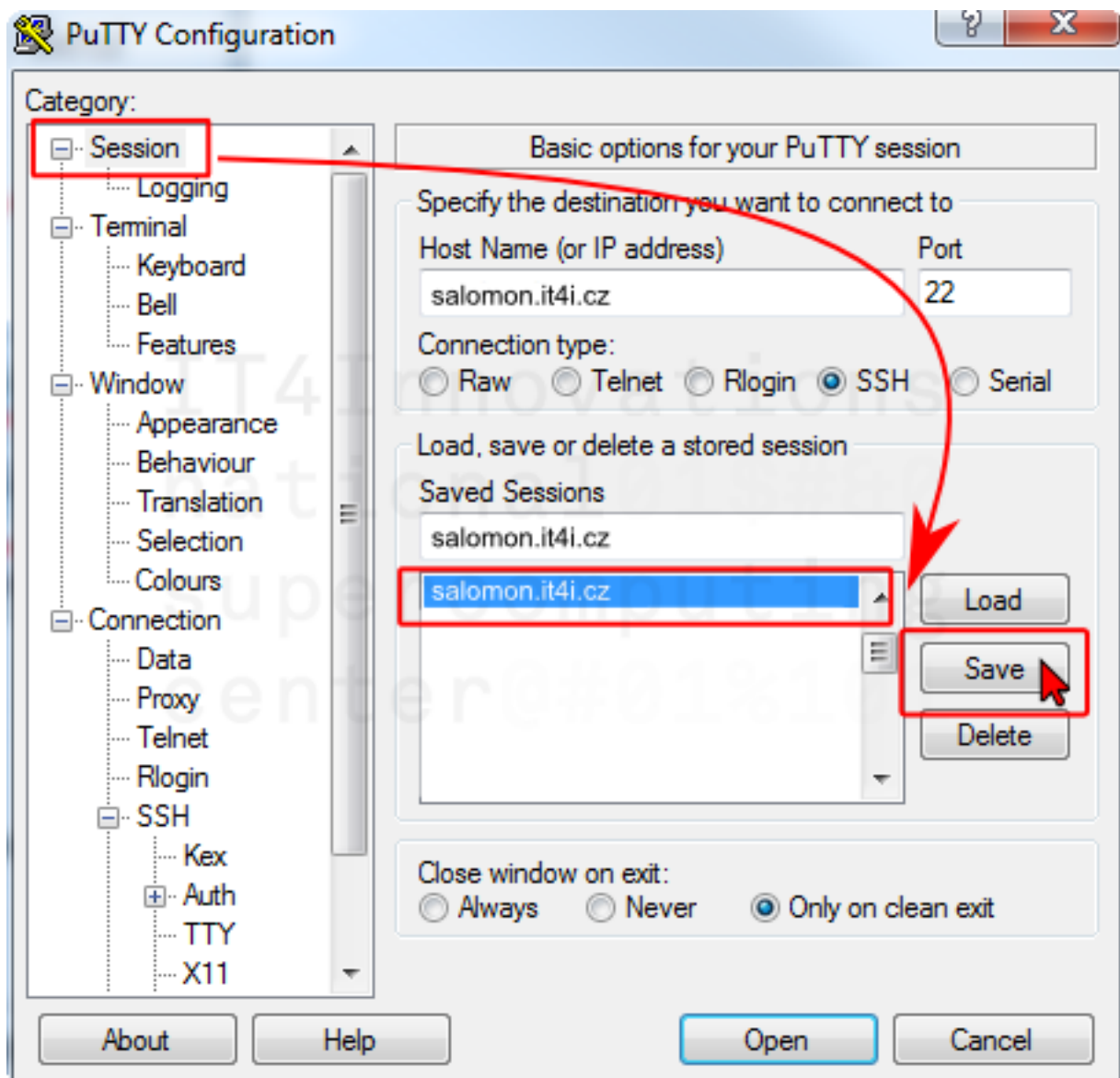


Figure 3:

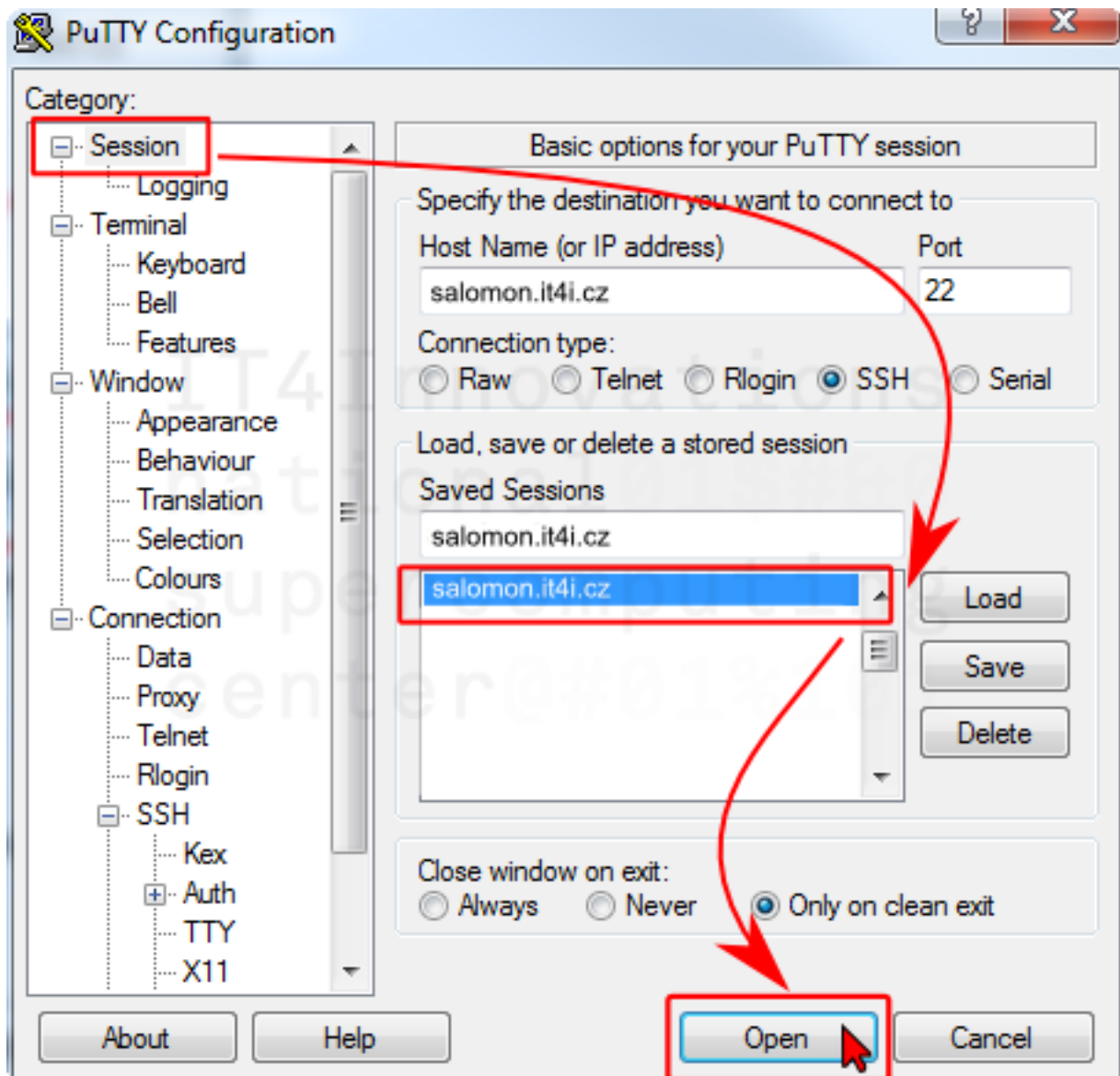


Figure 4:

- Category -> Terminal -> Features and select **Disable application keypad mode** (enable numpad)
- Save your configuration on Session page in to Default Settings with *Save* button.

SSH keys

Key management

After logging in, you can see .ssh/ directory with SSH keys and authorized_keys file:

```
$ cd /home/username/
$ ls -la .ssh/
total 24
drwx----- 2 username username 4096 May 13 15:12 .
drwxr-x---22 username username 4096 May 13 07:22 ..
-rw-r--r-- 1 username username  392 May 21 2014 authorized_keys
-rw----- 1 username username 1675 May 21 2014 id_rsa
-rw----- 1 username username 1460 May 21 2014 id_rsa.ppk
-rw-r--r-- 1 username username  392 May 21 2014 id_rsa.pub
```

!!! Note “Note” Please note that private keys in .ssh directory are without passphrase and allow you to connect within the cluster.

Access privileges on .ssh folder

- .ssh directory: 700 (drwx—)
- Authorized_keys, known_hosts and public key (.pub file): 644 (-rw-r--)
- Private key (id_rsa/id_rsa.ppk): 600 (-rw—)

```
cd /home/username/
chmod 700 .ssh/
chmod 644 .ssh/authorized_keys
chmod 644 .ssh/id_rsa.pub
chmod 644 .ssh/known_hosts
chmod 600 .ssh/id_rsa
chmod 600 .ssh/id_rsa.ppk
```

Private key

!!! Note “Note” The path to a private key is usually /home/username/.ssh/

Private key file in “id_rsa” or “*.ppk” format is used to authenticate with the servers. Private key is present locally on local side and used for example in SSH agent Pageant (for Windows users). The private key should always be kept in a safe place.

An example of private key format:

```
-----BEGIN RSA PRIVATE KEY-----
MIIEpAIBAAKCAQEAqbo7jokygnBpG2wYa5NB45ns6+UKTNMLHFOB03zmRtKEE1E
aGqXfbYwvX1cuRb2d9/Y5dVpCZHV0kbY3NhtV0cE1e+1R0aiU9BEsUAhMNEvgiLV
gSql4QvR04BWP1M8+WAWXDp3oeoBh8glXyuh9teb8yq98fv1r1peYGRrW3/s4V+q
01SQ0XY2T7rWCYRLIP6rTMXArTI35v3WU513mn7nm1fJ7oNOQgVH5b0W9V1Kyc4l
9vILHeMXxvz+i/5jTEfL0JpiRGYZYcaYrE4dIiHP13I1bV7hlkK23Xb1US8QJr5G
ADxp1VTkHjY+mKagEfx1lhQIb42JLHhKMEGqNQIDAQABAoIBAQCkypPuxZjL+vai
UGa5dAWiRZ46P2yrwHPKpvEdpCdDPbLAc1K/CtdBkHZsUPxNHVV6eFWweW99giIY
Av+mFWC58X8asBHQ7xkxmW0cqAZRzpkRA19IBS9/fKj028Fgy/p+su0i8oWbKIgJ
3LMkX0nnT9oz1AkOfTNC6Tv+3SE7eTj1RpCMjur4W1Cd1N3ELjLszdVk4tLx1XBS
```

```

y19NzVnJJbJR4t01145VfFECgYEAno1WJSB/SwdZvS9GkfhvmZd3r4vyV9Bmo3dn
XZA8HRW13imOnpk1DR4FRe98D9A7V3yh9h60Co4oAUd6N+0c68/qnv/809efA+M
/neI9ANYFo8F0+yFCp4Duj7zPV3aW1N/pd8TNzLqecqh10uZNMMy8rAjCxybeZjWd
DyhgYwXhAoGBAN3BCazNefYpLbpBQzwes+f2oStvw0YKDqySWsYVXeVgUI+OWTVZ
eZ26Y86E8MQ0+q0TIxpWou+TEaUg0SqCX40Q37rGS19K+rjnboJBYNcmwVp9bfyj
kCLL/3g57nTSqhghNa1xwemePvgNdn6FZteA8sXiCg5ZzaISqWAffek5AoGBAMPw
V/vwQ96C8E311cH5cUbmBCCcfXM2GLv74bb1V3SvCiAKg0rZ8gEgUiQ0+TfcbAbe
7MM20vRNQjaLTBpai/BTbmQm1Q+r1KNjq8k5bfTdAoGANGz1NM9omM10rd9WagL5
yuJcal/03p048mtB40I4Xr5ZJISHze8fK4jQ5veUT9Vu2Fy/w6QMsuRf+qWeCXR5
RPC2H0JzkS+2uZp8BOHk1iDPqbxWXJE9I57CxBV9C/tfzo2Iht00cuJ4LY+sw+y/
ocKpJbdLTwrTLdqLHwicdn80xeWot1mOukyK210UeDkY6H5pYPtHTpAZvRBd7ETL
Zs2RP3KFFvho6aIDGrY0wee740/jWotx7fbxxKwPyDRsbH3+1Wx/eX2RND40GdkH
gejJEzpk/7y/P/hCad7bSDdHZw0+Z03HIRCOE8yQz+JYatrQckaRCtd7cXryTmTR
FbvLJmECgYBDpfno2CzcFJCTdNBZF134oJRiDb+HdESXepk58PcNcgK3R8PXf+au
OqDBtZiUfv9U1WAg0gzGwt/0Y9u2c8m0nXziUS6AePxy5sBHs7g9C9WeZRz/nCWK
+cHIm7X0wBEzDKz5f9eBqRGipm0skDZNK18X/5QMTT5K3Eci2n+1Tw==
-----END RSA PRIVATE KEY-----

```

Public key

Public key file in “*.pub” format is used to verify a digital signature. Public key is present on the remote side and allows access to the owner of the matching private key.

An example of public key format:

```
ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAQACpuju0iTKCcGkbbBhrk0HjmeZr5QpM0swscXQE7f0ZG0oQSUROa
```

How to add your own key

First, generate a new keypair of your public and private key:

```
local $ ssh-keygen -C 'username@organization.example.com' -f additional_key
```

!!! Note “Note” Please, enter **strong passphrase** for securing your private key.

You can insert additional public key into `authorized_keys` file for authentication with your own private key. Additional records in `authorized_keys` file must be delimited by new line. Users are not advised to remove the default public key from `authorized_keys` file.

Example:

```
$ cat additional_key.pub > ~/.ssh/authorized_keys
```

In this example, we add an additional public key, stored in file `additional_key.pub` into the `authorized_keys`. Next time we log in, we will be able to use the private `additional_key` key to log in.

How to remove your own key

Removing your key from `authorized_keys` can be done simply by deleting the corresponding public key which can be identified by a comment at the end of line (eg. `username@organization.example.com`).

Compute Nodes

Nodes Configuration

Anselm is cluster of x86-64 Intel based nodes built on Bull Extreme Computing bullx technology. The cluster contains four types of compute nodes.

Compute Nodes Without Accelerator

- 180 nodes
- 2880 cores in total
- two Intel Sandy Bridge E5-2665, 8-core, 2.4GHz processors per node
- 64 GB of physical memory per node
- one 500GB SATA 2,5" 7,2 krpm HDD per node
- bullx B510 blade servers
- cn[1-180]

Compute Nodes With GPU Accelerator

- 23 nodes
- 368 cores in total
- two Intel Sandy Bridge E5-2470, 8-core, 2.3GHz processors per node
- 96 GB of physical memory per node
- one 500GB SATA 2,5" 7,2 krpm HDD per node
- GPU accelerator 1x NVIDIA Tesla Kepler K20 per node
- bullx B515 blade servers
- cn[181-203]

Compute Nodes With MIC Accelerator

- 4 nodes
- 64 cores in total
- two Intel Sandy Bridge E5-2470, 8-core, 2.3GHz processors per node
- 96 GB of physical memory per node
- one 500GB SATA 2,5" 7,2 krpm HDD per node
- MIC accelerator 1x Intel Phi 5110P per node
- bullx B515 blade servers
- cn[204-207]

Fat Compute Nodes

- 2 nodes
- 32 cores in total
- 2 Intel Sandy Bridge E5-2665, 8-core, 2.4GHz processors per node
- 512 GB of physical memory per node
- two 300GB SAS 3,5"15krpm HDD (RAID1) per node
- two 100GB SLC SSD per node
- bullx R423-E3 servers
- cn[208-209]



Figure 1:

Figure Anselm bullx B510 servers

Compute Nodes Summary

Node type	Count	Range	Memory	Cores	Access
Nodes without accelerator	180	cn[1-180]	64GB	16 @ 2.4Ghz	qexp, qprod, qlong, qfree
Nodes with GPU accelerator	23	cn[181-203]	96GB	16 @ 2.3Ghz	qgpu, qprod
Nodes with MIC accelerator	4	cn[204-207]	96GB	16 @ 2.3Ghz	qmic, qprod
Fat compute nodes	2	cn[208-209]	512GB	16 @ 2.4Ghz	qfat, qprod

Processor Architecture

Anselm is equipped with Intel Sandy Bridge processors Intel Xeon E5-2665 (nodes without accelerator and fat nodes) and Intel Xeon E5-2470 (nodes with accelerator). Processors support Advanced Vector Extensions (AVX) 256-bit instruction set.

- L2: 256 KB per core
- L3: 20 MB per processor
- memory bandwidth at the level of the processor: 51.2 GB/s

Intel Sandy Bridge E5-2470 Processor

- eight-core
- speed: 2.3 GHz, up to 3.1 GHz using Turbo Boost Technology
- peak performance: 18.4 Gflop/s per core
- caches:
 - L2: 256 KB per core
 - L3: 20 MB per processor
- memory bandwidth at the level of the processor: 38.4 GB/s

Nodes equipped with Intel Xeon E5-2665 CPU have set PBS resource attribute `cpu_freq = 24`, nodes equipped with Intel Xeon E5-2470 CPU have set PBS resource attribute `cpu_freq = 23`.

```
$ qsub -A OPEN-0-0 -q qprod -l select=4:ncpus=16:cpu_freq=24 -I
```

In this example, we allocate 4 nodes, 16 cores at 2.4GHz per node.

Intel Turbo Boost Technology is used by default, you can disable it for all nodes of job by using resource attribute `cpu_turbo_boost`.

```
$ qsub -A OPEN-0-0 -q qprod -l select=4:ncpus=16 -l cpu_turbo_boost=0 -I
```

Memory Architecture

Compute Node Without Accelerator

- 2 sockets
- Memory Controllers are integrated into processors.
 - 8 DDR3 DIMMS per node
 - 4 DDR3 DIMMS per CPU
 - 1 DDR3 DIMMS per channel
 - Data rate support: up to 1600MT/s
- Populated memory: 8x 8GB DDR3 DIMM 1600Mhz

Compute Node With GPU or MIC Accelerator

- 2 sockets
- Memory Controllers are integrated into processors.
 - 6 DDR3 DIMMS per node
 - 3 DDR3 DIMMS per CPU
 - 1 DDR3 DIMMS per channel
 - Data rate support: up to 1600MT/s
- Populated memory: 6x 16GB DDR3 DIMM 1600Mhz

Fat Compute Node

- 2 sockets
- Memory Controllers are integrated into processors.

- 16 DDR3 DIMMS per node
 - 8 DDR3 DIMMS per CPU
 - 2 DDR3 DIMMS per channel
 - Data rate support: up to 1600MT/s
- Populated memory: 16x 32GB DDR3 DIMM 1600Mhz

Outgoing connections

Connection restrictions

Outgoing connections, from Anselm Cluster login nodes to the outside world, are restricted to following ports:

Port	Protocol
22	ssh
80	http
443	https
9418	git

!!! Note “Note” Please use **ssh port forwarding** and proxy servers to connect from Anselm to all other remote ports.

Outgoing connections, from Anselm Cluster compute nodes are restricted to the internal network. Direct connections from compute nodes to outside world are cut.

Port forwarding

Port forwarding from login nodes

!!! Note “Note” Port forwarding allows an application running on Anselm to connect to arbitrary remote host and port.

It works by tunneling the connection from Anselm back to users workstation and forwarding from the workstation to the remote host.

Pick some unused port on Anselm login node (for example 6000) and establish the port forwarding:

```
local $ ssh -R 6000:remote.host.com:1234 anselm.it4i.cz
```

In this example, we establish port forwarding between port 6000 on Anselm and port 1234 on the remote.host.com. By accessing localhost:6000 on Anselm, an application will see response of remote.host.com:1234. The traffic will run via users local workstation.

Port forwarding may be done **using PuTTY** as well. On the PuTTY Configuration screen, load your Anselm configuration first. Then go to Connection->SSH->Tunnels to set up the port forwarding. Click Remote radio button. Insert 6000 to Source port textbox. Insert remote.host.com:1234. Click Add button, then Open.

Port forwarding may be established directly to the remote host. However, this requires that user has ssh access to remote.host.com

```
$ ssh -L 6000:localhost:1234 remote.host.com
```

Note: Port number 6000 is chosen as an example only. Pick any free port.

Port forwarding from compute nodes

Remote port forwarding from compute nodes allows applications running on the compute nodes to access hosts outside Anselm Cluster.

First, establish the remote port forwarding from the login node, as described above.

Second, invoke port forwarding from the compute node to the login node. Insert following line into your jobscript or interactive shell

```
$ ssh -TN -f -L 6000:localhost:6000 login1
```

In this example, we assume that port forwarding from login1:6000 to remote.host.com:1234 has been established beforehand. By accessing localhost:6000, an application running on a compute node will see response of remote.host.com:1234


Using proxy servers

Port forwarding is static, each single port is mapped to a particular port on remote host. Connection to other remote host, requires new forward.

!!! Note “Note” Applications with inbuilt proxy support, experience unlimited access to remote hosts, via single proxy server.

To establish local proxy server on your workstation, install and run SOCKS proxy server software. On Linux, sshd demon provides the functionality. To establish SOCKS proxy server listening on port 1080 run:

```
local $ ssh -D 1080 localhost
```

On Windows, install and run the free, open source Sock Puppet  server.

Once the proxy server is running, establish ssh port forwarding from Anselm to the proxy server, port 1080, exactly as described above.

```
local $ ssh -R 6000:localhost:1080 anselm.it4i.cz
```

Now, configure the applications proxy settings to **localhost:6000**. Use port forwarding to access the proxy server from compute nodes as well .

VPN Access

Accessing IT4Innovations internal resources via VPN

!!! Note “Note” **Failed to initialize connection subsystem Win 8.1 - 02-10-15 MS patch**

Workaround can be found at [vpn-connection-fail-in-win-8.1](../../vpn-connection-fail-in-win-8.1)

For using resources and licenses which are located at IT4Innovations local network, it is necessary to VPN connect to this network. We use Cisco AnyConnect Secure Mobility Client, which is supported on the following operating systems:

- Windows XP
- Windows Vista
- Windows 7
- Windows 8
- Linux
- MacOS

It is impossible to connect to VPN from other operating systems.

VPN client installation

You can install VPN client from web interface after successful login with LDAP credentials on address <https://vpn1.it4i.cz/anselm>

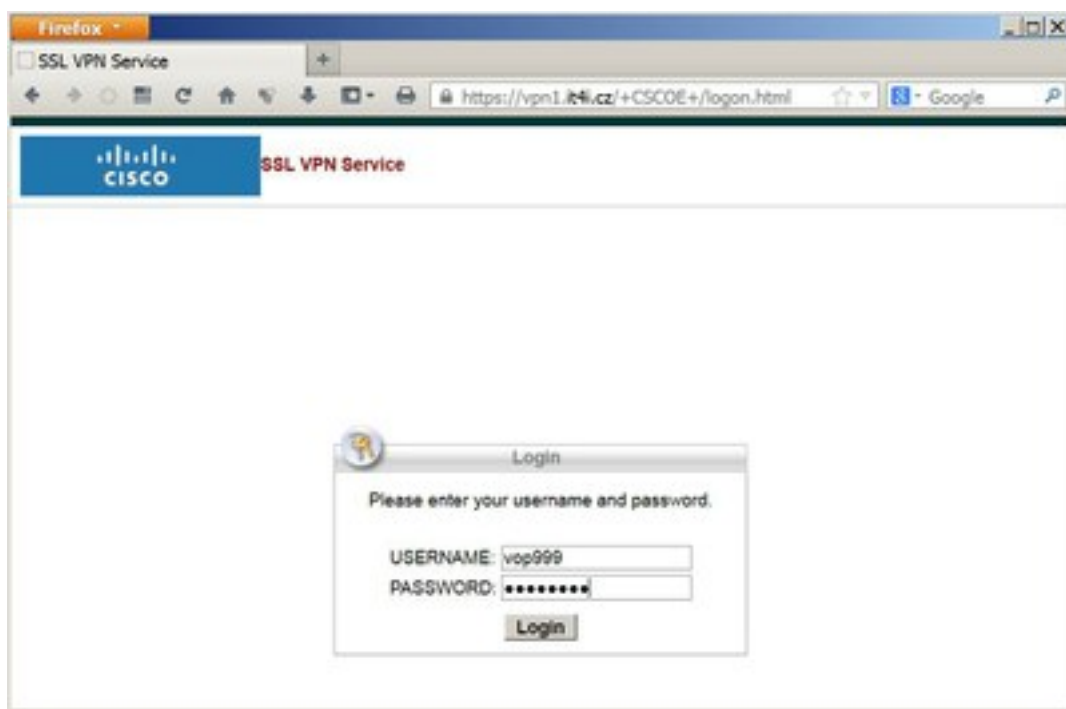


Figure 1:

According to the Java settings after login, the client either automatically installs, or downloads installation file for your operating system. It is necessary to allow start of installation tool for automatic installation.

After successful installation, VPN connection will be established and you can use available resources from IT4I network.



Figure 2:



Figure 3:



Figure 4:



Figure 5:

If your Java setting doesn't allow automatic installation, you can download installation file and install VPN client manually.



Figure 6:

After you click on the link, download of installation file will start.

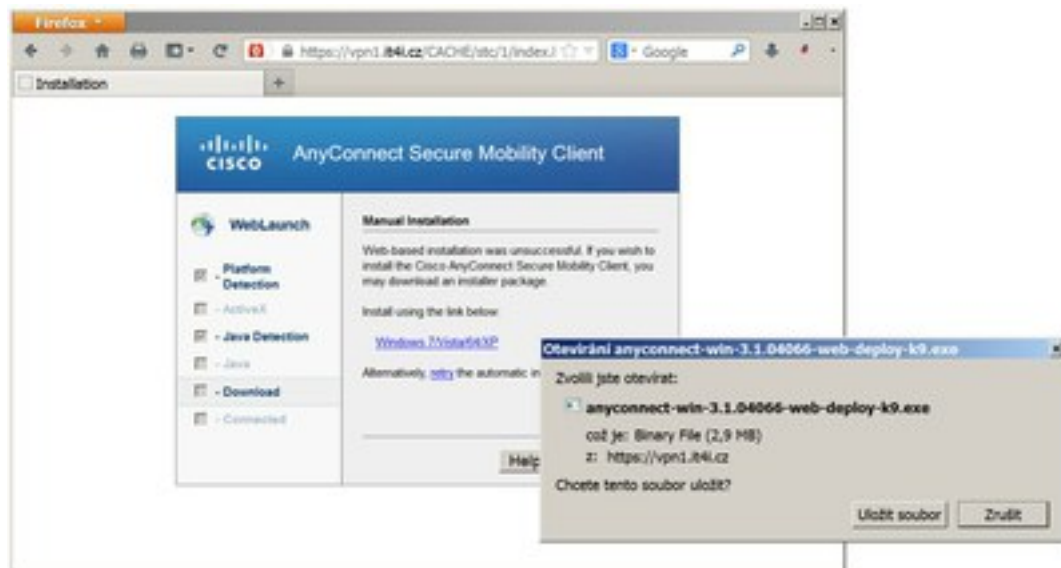


Figure 7:

After successful download of installation file, you have to execute this tool with administrator's rights and install VPN client manually.

Working with VPN client

You can use graphical user interface or command line interface to run VPN client on all supported operating systems. We suggest using GUI.

Before the first login to VPN, you have to fill URL **https://vpn1.it4i.cz/anselm** into the text field.

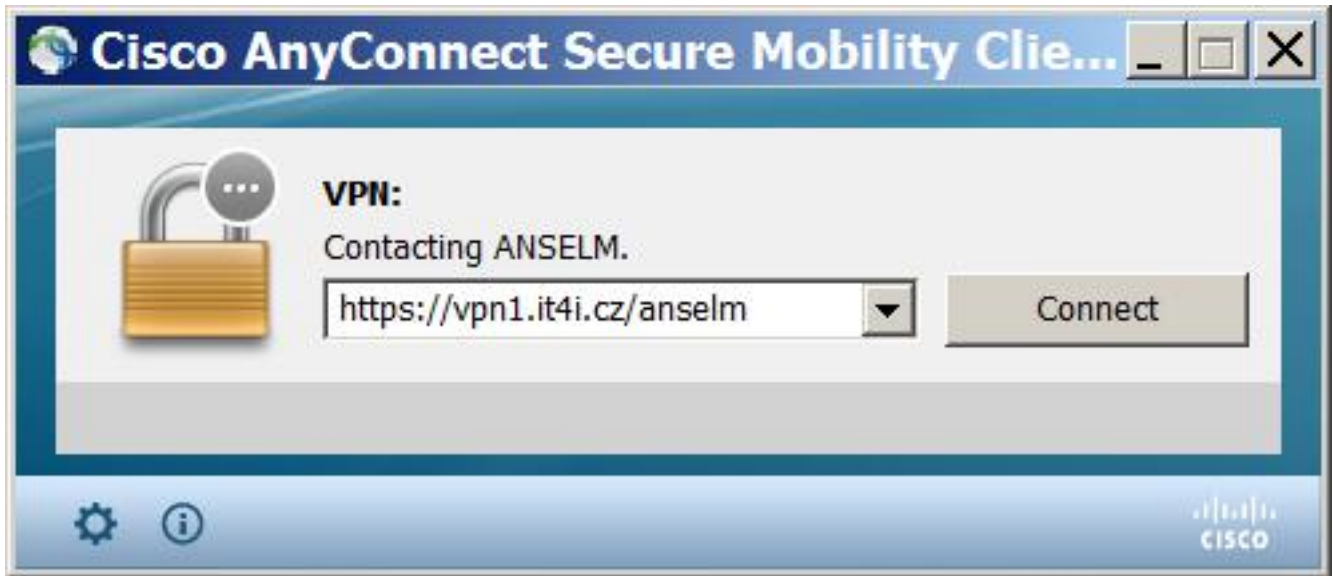


Figure 8:

After you click on the Connect button, you must fill your login credentials.

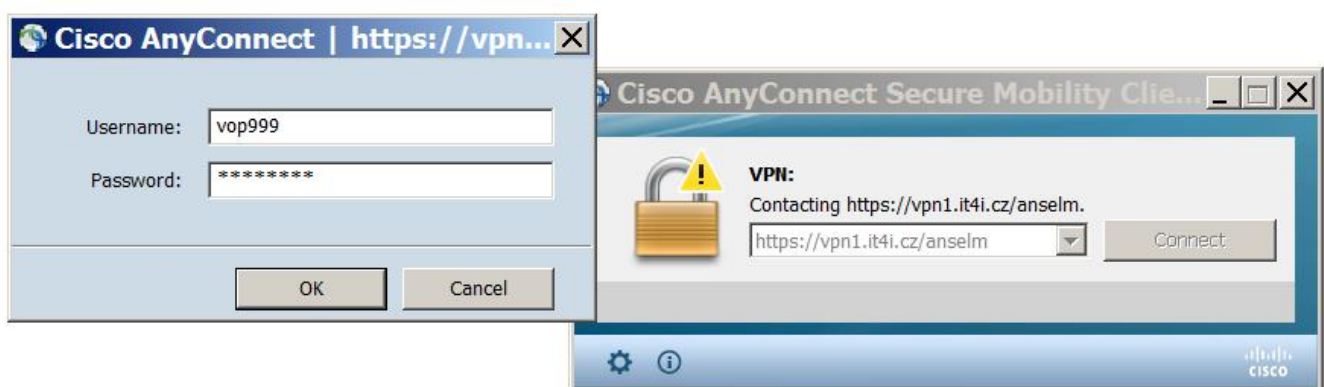


Figure 9:

After a successful login, the client will minimize to the system tray. If everything works, you can see a lock in the Cisco tray icon.

If you right-click on this icon, you will see a context menu in which you can control the VPN connection.

When you connect to the VPN for the first time, the client downloads the profile and creates a new item “ANSELM” in the connection list. For subsequent connections, it is not necessary to re-enter the URL address, but just select the corresponding item.

Then AnyConnect automatically proceeds like in the case of first login.

After a successful login, you can see a green circle with a tick mark on the lock icon.

For disconnecting, right-click on the AnyConnect client icon in the system tray and select **VPN Disconnect**.

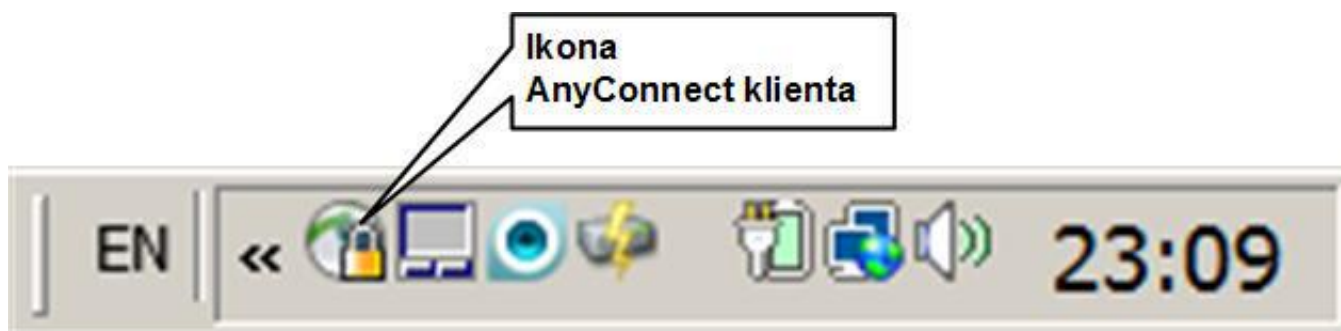


Figure 10:

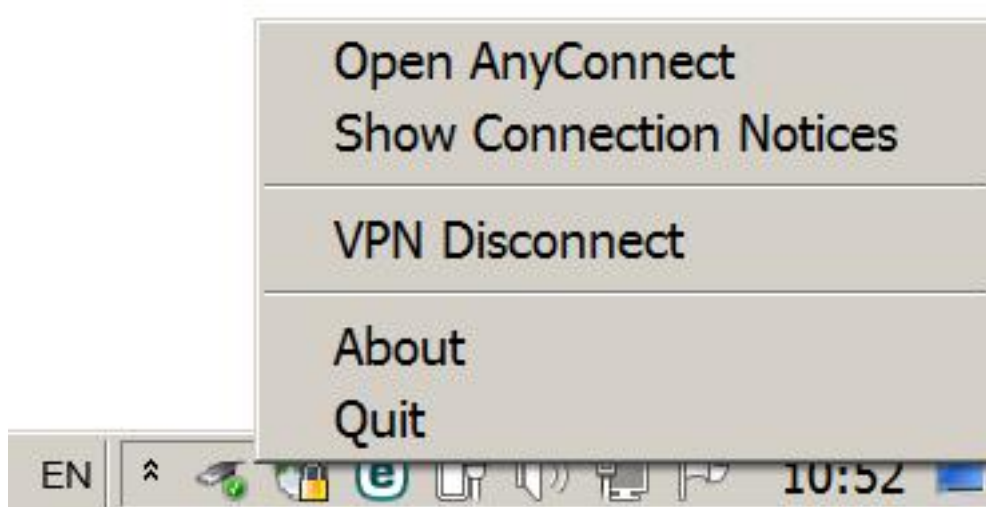


Figure 11:

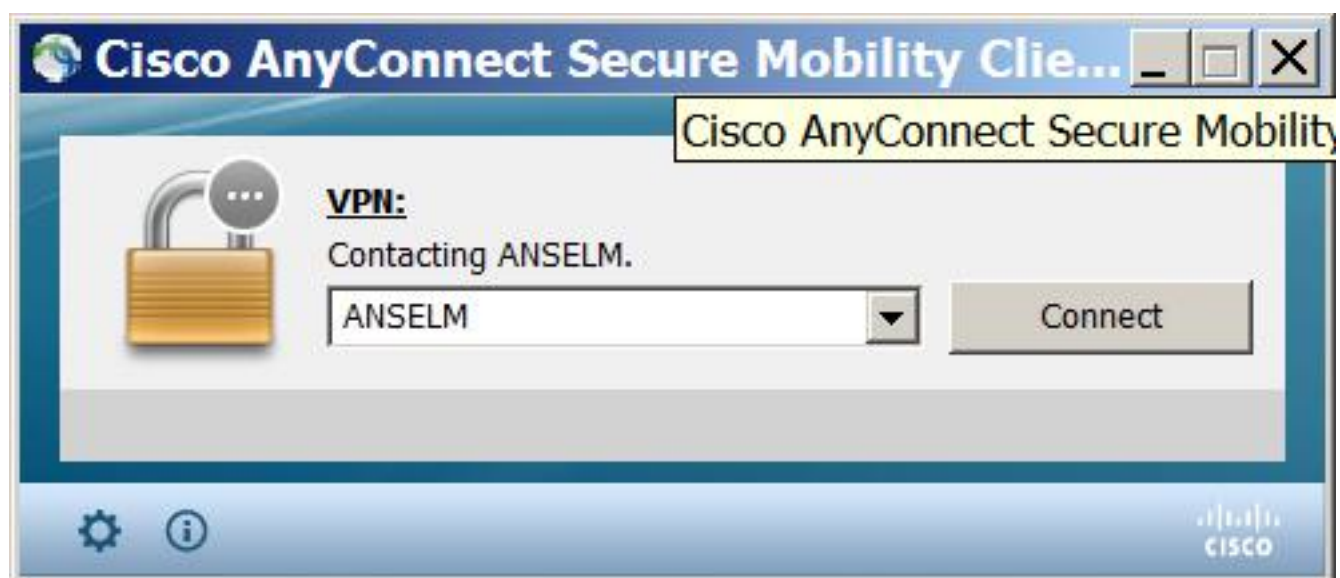


Figure 12:



Figure 13:

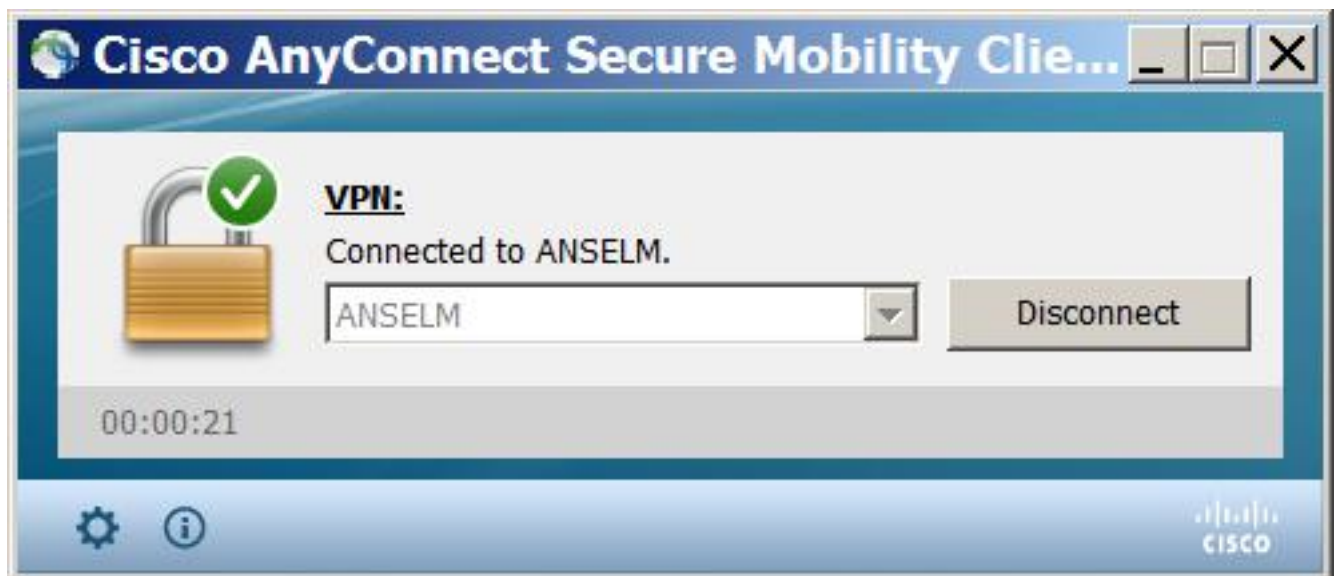


Figure 14:

Shell access and data transfer

Interactive Login

The Anselm cluster is accessed by SSH protocol via login nodes login1 and login2 at address anselm.it4i.cz. The login nodes may be addressed specifically, by prepending the login node name to the address.

Login address	Port
anselm.it4i.cz	22
login1.anselm.it4i.cz	22
login2.anselm.it4i.cz	22

The authentication is by the private key

!!! Note “Note” Please verify SSH fingerprints during the first logon. They are identical on all login nodes:

29:b3:f4:64:b0:73:f5:6f:a7:85:0f:e0:0d:be:76:bf (DSA)
d4:6f:5c:18:f4:3f:70:ef:bc:fc:cc:2b:fd:13:36:b7 (RSA)

Private key authentication:

On **Linux** or **Mac**, use

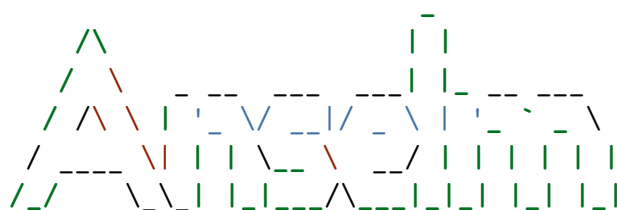
```
local $ ssh -i /path/to/id_rsa username@anselm.it4i.cz
```

If you see warning message “UNPROTECTED PRIVATE KEY FILE!”, use this command to set lower permissions to private key file.

```
local $ chmod 600 /path/to/id_rsa
```

On **Windows**, use PuTTY ssh client.

After logging in, you will see the command prompt:




```
http://www.it4i.cz/?lang=en
```

```
Last login: Tue Jul  9 15:57:38 2013 from your-host.example.com  
[username@login2.anselm ~]$
```

The environment is **not** shared between login nodes, except for shared filesystems.

Data Transfer

Data in and out of the system may be transferred by the scp  and sftp protocols. (Not available yet.) In case large volumes of data are transferred, use dedicated data mover node dm1.anselm.it4i.cz for increased performance.

Address	Port
anselm.it4i.cz	22
login1.anselm.it4i.cz	22
login2.anselm.it4i.cz	22
dm1.anselm.it4i.cz	22

The authentication is by the private key

!!! Note “Note” Data transfer rates up to **160MB/s** can be achieved with scp or sftp.

1TB may be transferred in 1:50h.

To achieve 160MB/s transfer rates, the end user must be connected by 10G line all the way to IT4Innovations and use computer with fast processor for the transfer. Using Gigabit ethernet connection, up to 110MB/s may be expected. Fast cipher (aes128-ctr) should be used.

!!! Note “Note” If you experience degraded data transfer performance, consult your local network provider.

On linux or Mac, use scp or sftp client to transfer the data to Anselm:

```
local $ scp -i /path/to/id_rsa my-local-file username@anselm.it4i.cz:directory/file
local $ scp -i /path/to/id_rsa -r my-local-dir username@anselm.it4i.cz:directory
```

or

```
local $ sftp -o IdentityFile=/path/to/id_rsa username@anselm.it4i.cz
```



Very convenient way to transfer files in and out of the Anselm computer is via the fuse filesystem sshfs

```
local $ sshfs -o IdentityFile=/path/to/id_rsa username@anselm.it4i.cz:. mountpoint
```

Using sshfs, the users Anselm home directory will be mounted on your local computer, just like an external disk.

Learn more on ssh, scp and sshfs by reading the manpages



```
$ man ssh
$ man scp
$ man sshfs
```

On Windows, use WinSCP client  to transfer the data. The win-sshfs client  provides a way to mount the Anselm filesystems directly as an external disc.

More information about the shared file systems is available [here](#).

Introduction

Welcome to Anselm supercomputer cluster. The Anselm cluster consists of 209 compute nodes, totaling 3344 compute cores with 15TB RAM and giving over 94 Tflop/s theoretical peak performance. Each node is a powerful x86-64 computer, equipped with 16 cores, at least 64GB RAM, and 500GB harddrive. Nodes are interconnected by fully non-blocking fat-tree Infiniband network and equipped with Intel Sandy Bridge processors. A few nodes are also equipped with NVIDIA Kepler GPU or Intel Xeon Phi MIC accelerators. Read more in Hardware Overview.

The cluster runs bullx Linux (bull ) operating system, which is compatible with the RedHat Linux family.  We have installed a wide range of software packages targeted at different scientific domains. These packages are accessible via the modules environment.

User data shared file-system (HOME, 320TB) and job data shared file-system (SCRATCH, 146TB) are available to users.

The PBS Professional workload manager provides computing resources allocations and job execution.

Read more on how to apply for resources, obtain login credentials, and access the cluster.

Compilers

Available compilers, including GNU, INTEL and UPC compilers

Currently there are several compilers for different programming languages available on the Anselm cluster:

- C/C++
- Fortran 77/90/95
- Unified Parallel C
- Java
- nVidia CUDA

The C/C++ and Fortran compilers are divided into two main groups GNU and Intel.

Intel Compilers

For information about the usage of Intel Compilers and other Intel products, please read the Intel Parallel studio page.

GNU C/C++ and Fortran Compilers

For compatibility reasons there are still available the original (old 4.4.6-4) versions of GNU compilers as part of the OS. These are accessible in the search path by default.

It is strongly recommended to use the up to date version (4.8.1) which comes with the module gcc:

```
$ module load gcc
$ gcc -v
$ g++ -v
$ gfortran -v
```

With the module loaded two environment variables are predefined. One for maximum optimizations on the Anselm cluster architecture, and the other for debugging purposes:

```
$ echo $OPTFLAGS
-O3 -march=corei7-avx

$ echo $DEBUGFLAGS
-O0 -g
```

For more informations about the possibilities of the compilers, please see the man pages.

Unified Parallel C

UPC is supported by two compiler/runtime implementations:

- GNU - SMP/multi-threading support only
- Berkley - multi-node support as well as SMP/multi-threading support

GNU UPC Compiler

To use the GNU UPC compiler and run the compiled binaries use the module gupc

```
$ module add gupc
$ gupc -v
$ g++ -v
```

Simple program to test the compiler

```
$ cat count.upc

/* hello.upc - a simple UPC example */
#include <upc.h>
#include <stdio.h>

int main() {
    if (MYTHREAD == 0) {
        printf("Welcome to GNU UPC!!!n");
    }
    upc_barrier;
    printf(" - Hello from thread %in", MYTHREAD);
    return 0;
}
```

To compile the example use

```
$ gupc -o count.upc.x count.upc
```

To run the example with 5 threads issue

```
$ ./count.upc.x -fupc-threads-5
```

For more informations see the man pages.

Berkley UPC Compiler

To use the Berkley UPC compiler and runtime environment to run the binaries use the module bupc

```
$ module add bupc
$ upcc -version
```

As default UPC network the “smp” is used. This is very quick and easy way for testing/debugging, but limited to one node only.

For production runs, it is recommended to use the native Infiniband implementation of UPC network “ibv”. For testing/debugging using multiple nodes, the “mpi” UPC network is recommended. Please note, that **the selection of the network is done at the compile time** and not at runtime (as expected)!

Example UPC code:

```
$ cat hello.upc

/* hello.upc - a simple UPC example */
#include <upc.h>
#include <stdio.h>
```

```

int main() {
    if (MYTHREAD == 0) {
        printf("Welcome to Berkeley UPC!!!n");
    }
    upc_barrier;
    printf(" - Hello from thread %in", MYTHREAD);
    return 0;
}

```

To compile the example with the “ibv” UPC network use

```
$ upcc -network=ibv -o hello.upc.x hello.upc
```

To run the example with 5 threads issue

```
$ upcrun -n 5 ./hello.upc.x
```

To run the example on two compute nodes using all 32 cores, with 32 threads, issue

```

$ qsub -I -q qprod -A PROJECT_ID -l select=2:ncpus=16
$ module add bupc
$ upcrun -n 32 ./hello.upc.x

```

For more informations see the man pages.

Java

For information how to use Java (runtime and/or compiler), please read the Java page.

nVidia CUDA

For information how to work with nVidia CUDA, please read the nVidia CUDA page.

GPI-2

A library that implements the GASPI specification

Introduction

Programming Next Generation Supercomputers: GPI-2 is an API library for asynchronous inter-process, cross-node communication. It provides a flexible, scalable and fault tolerant interface for parallel applications.

The GPI-2 library (www.gpi-site.com/gpi2/) implements the GASPI specification (Global Address Space Programming Interface, www.gaspi.de). GASPI is a Partitioned Global Address Space (PGAS) API. It aims at scalable, flexible and failure tolerant computing in massively parallel environments.

Modules

The GPI-2, version 1.0.2 is available on Anselm via module gpi2:

```
$ module load gpi2
```

The module sets up environment variables, required for linking and running GPI-2 enabled applications. This particular command loads the default module, which is gpi2/1.0.2

Linking

!!! Note “Note” Link with -lGPI2 -libverbs

Load the gpi2 module. Link using **-lGPI2** and **-libverbs** switches to link your code against GPI-2. The GPI-2 requires the OFED infiniband communication library ibverbs.

Compiling and linking with Intel compilers

```
$ module load intel
$ module load gpi2
$ icc myprog.c -o myprog.x -Wl,-rpath=$LIBRARY_PATH -lGPI2 -libverbs
```

Compiling and linking with GNU compilers

```
$ module load gcc
$ module load gpi2
$ gcc myprog.c -o myprog.x -Wl,-rpath=$LIBRARY_PATH -lGPI2 -libverbs
```

Running the GPI-2 codes

!!! Note “Note” gaspi_run starts the GPI-2 application

The gaspi_run utility is used to start and run GPI-2 applications:

```
$ gaspi_run -m machinefile ./myprog.x
```

A machine file (**machinefile**) with the hostnames of nodes where the application will run, must be provided. The machinefile lists all nodes on which to run, one entry per node per process. This file may be hand created or obtained from standard \$PBS_NODEFILE:

```
$ cut -f1 -d"." $PBS_NODEFILE > machinefile
```

machinefile:

```
cn79
cn80
```

This machinefile will run 2 GPI-2 processes, one on node cn79 other on node cn80.

machinefle:

```
cn79
cn79
cn80
cn80
```

This machinefile will run 4 GPI-2 processes, 2 on node cn79 o 2 on node cn80.

!!! Note “Note” Use the **mpiprocs** to control how many GPI-2 processes will run per node

Example:

```
$ qsub -A OPEN-0-0 -q qexp -l select=2:ncpus=16:mpiprocs=16 -I
```

This example will produce \$PBS_NODEFILE with 16 entries per node.

gaspi_logger

!!! Note “Note” gaspi_logger views the output form GPI-2 application ranks

The gaspi_logger utility is used to view the output from all nodes except the master node (rank 0). The gaspi_logger is started, on another session, on the master node - the node where the gaspi_run is executed. The output of the application, when called with gaspi_printf(), will be redirected to the gaspi_logger. Other I/O routines (e.g. printf) will not.

Example

Following is an example GPI-2 enabled code:

```
#include <GASPI.h>
#include <stdlib.h>

void success_or_exit ( const char* file, const int line, const int ec)
{
    if (ec != GASPI_SUCCESS)
    {
        gaspi_printf ("Assertion failed in %s[%i]:%dn", file, line, ec);
        exit (1);
    }
}

#define ASSERT(ec) success_or_exit (__FILE__, __LINE__, ec);
```

```

int main(int argc, char *argv[])
{
    gaspi_rank_t rank, num;
    gaspi_return_t ret;

    /* Initialize GPI-2 */
    ASSERT( gaspi_proc_init(GASPI_BLOCK) );

    /* Get ranks information */
    ASSERT( gaspi_proc_rank(&rank) );
    ASSERT( gaspi_proc_num(&num) );

    gaspi_printf("Hello from rank %d of %dn",
                 rank, num);

    /* Terminate */
    ASSERT( gaspi_proc_term(GASPI_BLOCK) );

    return 0;
}

```

Load modules and compile:

```

$ module load gcc gpi2
$ gcc helloworld_gpi.c -o helloworld_gpi.x -Wl,-rpath=$LIBRARY_PATH -lGPI2 -libverbs

```

Submit the job and run the GPI-2 application

```

$ qsub -q qexp -l select=2:ncpus=1:mpiprocs=1,place=scatter,walltime=00:05:00 -I
qsub: waiting for job 171247.dm2 to start
qsub: job 171247.dm2 ready

cn79 $ module load gpi2
cn79 $ cut -f1 -d"." $PBS_NODEFILE > machinefile
cn79 $ gaspi_run -m machinefile ./helloworld_gpi.x
Hello from rank 0 of 2

```

At the same time, in another session, you may start the gaspi logger:

```

$ ssh cn79
cn79 $ gaspi_logger
GASPI Logger (v1.1)
[cn80:0] Hello from rank 1 of 2

```

In this example, we compile the helloworld_gpi.c code using the **gnu compiler** (gcc) and link it to the GPI-2 and iverbs library. The library search path is compiled in. For execution, we use the qexp queue, 2 nodes 1 core each. The GPI module must be loaded on the master compute node (in this example the cn79), gaspi_logger is used from different session to view the output of the second process.

Diagnostic component (TEAM)

Access

TEAM is available at the following address: <http://omics.it4i.cz/team/>

!!! Note “Note” The address is accessible only via VPN.

Diagnostic component (TEAM)

VCF files are scanned by this diagnostic tool for known diagnostic disease-associated variants. When no diagnostic mutation is found, the file can be sent to the disease-causing gene discovery tool to see whether new disease associated variants can be found.

TEAM (27) is an intuitive and easy-to-use web tool that fills the gap between the predicted mutations and the final diagnostic in targeted enrichment sequencing analysis. The tool searches for known diagnostic mutations, corresponding to a disease panel, among the predicted patient's variants. Diagnostic variants for the disease are taken from four databases of disease-related variants (HGMD-public, HUMSAVAR, ClinVar and COSMIC). If no primary diagnostic variant is found, then a list of secondary findings that can help to establish a diagnostic is produced. TEAM also provides with an interface for the definition and customization of panels, by means of which, genes and mutations can be added or discarded to adjust panel definitions.

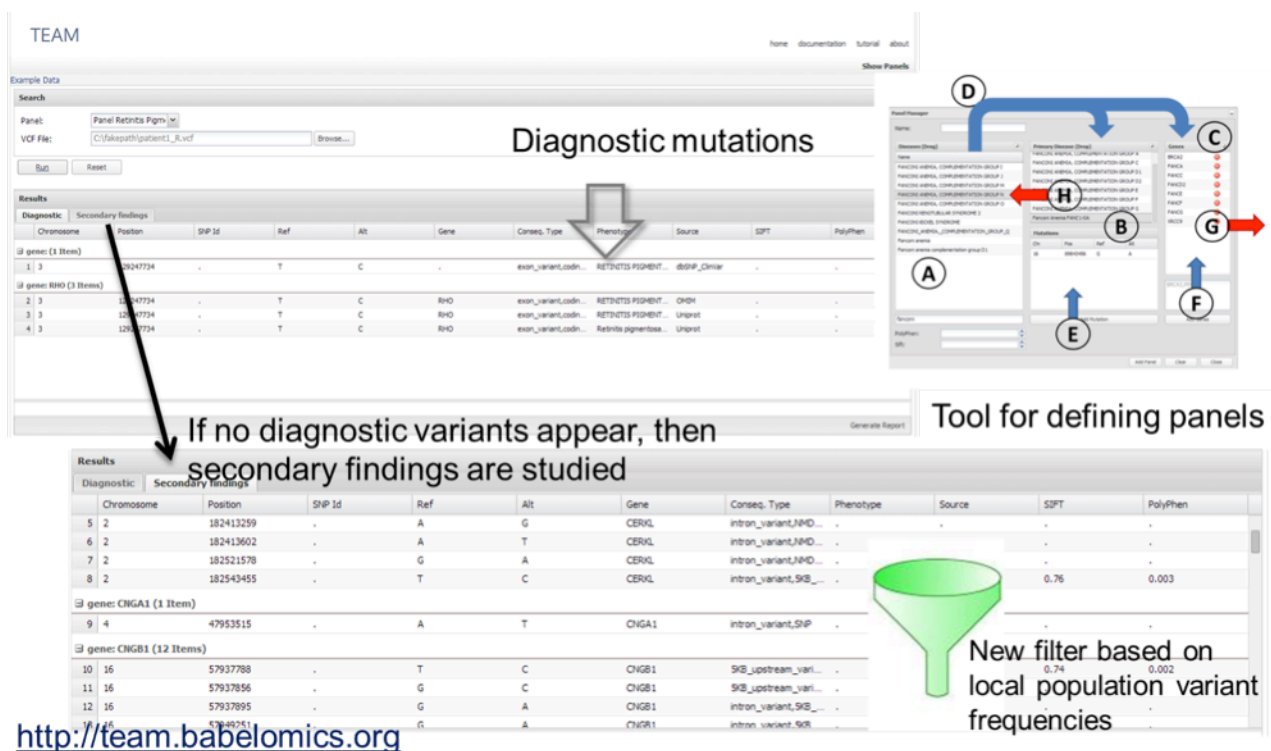


Figure 1: Interface of the application. Panels for defining targeted regions of interest can be set up by just drag and drop known disease genes or disease definitions from the lists. Thus, virtual panels can be interactively improved as the knowledge of the disease increases.

Figure 5. Interface of the application. Panels for defining targeted regions of interest can be set up by just drag and drop known disease genes or disease definitions from the lists. Thus, virtual panels can be interactively improved as the knowledge of the disease increases.

Priorization component (BiERApp)

Access

BiERApp is available at the following address: <http://omics.it4i.cz/bierapp/>

!!! Note “Note” The address is accessible only via VPN.

BiERApp

This tool is aimed to discover new disease genes or variants by studying affected families or cases and controls. It carries out a filtering process to sequentially remove: (i) variants which are not compatible with the disease because they are not expected to have impact on the protein function; (ii) variants that exist at frequencies incompatible with the disease; (iii) variants that do not segregate with the disease. The result is a reduced set of disease gene candidates that should be further validated experimentally.

BiERApp (28) efficiently helps in the identification of causative variants in family and sporadic genetic diseases. The program reads lists of predicted variants (nucleotide substitutions and indels) in affected individuals or tumor samples and controls. In family studies, different modes of inheritance can easily be defined to filter out variants that do not segregate with the disease along the family. Moreover, BiERApp integrates additional information such as allelic frequencies in the general population and the most popular damaging scores to further narrow down the number of putative variants in successive filtering steps. BiERApp provides an interactive and user-friendly interface that implements the filtering strategy used in the context of a large-scale genomic project carried out by the Spanish Network for Research in Rare Diseases (CIBERER) and the Medical Genome Project, in which more than 800 exomes have been analyzed.

Figure 6. Web interface to the prioritization tool. This figure shows the interface of the web tool for candidate gene prioritization with the filters available. The tool includes a genomic viewer (Genome Maps 30) that enables the representation of the variants in the corresponding genomic coordinates.

The image displays the BiERapp web interface, which is used for candidate gene prioritization. The interface includes several key components:

- Filters (Left Panel):** A sidebar with multiple filter categories:
 - Segregation:** Reload, Clear, Search.
 - MAF:** Reload, Clear, Search.
 - Effect:** Reload, Clear, Search.
 - Region:** Reload, Clear, Search.
 - Gene:** Reload, Clear, Search.
 - Annotations:** Select one or multiple consec. type (e.g., 5' UTR, Stop codon, Stop lost, Stop gained, KIXA polymerase promoter, Stop gained, DNaseI hypersensitive site, Exon variant, 3 prime UTR variant, Intron variant, SNP, Stop lost, Synonymous codon, NMD transcript variant, CpG island, MIRNA target site, Splice region variant, Non synonymous codon).
 - MAF:** Reload, Clear, Search.
 - Effect:** Reload, Clear, Search.
 - Region:** Reload, Clear, Search.
 - Gene:** Reload, Clear, Search.
- Pedigree Chart (Top Center):** A simple pedigree showing four individuals: NA19660 (male), NA19661 (female), NA19600 (male), and NA19685 (female). NA19660 and NA19661 are parents of NA19600 and NA19685.
- Filters (Right Panel):** A sidebar with filter categories:
 - Segregation:** Reload, Clear, Search.
 - MAF:** Reload, Clear, Search.
 - Effect:** Reload, Clear, Search.
 - Region:** Reload, Clear, Search.
 - Gene:** Reload, Clear, Search.
 - Annotations:** Enter regions (comma separated) (e.g., 1:1-1000000000), Enter genes (comma separated) (e.g., BRCA2,PPL).
- Variant Table (Bottom):** A table showing variant information, including Variant ID, Allele, Gene, Samples, and various metrics. The table is filtered to show variants associated with the selected filters.

BiERapp: the interactive filtering tool for easy candidate prioritization
<http://bierapp.babelomics.org>

Figure 1: Web interface to the prioritization tool. This figure shows the interface of the web tool for candidate gene prioritization with the filters available. The tool includes a genomic viewer (Genome Maps 30) that enables the representation of the variants in the corresponding genomic coordinates.

Overview

The human NGS data processing solution

Introduction

The scope of this OMICS MASTER solution is restricted to human genomics research (disease causing gene discovery in whole human genome or exome) or diagnosis (panel sequencing), although it could be extended in the future to other usages.

The pipeline inputs the raw data produced by the sequencing machines and undergoes a processing procedure that consists on a quality control, the mapping and variant calling steps that result in a file containing the set of variants in the sample. From this point, the prioritization component or the diagnostic component can be launched.

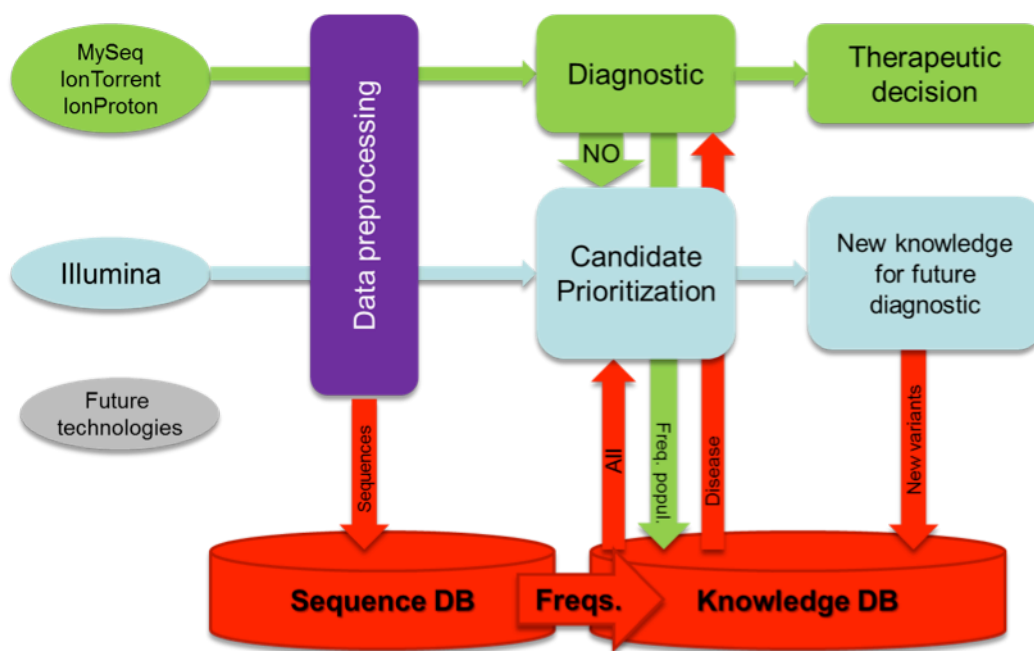


Figure 1: OMICS MASTER solution overview. Data is produced in the external labs and comes to IT4I (represented by the blue dashed line). The data pre-processor converts raw data into a list of variants and annotations for each sequenced patient. These lists files together with primary and secondary (alignment) data files are stored in IT4I sequence DB and uploaded to the discovery (candidate prioritization) or diagnostic component where they can be analysed directly by the user that produced them, depending of the experimental design carried out.

Figure 1. OMICS MASTER solution overview. Data is produced in the external labs and comes to IT4I (represented by the blue dashed line). The data pre-processor converts raw data into a list of variants and annotations for each sequenced patient. These lists files together with primary and secondary (alignment) data files are stored in IT4I sequence DB and uploaded to the discovery (candidate prioritization) or diagnostic component where they can be analyzed directly by the user that produced them, depending of the experimental design carried out.

Typical genomics pipelines are composed by several components that need to be launched manually. The advantage of OMICS MASTER pipeline is that all these components are invoked sequentially in an automated way.

OMICS MASTER pipeline inputs a FASTQ file and outputs an enriched VCF file. This pipeline is able to queue all the jobs to PBS by only launching a process taking all the necessary input files and creates the intermediate and final folders

Let's see each of the OMICS MASTER solution components:

Components

Processing

This component is composed by a set of programs that carry out quality controls, alignment, re-alignment, variant calling and variant annotation. It turns raw data from the sequencing machine into files containing lists of variants (VCF) that once annotated, can be used by the following components (discovery and diagnosis).

We distinguish three types of sequencing instruments: bench sequencers (MySeq, IonTorrent, and Roche Junior, although this last one is about being discontinued), which produce relatively Genomes in the clinic

low throughput (tens of million reads), and high end sequencers, which produce high throughput (hundreds of million reads) among which we have Illumina HiSeq 2000 (and new models) and SOLiD. All of them but SOLiD produce data in sequence format. SOLiD produces data in a special format called colour space that require of specific software for the mapping process. Once the mapping has been done, the rest of the pipeline is identical. Anyway, SOLiD is a technology which is also about being discontinued by the manufacturer so, this type of data will be scarce in the future.

Quality control, preprocessing and statistics for FASTQ

FastQC& FastQC.

These steps are carried out over the original FASTQ file with optimized scripts and includes the following steps: sequence cleansing, estimation of base quality scores, elimination of duplicates and statistics.

Input: **FASTQ file.**

Output: **FASTQ file plus an HTML file containing statistics on the data.**

FASTQ format It represents the nucleotide sequence and its corresponding quality scores.

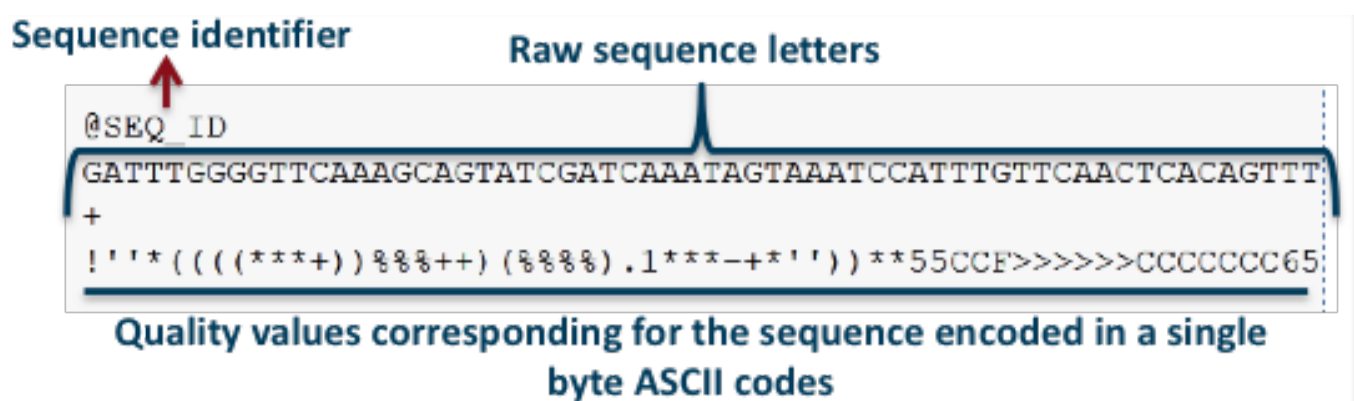


Figure 2.FASTQ file.

Mapping

Component:** Hpg-aligner.**

Sequence reads are mapped over the human reference genome. SOLiD reads are not covered by this solution; they should be mapped with specific software (among the few available options, SHRiMP seems to be the best one). For the rest of NGS machine outputs we use HPG Aligner. HPG-Aligner

is an innovative solution, based on a combination of mapping with BWT and local alignment with Smith-Waterman (SW), that drastically increases mapping accuracy (97% versus 62-70% by current mappers, in the most common scenarios). This proposal provides a simple and fast solution that maps almost all the reads, even those containing a high number of mismatches or indels.

Input: **FASTQ file**.

Output:** Aligned file in BAM format.**

Sequence Alignment/Map (SAM)

It is a human readable tab-delimited format in which each read and its alignment is represented on a single line. The format can represent unmapped reads, reads that are mapped to unique locations, and reads that are mapped to multiple locations.

The SAM format (1) consists of one header section and one alignment section. The lines in the header section start with character '@', and lines in the alignment section do not. All lines are TAB delimited.

In SAM, each alignment line has 11 mandatory fields and a variable number of optional fields. The mandatory fields are briefly described in Table 1. They must be present but their value can be a '*' or a zero (depending on the field) if the corresponding information is unavailable.

No.	Name	Description									
				1	QNAME	Query NAME of the read or the read pair		2	FLAG	Bitwise FLAG (pairing, strand, mate strand, etc.)	
				3	RNAME	Reference sequence NAME		4	POS	1-Based leftmost POSition of clipped alignment	
				5	MAPQ	MAPping Quality (Phred-scaled)		6	CIGAR	Extended CIGAR string (operations: MIDNSHP)	
				7	MRNM	Mate REference NaMe ('=' if same RNAME)		8	MPOS	1-Based leftmost Mate POSition	
				9	ISIZE	Inferred Insert SIZE		10	SEQ	Query SEQUENCE on the same strand as the reference	
				11	QUAL	Query QUALity (ASCII-33=Phred base quality)					

Table 1. Mandatory fields in the SAM format.

The standard CIGAR description of pairwise alignment defines three operations: 'M' for match/mismatch, 'I' for insertion compared with the reference and 'D' for deletion. The extended CIGAR proposed in SAM added four more operations: 'N' for skipped bases on the reference, 'S' for soft clipping, 'H' for hard clipping and 'P' for padding. These support splicing, clipping, multi-part and padded alignments. Figure 3 shows examples of CIGAR strings for different types of alignments.

Figure 3. SAM format file. The '@SQ' line in the header section gives the order of reference sequences. Notably, r001 is the name of a read pair. According to FLAG 163 (=1+2+32+128), the read mapped to position 7 is the second read in the pair (128) and regarded as properly paired (1 + 2); its mate is mapped to 37 on the reverse strand (32). Read r002 has three soft-clipped (unaligned) bases. The coordinate shown in SAM is the position of the first aligned base. The CIGAR string for this alignment contains a P (padding) operation which correctly aligns the inserted sequences. Padding operations can be absent when an aligner does not support multiple sequence alignment. The last six bases of read r003 map to position 9, and the first five to position 29 on the reverse strand. The hard clipping operation H indicates that the clipped sequence is not present in the

```
@SQ SN:ref LN:45
r001 163 ref 7 30 8M2I4M1D3M = 37 39 TTAGATAAAGGATACTA *
r002 0 ref 9 30 3S6M1P1I4M * 0 0 AAAAGATAAGGATA *
r003 0 ref 9 30 5H6M * 0 0 AGCTAA * NM:i:1
r004 0 ref 16 30 6M14N5M * 0 0 ATAGCTTCAGC *
r003 16 ref 29 30 6H5M * 0 0 TAGGC * NM:i:0
r001 83 ref 37 30 9M = 7 -39 CAGCGCCAT *
```

Figure 2: SAM format file. The '@SQ' line in the header section gives the order of reference sequences. Notably, r001 is the name of a read pair. According to FLAG 163 (=1+2+32+128), the read mapped to position 7 is the second read in the pair (128) and regarded as properly paired (1 + 2); its mate is mapped to 37 on the reverse strand (32). Read r002 has three soft-clipped (unaligned) bases. The coordinate shown in SAM is the position of the first aligned base. The CIGAR string for this alignment contains a P (padding) operation which correctly aligns the inserted sequences. Padding operations can be absent when an aligner does not support multiple sequence alignment. The last six bases of read r003 map to position 9, and the first five to position 29 on the reverse strand. The hard clipping operation H indicates that the clipped sequence is not present in the sequence field. The NM tag gives the number of mismatches. Read r004 is aligned across an intron, indicated by the N operation.

sequence field. The NM tag gives the number of mismatches. Read r004 is aligned across an intron, indicated by the N operation.

Binary Alignment/Map (BAM)

BAM is the binary representation of SAM and keeps exactly the same information as SAM. BAM uses lossless compression to reduce the size of the data by about 75% and provides an indexing system that allows reads that overlap a region of the genome to be retrieved and rapidly traversed.

Quality control, preprocessing and statistics for BAM

Component: Hpg-Fastq & FastQC. Some features:

- Quality control: % reads with N errors, % reads with multiple mappings, strand bias, paired-end insert, ...
- Filtering: by number of errors, number of hits, ...
 - Comparator: stats, intersection, ...

Input: BAM file.

Output: BAM file plus an HTML file containing statistics.

Variant Calling

Component: GATK.

Identification of single nucleotide variants and indels on the alignments is performed using the Genome Analysis Toolkit (GATK). GATK (2) is a software package developed at the Broad Institute to analyze high-throughput sequencing data. The toolkit offers a wide variety of tools, with a primary focus on variant discovery and genotyping as well as strong emphasis on data quality assurance.

Input: BAM

Output: VCF

Variant Call Format (VCF)

VCF (3) is a standardized format for storing the most prevalent types of sequence variation, including SNPs, indels and larger structural variants, together with rich annotations. The format was developed with the primary intention to represent human genetic variation, but its use is not restricted to diploid genomes and can be used in different contexts as well. Its flexibility and user extensibility allows representation of a wide variety of genomic variation with respect to a single reference sequence.

A VCF file consists of a header section and a data section. The header contains an arbitrary number of meta-information lines, each starting with characters ‘##’, and a TAB delimited field definition line, starting with a single ‘#’ character. The meta-information header lines provide a standardized description of tags and annotations used in the data section. The use of meta-information allows the information stored within a VCF file to be tailored to the dataset in question. It can be also used to provide information about the means of file creation, date of creation, version of the reference sequence, software used and any other information relevant to the history of the file. The field definition line names eight mandatory columns, corresponding to data columns representing the chromosome (CHROM), a 1-based position of the start of the variant (POS), unique identifiers of the variant (ID), the reference allele (REF), a comma separated list of alternate non-reference alleles (ALT), a phred-scaled quality score (QUAL), site filtering information (FILTER) and a semicolon separated list of additional, user extensible annotation (INFO). In addition, if samples are present in the file, the mandatory header columns are followed by a FORMAT column and an arbitrary number of sample IDs that define the samples included in the VCF file. The FORMAT column is used to define the information contained within each subsequent genotype column, which consists of a colon separated list of fields. For example, the FORMAT field GT:GQ:DP in the fourth data entry of Figure 1a indicates that the subsequent entries contain information regarding the genotype, genotype quality and read depth for each sample. All data lines are TAB delimited and the number of fields in each data line must match the number of fields in the header line. It is strongly recommended that all annotation tags used are declared in the VCF header section.

Figure 4. (a) Example of valid VCF. The header lines ##fileformat and #CHROM are mandatory, the rest is optional but strongly recommended. Each line of the body describes variants present in the sampled population at one genomic position or region. All alternate alleles are listed in the ALT column and referenced from the genotype fields as 1-based indexes to this list; the reference haplotype is designated as 0. For multiploid data, the separator indicates whether the data are phased (|) or unphased (/). Thus, the two alleles C and G at the positions 2 and 5 in this figure occur on the same chromosome in SAMPLE1. The first data line shows an example of a deletion (present in SAMPLE1) and a replacement of two bases by another base (SAMPLE2); the second line shows a SNP and an insertion; the third a SNP; the fourth a large structural variant described by the annotation in the INFO column, the coordinate is that of the base before the variant. (b–f) Alignments and VCF representations of different sequence variants: SNP, insertion, deletion, replacement, and a large deletion. The REF column shows the reference bases replaced by the haplotype in the ALT column. The coordinate refers to the first reference base. (g) Users are advised to use simplest representation possible and lowest coordinate in cases where the position is ambiguous.

Annotating

Component: HPG-Variant

The functional consequences of every variant found are then annotated using the HPG-Variant software, which extracts from CellBase, the Knowledge database, all the information relevant on the predicted pathologic effect of the variants.

(a) **VCF example**

```
##fileformat=VCFv4.1
##fileDate=20110413
##source=VCFtools
##reference=file:///refs/human_NCBIS36.fasta
##contig=<ID=1,length=249250621,mD5=1b22b98cdeb4a9304cb5d48026a85128,species="Homo Sapiens">
##contig=<ID=X,length=155270560,mD5=7e0e2e580297b7764e31dbc80c2540dd,species="Homo Sapiens">
##INFO=<ID=AA,Number=1,Type=String,Description="Ancestral Allele">
##INFO=<ID=H2,Number=0,Type=Flag,Description="HapMap2 membership">
##FORMAT=<ID=GT,Number=1,Type=String,Description="Genotype">
##FORMAT=<ID=GQ,Number=1,Type=Integer,Description="Genotype Quality">
##FORMAT=<ID=DP,Number=1,Type=Integer,Description="Read Depth">
##ALT=<ID=DEL,Description="Deletion">
##INFO=<ID=SVTYPE,Number=1,Type=String,Description="Type of structural variant">
##INFO=<ID=END,Number=1,Type=Integer,Description="End position of the variant">
```

#CHROM	POS	ID	REF	ALT	QUAL	FILTER	INFO	FORMAT	SAMPLE1	SAMPLE2
1	1	.	ACG	A,AT	40	PASS	.	GT:DP	1/1:13	2/2:29
1	2	C	T,CT	.	PASS	H2;AA=T	GT	GT	0/1	2/2
1	5	rs12	A	G	67	PASS	.	GT:DP	1/0:16	2/2:20
X	100	.	T		.	PASS	SVTYPE=DEL;END=299	GT:GQ:DP	1:12:.	0/0:20:36

(b) **SNP**

Alignment	VCF representation			
	POS	REF	ALT	
1234				
ACGT	2	C	T	
ATGT				

(c) **Insertion**

Alignment	POS	REF	ALT
12345			
AC-GT	2	C	CT
ACTGT			

(d) **Deletion**

Alignment	POS	REF	ALT
1234			
ACGT	1	ACG	A
A--T			

(e) **Replacement**

Alignment	POS	REF	ALT
1234			
ACGT			
A-TT			

(f) **Large structural variant**

Alignment	VCF representation				
	POS	REF	ALT	INFO	
ACGTACGTACGTACGTACGTACGTACGT[...]	100	110	120	290	300
ACGT-----[...]-GTAC	100	T		SVTYPE=DEL;END=299	

(g) **Resolving ambiguity**

Alignment	Possible representation		Possible representation		Recommended VCF representation	
	POS	REF	ALT	POS	REF	ALT
1234567890						
TTTCCTCTTA	1	TTTCCTCTT	CTTACCTA	1	T	C
CTTACCT--A				4	C	A
^ ^ ^ ^				7	TCT	T
				5	CCT	C

Figure 3: a) Example of valid VCF. The header lines `##fileformat` and `#CHROM` are mandatory, the rest is optional but strongly recommended. Each line of the body describes variants present in the sampled population at one genomic position or region. All alternate alleles are listed in the ALT column and referenced from the genotype fields as 1-based indexes to this list; the reference haplotype is designated as 0. For multiploid data, the separator indicates whether the data are phased (|) or unphased (/). Thus, the two alleles C and G at the positions 2 and 5 in this figure occur on the same chromosome in SAMPLE1. The first data line shows an example of a deletion (present in SAMPLE1) and a replacement of two bases by another base (SAMPLE2); the second line shows a SNP and an insertion; the third a SNP; the fourth a large structural variant described by the annotation in the INFO column, the coordinate is that of the base before the variant. (b–f) Alignments and VCF representations of different sequence variants: SNP, insertion, deletion, replacement, and a large deletion. The REF columns shows the reference bases replaced by the haplotype in the ALT column. The coordinate refers to the first reference base. (g) Users are advised to use simplest representation possible and lowest coordinate in cases where the position is ambiguous.

VARIANT (VARiAnt Analysis Tool) (4) reports information on the variants found that include consequence type and annotations taken from different databases and repositories (SNPs and variants from dbSNP and 1000 genomes, and disease-related variants from the Genome-Wide Association Study (GWAS) catalog, Online Mendelian Inheritance in Man (OMIM), Catalog of Somatic Mutations in Cancer (COSMIC) mutations, etc. VARIANT also produces a rich variety of annotations that include information on the regulatory (transcription factor or miRNA binding sites, etc.) or structural roles, or on the selective pressures on the sites affected by the variation. This information allows extending the conventional reports beyond the coding regions and expands the knowledge on the contribution of non-coding or synonymous variants to the phenotype studied.

Input: VCF

Output: The output of this step is the Variant Calling Format (VCF) file, which contains changes with respect to the reference genome with the corresponding QC and functional annotations.

CellBase

CellBase(5) is a relational database integrates biological information from different sources and includes:

Core features:

We took genome sequences, genes, transcripts, exons, cytobands or cross references (xrefs) identifiers (IDs) from Ensembl (6). Protein information including sequences, xrefs or protein features (natural variants, mutagenesis sites, post-translational modifications, etc.) were imported from UniProt (7).

Regulatory:

CellBase imports miRNA from miRBase (8); curated and non-curated miRNA targets from miRecords (9), miRTarBase (10), TargetScan(11) and microRNA.org (12) and CpG islands and conserved regions from the UCSC database (13).

Functional annotation

OBO Foundry (14) develops many biomedical ontologies that are implemented in OBO format. We designed a SQL schema to store these OBO ontologies and >30 ontologies were imported. OBO ontology term annotations were taken from Ensembl (6). InterPro (15) annotations were also imported.

Variation

CellBase includes SNPs from dbSNP (16)[^]; SNP population frequencies from HapMap (17), 1000 genomes project (18) and Ensembl (6); phenotypically annotated SNPs were imported from NHRI GWAS Catalog (19), HGMD (20), Open Access GWAS Database (21), UniProt (7) and OMIM (22); mutations from COSMIC (23) and structural variations from Ensembl (6).

Systems biology

We also import systems biology information like interactome information from IntAct (24). Reactome (25) stores pathway and interaction information in BioPAX (26) format. BioPAX data exchange format enables the integration of diverse pathway resources. We successfully solved the problem of storing data released in BioPAX format into a SQL relational schema, which allowed us importing Reactome in CellBase.

Diagnostic component (TEAM)

Priorization component (BiERApp)

Usage

First of all, we should load ngsPipeline module:

```
$ module load ngsPipeline
```

This command will load python/2.7.5 module and all the required modules (hpg-aligner, gatk, etc)

If we launch ngsPipeline with '-h', we will get the usage help:

```
$ ngsPipeline -h
Usage: ngsPipeline.py [-h] -i INPUT -o OUTPUT -p PED --project PROJECT --queue
                        QUEUE [--stages-path STAGES_PATH] [--email EMAIL]
                        [--prefix PREFIX] [-s START] [-e END] --log
```

Python pipeline

optional arguments:

```
-h, --help            show this help message and exit
-i INPUT, --input INPUT
-o OUTPUT, --output OUTPUT
                        Output Data directory
-p PED, --ped PED      Ped file with all individuals
--project PROJECT      Project Id
--queue QUEUE          Queue Id
--stages-path STAGES_PATH
                        Custom Stages path
--email EMAIL          Email
--prefix PREFIX        Prefix name for Queue Jobs name
-s START, --start START
                        Initial stage
-e END, --end END      Final stage
--log                 Log to file
```

Let us see a brief description of the arguments:

-h --help. Show the help.

-i, --input. The input data directory. This directory must to have a special structure, **second** one.

-o , --output. The output folder. This folder will contain all the intermediate and final results.

-p , --ped. The ped file with the pedigree. This file contains all the sample names and their relationships.

--email. Email for PBS notifications.

--prefix. Prefix for PBS Job names.

-s, --start & -e, --end. Initial and final stage. If we want to launch the pipeline, we should use **-s** and **-e** options.

---log*. Using log argument NGSPipeline will prompt all the logs to this file.

---project*>. Project ID of your supercomputer allocation.

---queue*. [Queue](../../resource-allocation-and-job-execution/introduction.html) t

Input, output and ped arguments are mandatory. If the output folder does not exist, the pipeline will create it.

Examples

This is an example usage of NGSPipeline:

We have a folder with the following structure in

`/apps/bio/omics/1.0/sample_data/ >:`

```
/apps/bio/omics/1.0/sample_data
data
  file.ped
  sample1
    sample1_1.fq
    sample1_2.fq
  sample2
    sample2_1.fq
    sample2_2.fq
```

The ped file (file.ped) contains the following info:

```
#family_ID sample_ID parental_ID maternal_ID sex phenotype
FAM sample_A 0 0 1 1
FAM sample_B 0 0 2 2
```

Now, lets load the NGSPipeline module and copy the sample data to a scratch directory:

```
$ module load ngsPipeline
$ mkdir -p /scratch/$USER/omics/results
$ cp -r /apps/bio/omics/1.0/sample_data /scratch/$USER/omics/
```

Now, we can launch the pipeline (replace OPEN-0-0 with your Project ID):

```
$ ngsPipeline -i /scratch/$USER/omics/sample_data/data -o /scratch/$USER/omics/result
```

This command submits the processing jobs to the queue.

If we want to re-launch the pipeline from stage 4 until stage 20 we should use the next command:

```
$ ngsPipeline -i /scratch/$USER/omics/sample_data/data -o /scratch/$USER/omics/result
```

Details on the pipeline

The pipeline calls the following tools: - fastqc¹, quality control tool for high throughput sequence data. - gatk², The Genome Analysis Toolkit or GATK is a software package developed at the Broad Institute to analyze high-throughput sequencing data. The toolkit offers a wide variety of tools, with a primary focus on variant discovery and genotyping as well as strong emphasis on data quality assurance. Its robust architecture, powerful processing engine and high-performance computing

features make it capable of taking on projects of any size. - [hpg-aligner](#), HPG Aligner has been designed to align short and long reads with high sensitivity, therefore any number of mismatches or indels are allowed. HPG Aligner implements and combines two well known algorithms: *Burrows-Wheeler Transform* (BWT) to speed-up mapping high-quality reads, and *Smith-Waterman* (SW) to increase sensitivity when reads cannot be mapped using BWT. - [hpg-fastq](#), a quality control tool for high throughput sequence data. - [hpg-variant](#), The HPG Variant suite is an ambitious project aimed to provide a complete suite of tools to work with genomic variation data, from VCF tools to variant profiling or genomic statistics. It is being implemented using High Performance Computing technologies to provide the best performance possible. - [picard](#), Picard comprises Java-based command-line utilities that manipulate SAM files, and a Java API (HTSJDK) for creating new programs that read and write SAM files. Both SAM text format and SAM binary (BAM) format are supported. - [samtools](#), SAM Tools provide various utilities for manipulating alignments in the SAM format, including sorting, merging, indexing and generating alignments in a per-position format. - [snpEff](#), Genetic variant annotation and effect prediction toolbox.

This listing show which tools are used in each step of the pipeline :

- stage-00: fastqc
- stage-01: hpg_fastq
- stage-02: fastqc
- stage-03: hpg_aligner and samtools
- stage-04: samtools
- stage-05: samtools
- stage-06: fastqc
- stage-07: picard
- stage-08: fastqc
- stage-09: picard
- stage-10: gatk
- stage-11: gatk
- stage-12: gatk
- stage-13: gatk
- stage-14: gatk
- stage-15: gatk
- stage-16: samtools
- stage-17: samtools
- stage-18: fastqc
- stage-19: gatk
- stage-20: gatk
- stage-21: gatk
- stage-22: gatk
- stage-23: gatk
- stage-24: hpg-variant
- stage-25: hpg-variant
- stage-26: snpEff
- stage-27: snpEff
- stage-28: hpg-variant

Interpretation

The output folder contains all the subfolders with the intermediate data. This folder contains the final VCF with all the variants. This file can be uploaded into TEAM by using the VCF file button. It is important to note here that the entire management of the VCF file is local: no patient's

sequence data is sent over the Internet thus avoiding any problem of data privacy or confidentiality.

Figure 7. *TEAM upload panel. Once the file has been uploaded, a panel must be chosen from the Panel list. Then, pressing the Run button the diagnostic process starts.*

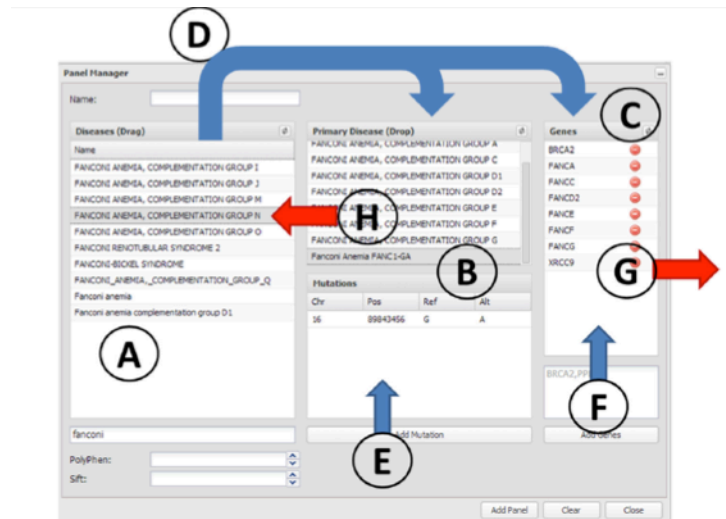









Figure 7. The panel manager. The elements used to define a panel are (**A**) disease terms, (**B**) diagnostic mutations and (**C**) genes. Arrows represent actions that can be taken in the panel manager. Panels can be defined by using the known mutations and genes of a particular disease. This can be done by dragging them to the **Primary Diagnostic** box (action **D**). This action, in addition to defining the diseases in the **Primary Diagnostic** box, automatically adds the corresponding genes to the **Genes** box. The panels can be customized by adding new genes (action **F**) or removing undesired genes (action **G**). New disease mutations can be added independently or associated to an already existing disease term (action **E**). Disease terms can be removed by simply dragging them back (action **H**).






3. Petr Danecek, Adam Auton, Goncalo Abecasis, Cornelis A. Albers, Eric Banks, Mark A. DePristo, Robert E. Handsaker, Gerton Lunter, Gabor T. Marth, Stephen T. Sherry, Gilean McVean, Richard Durbin, and 1000 Genomes Project Analysis Group. The variant call format and VCFtools. *Bioinformatics* 2011, 27: 2156-2158.
4. Medina I, De Maria A, Bleda M, Salavert F, Alonso R, Gonzalez CY, Dopazo J: VARIANT: Command Line, Web service and Web interface for fast and accurate functional characterization of variants found by Next-Generation Sequencing. *Nucleic Acids Res* 2012, 40:W54-58.
5. Bleda M, Tarraga J, de Maria A, Salavert F, Garcia-Alonso L, Celma M, Martin A, Dopazo J, Medina I: CellBase, a comprehensive collection of RESTful web services for retrieving relevant biological information from heterogeneous sources. *Nucleic Acids Res* 2012, 40:W609-614.
6. Flicek,P., Amode,M.R., Barrell,D., Beal,K., Brent,S., Carvalho-Silva,D., Clapham,P., Coates,G., Fairley,S., Fitzgerald,S. et al. (2012) Ensembl 2012. *Nucleic Acids Res.*, 40, D84–D90.
7. UniProt Consortium. (2012) Reorganizing the protein space at the Universal Protein Resource (UniProt). *Nucleic Acids Res.*, 40, D71–D75.
8. Kozomara,A. and Griffiths-Jones,S. (2011) miRBase: integrating microRNA annotation and deep-sequencing data. *Nucleic Acids Res.*, 39, D152–D157.
9. Xiao,F., Zuo,Z., Cai,G., Kang,S., Gao,X. and Li,T. (2009) miRecords: an integrated resource for microRNA-target interactions. *Nucleic Acids Res.*, 37, D105–D110.
10. Hsu,S.D., Lin,F.M., Wu,W.Y., Liang,C., Huang,W.C., Chan,W.L., Tsai,W.T., Chen,G.Z., Lee,C.J., Chiu,C.M. et al. (2011) miRTarBase: a database curates experimentally validated microRNA-target interactions. *Nucleic Acids Res.*, 39, D163–D169.
11. Friedman,R.C., Farh,K.K., Burge,C.B. and Bartel,D.P. (2009) Most mammalian mRNAs are conserved targets of microRNAs. *Genome Res.*, 19, 92–105.
12. Betel,D., Wilson,M., Gabow,A., Marks,D.S. and Sander,C. (2008) The microRNA.org resource: targets and expression. *Nucleic Acids Res.*, 36, D149–D153.
12. Dreszer,T.R., Karolchik,D., Zweig,A.S., Hinrichs,A.S., Raney,B.J., Kuhn,R.M., Meyer,L.R., Wong,M., Sloan,C.A., Rosenbloom,K.R. et al. (2012) The UCSC genome browser database: extensions and updates 2011. *Nucleic Acids Res.*,40, D918–D923.
13. Smith,B., Ashburner,M., Rosse,C., Bard,J., Bug,W., Ceusters,W., Goldberg,L.J., Eilbeck,K., Ireland,A., Mungall,C.J. et al. (2007) The OBO Foundry: coordinated evolution of ontologies to support biomedical data integration. *Nat. Biotechnol.*, 25, 1251–1255.
14. Hunter,S., Jones,P., Mitchell,A., Apweiler,R., Attwood,T.K.,Bateman,A., Bernard,T., Binns,D., Bork,P., Burge,S. et al. (2012) InterPro in 2011: new developments in the family and domain prediction database. *Nucleic Acids Res.*,40, D306–D312.
15. Sherry,S.T., Ward,M.H., Kholodov,M., Baker,J., Phan,L., Smigielski,E.M. and Sirotkin,K. (2001) dbSNP: the NCBI database of genetic variation. *Nucleic Acids Res.*, 29, 308–311.
16. Altshuler,D.M., Gibbs,R.A., Peltonen,L., Dermitzakis,E., Schaffner,S.F., Yu,F., Bonnen,P.E., de Bakker,P.I., Deloukas,P., Gabriel,S.B. et al. (2010) Integrating common and rare genetic variation in diverse human populations. *Nature*, 467, 52–58.
17. 1000 Genomes Project Consortium. (2010) A map of human genome variation from population-scale sequencing. *Nature*, 467, 1061–1073.
18. Hindorff,L.A., Sethupathy,P., Junkins,H.A., Ramos,E.M., Mehta,J.P., Collins,F.S. and Manolio,T.A. (2009) Potential etiologic and functional implications of genome-wide association loci for human diseases and traits. *Proc. Natl Acad. Sci. USA*, 106, 9362–9367.
19. Stenson,P.D., Ball,E.V., Mort,M., Phillips,A.D., Shiel,J.A., Thomas,N.S., Abeysinghe,S., Krawczak,M. and Cooper,D.N. (2003) Human gene mutation database (HGMD): 2003 update. *Hum. Mutat.*, 21, 577–581.
20. Johnson,A.D. and O'Donnell,C.J. (2009) An open access database of genome-wide association results. *BMC Med. Genet*, 10, 6.
21. McKusick,V. (1998) A Catalog of Human Genes and Genetic Disorders, 12th edn. John Hopkins University Press,Baltimore, MD.

22. Forbes,S.A., Bindal,N., Bamford,S., Cole,C., Kok,C.Y., Beare,D., Jia,M., Shepherd,R., Leung,K., Menzies,A. et al. (2011) COSMIC: mining complete cancer genomes in the catalogue of somatic mutations in cancer. *Nucleic Acids Res.*, 39, D945–D950.
23. Kerrien,S., Aranda,B., Breuza,L., Bridge,A., Broackes-Carter,F., Chen,C., Duesbury,M., Dumousseau,M., Feuermann,M., Hinz,U. et al. (2012) The Intact molecular interaction database in 2012. *Nucleic Acids Res.*, 40, D841–D846.
24. Croft,D., O’Kelly,G., Wu,G., Haw,R., Gillespie,M., Matthews,L., Caudy,M., Garapati,P., Gopinath,G., Jassal,B. et al. (2011) Reactome: a database of reactions, pathways and biological processes. *Nucleic Acids Res.*, 39, D691–D697.
25. Demir,E., Cary,M.P., Paley,S., Fukuda,K., Lemer,C., Vastrik,I.,Wu,G., D’Eustachio,P., Schaefer,C., Luciano,J. et al. (2010) The BioPAX community standard for pathway data sharing. *Nature Biotechnol.*, 28, 935–942.
26. Alemán Z, García-García F, Medina I, Dopazo J (2014): A web tool for the design and management of panels of genes for targeted enrichment and massive sequencing for clinical applications. *Nucleic Acids Res* 42: W83-7.
27. Alemán A , Garcia-Garcia F , Salavert F , Medina I , Dopazo J  (2014). A web-based interactive framework to assist in the prioritization of disease candidate genes in whole-exome sequencing studies. *Nucleic Acids Res.* 42 :W88-93.
28. Landrum,M.J., Lee,J.M., Riley,G.R., Jang,W., Rubinstein,W.S., Church,D.M. and Maglott,D.R. (2014) ClinVar: public archive of relationships among sequence variation and human phenotype. *Nucleic Acids Res.*, 42, D980–D985.
29. Medina I, Salavert F, Sanchez R, de Maria A, Alonso R, Escobar P, Bleda M, Dopazo J: Genome Maps, a new generation genome browser. *Nucleic Acids Res* 2013, 41:W41-46.

COMSOL Multiphysics®

Introduction

COMSOL  is a powerful environment for modelling and solving various engineering and scientific problems based on partial differential equations. COMSOL is designed to solve coupled or multi-physics phenomena. For many standard engineering problems COMSOL provides add-on products such as electrical, mechanical, fluid flow, and chemical applications.

- Structural Mechanics Module ,
- Heat Transfer Module ,
- CFD Module ,
- Acoustics Module ,
- and many others 

COMSOL also allows an interface support for equation-based modelling of partial differential equations.

Execution

On the Anselm cluster COMSOL is available in the latest stable version. There are two variants of the release:

- **Non commercial** or so called **EDU variant**, which can be used for research and educational purposes.
- **Commercial** or so called **COM variant**, which can be used also for commercial activities. **COM variant** has only subset of features compared to the **EDU variant** available. More about licensing will be posted here soon.

To load the of COMSOL load the module

```
$ module load comsol
```

By default the **EDU variant** will be loaded. If user needs other version or variant, load the particular version. To obtain the list of available versions use

```
$ module avail comsol
```

If user needs to prepare COMSOL jobs in the interactive mode it is recommend to use COMSOL on the compute nodes via PBS Pro scheduler. In order run the COMSOL Desktop GUI on Windows is recommended to use the Virtual Network Computing (VNC).

```
$ xhost +  
$ qsub -I -X -A PROJECT_ID -q qprod -l select=1:ncpus=16  
$ module load comsol  
$ comsol
```

To run COMSOL in batch mode, without the COMSOL Desktop GUI environment, user can utilized the default (comsol.pbs) job script and execute it via the qsub command.

```
#!/bin/bash  
#PBS -l select=3:ncpus=16  
#PBS -q qprod  
#PBS -N JOB_NAME  
#PBS -A PROJECT_ID
```



```
cd /scratch/$USER/ || exit
```

```
echo Time is `date`  
echo Directory is `pwd`  
echo '**PBS_NODEFILE***START*****'  
cat $PBS_NODEFILE  
echo '**PBS_NODEFILE***END*****'
```

```
text_nodes < cat $PBS_NODEFILE
```

```
module load comsol  
# module load comsol/43b-COM
```

```
ntask=$(wc -l $PBS_NODEFILE)
```

```
comsol -nn ${ntask} batch -configuration /tmp -mpiarg -rmk -mpiarg pbs -tmpdir /scratch/$
```

Working directory has to be created before sending the (comsol.pbs) job script into the queue. Input file (name_input_f.mph) has to be in working directory or full path to input file has to be specified. The appropriate path to the temp directory of the job has to be set by command option (-tmpdir).

LiveLink™ for MATLAB®

COMSOL is the software package for the numerical solution of the partial differential equations. LiveLink for MATLAB allows connection to the COMSOL® API (Application Programming Interface) with the benefits of the programming language and computing environment of the MATLAB.

LiveLink for MATLAB is available in both **EDU** and **COM variant** of the COMSOL release. On Anselm 1 commercial (**COM**) license and the 5 educational (**EDU**) licenses of LiveLink for MATLAB (please see the ISV Licenses) are available. Following example shows how to start COMSOL model from MATLAB via LiveLink in the interactive mode.

```
$ xhost +  
$ qsub -I -X -A PROJECT_ID -q qexp -l select=1:ncpus=16  
$ module load matlab  
$ module load comsol  
$ comsol server matlab
```

At the first time to launch the LiveLink for MATLAB (client-MATLAB/server-COMSOL connection) the login and password is requested and this information is not requested again.

To run LiveLink for MATLAB in batch mode with (comsol_matlab.pbs) job script you can utilize/modify the following script and execute it via the qsub command.

```
#!/bin/bash  
#PBS -l select=3:ncpus=16  
#PBS -q qprod  
#PBS -N JOB_NAME  
#PBS -A PROJECT_ID
```

```
cd /scratch/$USER || exit
```

```
echo Time is `date`  
echo Directory is `pwd`  
echo '**PBS_NODEFILE***START*****'
```

```

cat $PBS_NODEFILE
echo '**PBS_NODEFILE***END*****'

text_nodes < cat $PBS_NODEFILE

module load matlab
module load comsol/43b-EDU

ntask=$(wc -l $PBS_NODEFILE)

comsol -nn ${ntask} server -configuration /tmp -mpiarg -rmk -mpiarg pbs -tmpdir /scratch/
cd /apps/engineering/comsol/comsol43b/mli
matlab -nodesktop -nosplash -r "mphstart; addpath /scratch/$USER; test_job"

```

This example shows how to run Livelink for MATLAB with following configuration: 3 nodes and 16 cores per node. Working directory has to be created before submitting (comsol_matlab.pbs) job script into the queue. Input file (test_job.m) has to be in working directory or full path to input file has to be specified. The Matlab command option (-r "mphstart") created a connection with a COMSOL server using the default port number.

Operating System

The operating system on Anselm is Linux - **bullx Linux Server release 6.X**

bullx Linux is based on Red Hat Enterprise Linux. bullx Linux is a Linux distribution provided by Bull and dedicated to HPC applications.

Virtualization

Running virtual machines on compute nodes

Introduction

There are situations when Anselm's environment is not suitable for user needs.

- Application requires different operating system (e.g Windows), application is not available for Linux
- Application requires different versions of base system libraries and tools
- Application requires specific setup (installation, configuration) of complex software stack
- Application requires privileged access to operating system
- ... and combinations of above cases

We offer solution for these cases - **virtualization**. Anselm's environment gives the possibility to run virtual machines on compute nodes. Users can create their own images of operating system with specific software stack and run instances of these images as virtual machines on compute nodes. Run of virtual machines is provided by standard mechanism of Resource Allocation and Job Execution.

Solution is based on QEMU-KVM software stack and provides hardware-assisted x86 virtualization.

Limitations

Anselm's infrastructure was not designed for virtualization. Anselm's environment is not intended primary for virtualization, compute nodes, storages and all infrastructure of Anselm is intended and optimized for running HPC jobs, this implies suboptimal configuration of virtualization and limitations.

Anselm's virtualization does not provide performance and all features of native environment. There is significant performance hit (degradation) in I/O performance (storage, network). Anselm's virtualization is not suitable for I/O (disk, network) intensive workloads.

Virtualization has also some drawbacks, it is not so easy to setup efficient solution.

Solution described in chapter HOWTO is suitable for single node tasks, does not introduce virtual machine clustering.

!!! Note "Note" Please consider virtualization as last resort solution for your needs.

Please consult use of virtualization with IT4Innovation's support.

For running Windows application (when source code and Linux native application are not available)

Licensing

IT4Innovations does not provide any licenses for operating systems and software of virtual machines. Users are (in accordance with Acceptable use policy document [\[4\]](#)) fully responsible for licensing all software running in virtual machines on Anselm. Be aware of complex conditions of licensing software in virtual environments.

!!! Note "Note" Users are responsible for licensing OS e.g. MS Windows and all software running in their virtual machines.

HOWTO

Virtual Machine Job Workflow

We propose this job workflow:

Our recommended solution is that job script creates distinct shared job directory, which makes a central point for data exchange between Anselm's environment, compute node (host) (e.g HOME, SCRATCH, local scratch and other local or cluster filesystems) and virtual machine (guest). Job script links or copies input data and instructions what to do (run script) for virtual machine to job directory and virtual machine process input data according instructions in job directory and store output back to job directory. We recommend, that virtual machine is running in so called snapshot mode, image is immutable - image does not change, so one image can be used for many concurrent jobs.

Procedure

1. Prepare image of your virtual machine
2. Optimize image of your virtual machine for Anselm's virtualization
3. Modify your image for running jobs
4. Create job script for executing virtual machine
5. Run jobs

Prepare image of your virtual machine

You can either use your existing image or create new image from scratch.

QEMU currently supports these image types or formats:

- raw
- cloop
- cow
- qcow
- qcow2
- vmdk - VMware 3 & 4, or 6 image format, for exchanging images with that product
- vdi - VirtualBox 1.1 compatible image format, for exchanging images with VirtualBox.

You can convert your existing image using `qemu-img convert` command. Supported formats of this command are: `blkdebug` `blkverify` `bochs` `cloop` `cow` `dmg` `file` `ftp` `ftps` `host_cdrom` `host_device` `host_floppy` `http` `https` `nbd` `parallels` `qcow` `qcow2` `qed` `raw` `sheepdog` `tftp` `vdi` `vhdx` `vmdk` `vpc` `vfat`.

We recommend using advanced QEMU native image format `qcow2`.

More about QEMU Images [🔗](#)

Optimize image of your virtual machine

Use virtio devices (for disk/drive and network adapter) and install virtio drivers (paravirtualized drivers) into virtual machine. There is significant performance gain when using virtio drivers. For more information see Virtio Linux [🔗](#) and Virtio Windows [🔗](#).

Disable all unnecessary services and tasks. Restrict all unnecessary operating system operations.

Remove all unnecessary software and files.

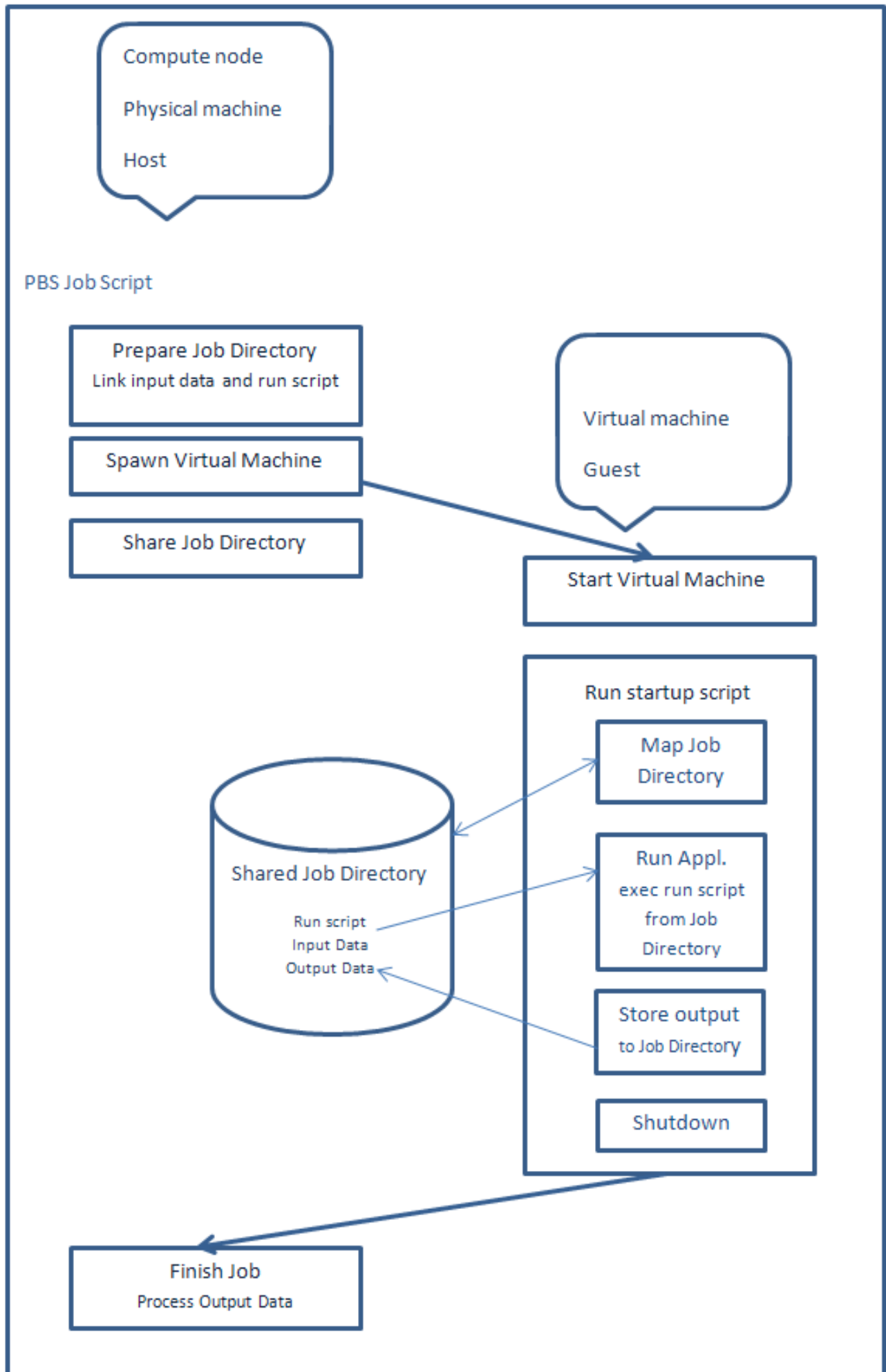


Figure 1: Workflow

Remove all paging space, swap files, partitions, etc.

Shrink your image. (It is recommended to zero all free space and reconvert image using qemu-img.)

Modify your image for running jobs

Your image should run some kind of operating system startup script. Startup script should run application and when application exits run shutdown or quit virtual machine.

We recommend, that startup script

- maps Job Directory from host (from compute node)
- runs script (we call it “run script”) from Job Directory and waits for application’s exit
- for management purposes if run script does not exist wait for some time period (few minutes)
- shutdowns/quits OS

For Windows operating systems we suggest using Local Group Policy Startup script, for Linux operating systems rc.local, runlevel init script or similar service.

Example startup script for Windows virtual machine:

```
@echo off
set LOG=c:startup.log
set MAPDRIVE=z:
set SCRIPT=%MAPDRIVE%run.bat
set TIMEOUT=300

echo %DATE% %TIME% Running startup script>%LOG%

rem Mount share
echo %DATE% %TIME% Mounting shared drive>%LOG%
net use z: 10.0.2.4qemu >%LOG% 2>&1
dir z: >%LOG% 2>&1
echo. >%LOG%

if exist %MAPDRIVE% (
    echo %DATE% %TIME% The drive "%MAPDRIVE%" exists>%LOG%

    if exist %SCRIPT% (
        echo %DATE% %TIME% The script file "%SCRIPT%"exists>%LOG%
        echo %DATE% %TIME% Running script %SCRIPT%>%LOG%
        set TIMEOUT=0
        call %SCRIPT%
    ) else (
        echo %DATE% %TIME% The script file "%SCRIPT%"does not exist>%LOG%
    )

) else (
    echo %DATE% %TIME% The drive "%MAPDRIVE%" does not exist>%LOG%
)
echo. >%LOG%

timeout /T %TIMEOUT%
```

```
echo %DATE% %TIME% Shut down>%LOG%
shutdown /s /t 0
```

Example startup script maps shared job script as drive z: and looks for run script called run.bat. If run script is found it is run else wait for 5 minutes, then shutdown virtual machine.

Create job script for executing virtual machine

Create job script according recommended

Virtual Machine Job Workflow .

Example job for Windows virtual machine:

```
#!/bin/sh

JOB_DIR=/scratch/$USER/win/${PBS_JOBID}

#Virtual machine settings
VM_IMAGE=~/.work/img/win.img
VM_MEMORY=49152
VM_SMP=16

# Prepare job dir
mkdir -p ${JOB_DIR} && cd ${JOB_DIR} || exit 1
ln -s ~/.work/win .
ln -s /scratch/$USER/data .
ln -s ~/.work/win/script/run/run-appl.bat run.bat

# Run virtual machine
export TMPDIR=/lscratch/${PBS_JOBID}
module add qemu
qemu-system-x86_64
    -enable-kvm
    -cpu host
    -smp ${VM_SMP}
    -m ${VM_MEMORY}
    -vga std
    -localtime
    -usb -usbdevice tablet
    -device virtio-net-pci,netdev=net0
    -netdev user,id=net0,smb=${JOB_DIR},hostfwd=tcp::3389-:3389
    -drive file=${VM_IMAGE},media=disk,if=virtio
    -snapshot
    -nographic
```

Job script links application data (win), input data (data) and run script (run.bat) into job directory and runs virtual machine.

Example run script (run.bat) for Windows virtual machine:

```
z:
cd winappl
call application.bat z:data z:output
```


Run script runs application from shared job directory (mapped as drive z:), process input data (z:data) from job directory and store output to job directory (z:output).

Run jobs

Run jobs as usual, see Resource Allocation and Job Execution. Use only full node allocation for virtualization jobs.

Running Virtual Machines

Virtualization is enabled only on compute nodes, virtualization does not work on login nodes.

Load QEMU environment module:

```
$ module add qemu
```

Get help

```
$ man qemu
```

Run virtual machine (simple)

```
$ qemu-system-x86_64 -hda linux.img -enable-kvm -cpu host -smp 16 -m 32768 -vga std -
```

```
$ qemu-system-x86_64 -hda win.img -enable-kvm -cpu host -smp 16 -m 32768 -vga std -
```

You can access virtual machine by VNC viewer (option -vnc) connecting to IP address of compute node. For VNC you must use VPN network.

Install virtual machine from iso file

```
$ qemu-system-x86_64 -hda linux.img -enable-kvm -cpu host -smp 16 -m 32768 -vga std -
```

```
$ qemu-system-x86_64 -hda win.img -enable-kvm -cpu host -smp 16 -m 32768 -vga std -
```

Run virtual machine using optimized devices, user network backend with sharing and port forwarding, in snapshot mode

```
$ qemu-system-x86_64 -drive file=linux.img,media=disk,if=virtio -enable-kvm -cpu host -
```

```
$ qemu-system-x86_64 -drive file=win.img,media=disk,if=virtio -enable-kvm -cpu host -
```

Thanks to port forwarding you can access virtual machine via SSH (Linux) or RDP (Windows) connecting to IP address of compute node (and port 2222 for SSH). You must use VPN network.

!!! Note “Note” Keep in mind, that if you use virtio devices, you must have virtio drivers installed on your virtual machine.

Networking and data sharing

For networking virtual machine we suggest to use (default) user network backend (sometimes called slirp). This network backend NATs virtual machines and provides useful services for virtual machines as DHCP, DNS, SMB sharing, port forwarding.

In default configuration IP network 10.0.2.0/24 is used, host has IP address 10.0.2.2, DNS server 10.0.2.3, SMB server 10.0.2.4 and virtual machines obtain address from range 10.0.2.15-10.0.2.31. Virtual machines have access to Anselm’s network via NAT on compute node (host).

Simple network setup

```
$ qemu-system-x86_64 ... -net nic -net user
```

(It is default when no -net options are given.)

Simple network setup with sharing and port forwarding (obsolete but simpler syntax, lower performance)

```
$ qemu-system-x86_64 ... -net nic -net user,smb=/scratch/$USER/tmp,hostfwd=tcp::3389
```

Optimized network setup with sharing and port forwarding

```
$ qemu-system-x86_64 ... -device virtio-net-pci,netdev=net0 -netdev user,id=net0,smb=
```

Advanced networking

Internet access

Sometime your virtual machine needs access to internet (install software, updates, software activation, etc). We suggest solution using Virtual Distributed Ethernet (VDE) enabled QEMU with SLIRP running on login node tunnelled to compute node. Be aware, this setup has very low performance, the worst performance of all described solutions.

Load VDE enabled QEMU environment module (unload standard QEMU module first if necessary).

```
$ module add qemu/2.1.2-vde2
```

Create virtual network switch.

```
$ vde_switch -sock /tmp/sw0 -mgmt /tmp/sw0.mgmt -daemon
```

Run SLIRP daemon over SSH tunnel on login node and connect it to virtual network switch.

```
$ dpipe vde_plugin /tmp/sw0 = ssh login1 $VDE2_DIR/bin/slirpvde -s - --dhcp &
```

Run qemu using vde network backend, connect to created virtual switch.

Basic setup (obsolete syntax)

```
$ qemu-system-x86_64 ... -net nic -net vde,sock=/tmp/sw0
```

Setup using virtio device (obsolete syntax)

```
$ qemu-system-x86_64 ... -net nic,model=virtio -net vde,sock=/tmp/sw0
```

Optimized setup

```
$ qemu-system-x86_64 ... -device virtio-net-pci,netdev=net0 -netdev vde,id=net0,sock=
```

TAP interconnect

Both user and vde network backend have low performance. For fast interconnect (10Gbps and more) of compute node (host) and virtual machine (guest) we suggest using Linux kernel TAP device.

Cluster Anselm provides TAP device tap0 for your job. TAP interconnect does not provide any services (like NAT, DHCP, DNS, SMB, etc.) just raw networking, so you should provide your services if you need them.

Run qemu with TAP network backend:

```
$ qemu-system-x86_64 ... -device virtio-net-pci,netdev=net1  
-netdev tap,id=net1,ifname=tap0,script=no,downscript=no
```

Interface tap0 has IP address 192.168.1.1 and network mask 255.255.255.0 (/24). In virtual machine use IP address from range 192.168.1.2-192.168.1.254. For your convenience some ports on tap0 interface are redirected to higher numbered ports, so you as non-privileged user can provide services on these ports.

Redirected ports:

- DNS udp/53->udp/3053, tcp/53->tcp3053
- DHCP udp/67->udp3067
- SMB tcp/139->tcp3139, tcp/445->tcp3445).

You can configure IP address of virtual machine statically or dynamically. For dynamic addressing provide your DHCP server on port 3067 of tap0 interface, you can also provide your DNS server on port 3053 of tap0 interface for example:

```
$ dnsmasq --interface tap0 --bind-interfaces -p 3053 --dhcp-alternate-port=3067,68 --
```

You can also provide your SMB services (on ports 3139, 3445) to obtain high performance data sharing.

Example smb.conf (not optimized)

```
[global]
socket address=192.168.1.1
smb ports = 3445 3139

private dir=/tmp/qemu-smb
pid directory=/tmp/qemu-smb
lock directory=/tmp/qemu-smb
state directory=/tmp/qemu-smb
ncalrpc dir=/tmp/qemu-smb/ncalrpc
log file=/tmp/qemu-smb/log.smbd
smb passwd file=/tmp/qemu-smb/smbpasswd
security = user
map to guest = Bad User
unix extensions = no
load printers = no
printing = bsd
printcap name = /dev/null
disable spoolss = yes
log level = 1
guest account = USER
[qemu]
path=/scratch/USER/tmp
read only=no
guest ok=yes
writable=yes
follow symlinks=yes
wide links=yes
force user=USER
```

(Replace USER with your login name.)

Run SMB services

```
smbd -s /tmp/qemu-smb/smb.conf
```

Virtual machine can of course have more than one network interface controller, virtual machine can

use more than one network backend. So, you can combine for example use network backend and TAP interconnect.

Snapshot mode

In snapshot mode image is not written, changes are written to temporary file (and discarded after virtual machine exits). **It is strongly recommended mode for running your jobs.** Set TMPDIR environment variable to local scratch directory for placement temporary files.

```
$ export TMPDIR=/lscratch/${PBS_JOBID}
$ qemu-system-x86_64 ... -snapshot
```

Windows guests

For Windows guests we recommend these options, life will be easier:

```
$ qemu-system-x86_64 ... -localtime -usb -usbdevice tablet
```

nVidia CUDA

A guide to nVidia CUDA programming and GPU usage

CUDA Programming on Anselm

The default programming model for GPU accelerators on Anselm is Nvidia CUDA. To set up the environment for CUDA use

```
$ module load cuda
```

If the user code is hybrid and uses both CUDA and MPI, the MPI environment has to be set up as well. One way to do this is to use the PrgEnv-gnu module, which sets up correct combination of GNU compiler and MPI library.

```
$ module load PrgEnv-gnu
```

CUDA code can be compiled directly on login1 or login2 nodes. User does not have to use compute nodes with GPU accelerator for compilation. To compile a CUDA source code, use nvcc compiler.

```
$ nvcc --version
```

CUDA Toolkit comes with large number of examples, that can be helpful to start with. To compile and test these examples user should copy them to its home directory

```
$ cd ~  
$ mkdir cuda-samples  
$ cp -R /apps/nvidia/cuda/6.5.14/samples/* ~/cuda-samples/
```

To compile an examples, change directory to the particular example (here the example used is deviceQuery) and run “make” to start the compilation

```
$ cd ~/cuda-samples/1_Uutilities/deviceQuery  
$ make
```

To run the code user can use PBS interactive session to get access to a node from qnvidia queue (note: use your project name with parameter -A in the qsub command) and execute the binary file

```
$ qsub -I -q qnvidia -A OPEN-0-0  
$ module load cuda  
$ ~/cuda-samples/1_Uutilities/deviceQuery/deviceQuery
```

Expected output of the deviceQuery example executed on a node with Tesla K20m is

```
CUDA Device Query (Runtime API) version (CUDART static linking)  
  
Detected 1 CUDA Capable device(s)  
  
Device 0: "Tesla K20m"  
CUDA Driver Version / Runtime Version 5.0 / 5.0  
CUDA Capability Major/Minor version number: 3.5  
Total amount of global memory: 4800 MBytes (5032706048 bytes)  
(13) Multiprocessors x (192) CUDA Cores/MP: 2496 CUDA Cores  
GPU Clock rate: 706 MHz (0.71 GHz)  
Memory Clock rate: 2600 Mhz  
Memory Bus Width: 320-bit  
L2 Cache Size: 1310720 bytes  
Max Texture Dimension Size (x,y,z) 1D=(65536), 2D=(65536,65536), 3D=(4096,4096,4096)
```

Max Layered Texture Size (dim) x layers 1D=(16384) x 2048, 2D=(16384,16384) x 2048
 Total amount of constant memory: 65536 bytes
 Total amount of shared memory per block: 49152 bytes
 Total number of registers available per block: 65536
 Warp size: 32
 Maximum number of threads per multiprocessor: 2048
 Maximum number of threads per block: 1024
 Maximum sizes of each dimension of a block: 1024 x 1024 x 64
 Maximum sizes of each dimension of a grid: 2147483647 x 65535 x 65535
 Maximum memory pitch: 2147483647 bytes
 Texture alignment: 512 bytes
 Concurrent copy and kernel execution: Yes with 2 copy engine(s)
 Run time limit on kernels: No
 Integrated GPU sharing Host Memory: No
 Support host page-locked memory mapping: Yes
 Alignment requirement for Surfaces: Yes
 Device has ECC support: Enabled
 Device supports Unified Addressing (UVA): Yes
 Device PCI Bus ID / PCI location ID: 2 / 0
 Compute Mode:
 < Default (multiple host threads can use ::cudaSetDevice() with device simultaneously
 deviceQuery, CUDA Driver = CUDART, CUDA Driver Version = 5.0, CUDA Runtime Version =

Code example

In this section we provide a basic CUDA based vector addition code example. You can directly copy and paste the code to test it.

```

$ vim test.cu

#define N (2048*2048)
#define THREADS_PER_BLOCK 512

#include <stdio.h>
#include <stdlib.h>

// GPU kernel function to add two vectors
__global__ void add_gpu( int *a, int *b, int *c, int n){
    int index = threadIdx.x + blockIdx.x * blockDim.x;
    if (index < n)
        c[index] = a[index] + b[index];
}

// CPU function to add two vectors
void add_cpu (int *a, int *b, int *c, int n) {
    for (int i=0; i < n; i++)
        c[i] = a[i] + b[i];
}

// CPU function to generate a vector of random integers
void random_ints (int *a, int n) {
    for (int i = 0; i < n; i++)

```

```

    a[i] = rand() % 10000; // random number between 0 and 9999
}

// CPU function to compare two vectors
int compare_ints( int *a, int *b, int n ){
    int pass = 0;
    for (int i = 0; i < N; i++){
        if (a[i] != b[i]) {
            printf("Value mismatch at location %d, values %d and %dn",i, a[i], b[i]);
            pass = 1;
        }
    }
    if (pass == 0) printf ("Test passedn"); else printf ("Test Failedn");
    return pass;
}

int main( void ) {

    int *a, *b, *c; // host copies of a, b, c
    int *dev_a, *dev_b, *dev_c; // device copies of a, b, c
    int size = N * sizeof( int ); // we need space for N integers

    // Allocate GPU/device copies of dev_a, dev_b, dev_c
    cudaMalloc( (void**)&dev_a, size );
    cudaMalloc( (void**)&dev_b, size );
    cudaMalloc( (void**)&dev_c, size );

    // Allocate CPU/host copies of a, b, c
    a = (int*)malloc( size );
    b = (int*)malloc( size );
    c = (int*)malloc( size );

    // Fill input vectors with random integer numbers
    random_ints( a, N );
    random_ints( b, N );

    // copy inputs to device
    cudaMemcpy( dev_a, a, size, cudaMemcpyHostToDevice );
    cudaMemcpy( dev_b, b, size, cudaMemcpyHostToDevice );

    // launch add_gpu() kernel with blocks and threads
    add_gpu<<< N/THREADS_PER_BLOCK, THREADS_PER_BLOCK >>>( dev_a, dev_b, dev_c, N );

    // copy device result back to host copy of c
    cudaMemcpy( c, dev_c, size, cudaMemcpyDeviceToHost );

    //Check the results with CPU implementation
    int *c_h; c_h = (int*)malloc( size );
    add_cpu (a, b, c_h, N);
    compare_ints(c, c_h, N);

    // Clean CPU memory allocations

```

```

    free( a ); free( b ); free( c ); free (c_h);

    // Clean GPU memory allocations
    cudaFree( dev_a );
    cudaFree( dev_b );
    cudaFree( dev_c );

    return 0;
}

```

This code can be compiled using following command

```
$ nvcc test.cu -o test_cuda
```

To run the code use interactive PBS session to get access to one of the GPU accelerated nodes

```

$ qsub -I -q qnvidia -A OPEN-0-0
$ module load cuda
$ ./test_cuda

```

CUDA Libraries

CuBLAS

The NVIDIA CUDA Basic Linear Algebra Subroutines (cuBLAS) library is a GPU-accelerated version of the complete standard BLAS library with 152 standard BLAS routines. Basic description of the library together with basic performance comparison with MKL can be found [here](#).

CuBLAS example: SAXPY

SAXPY function multiplies the vector x by the scalar alpha and adds it to the vector y overwriting the latest vector with the result. The description of the cuBLAS function can be found in NVIDIA CUDA documentation. Code can be pasted in the file and compiled without any modification.

```

/* Includes, system */
#include <stdio.h>
#include <stdlib.h>

/* Includes, cuda */
#include <cuda_runtime.h>
#include < cublas_v2.h>

/* Vector size */
#define N (32)

/* Host implementation of a simple version of saxpy */
void saxpy(int n, float alpha, const float *x, float *y)
{
    for (int i = 0; i < n; ++i)
        y[i] = alpha*x[i] + y[i];
}

/* Main */
int main(int argc, char **argv)
{

```



```

float *h_X, *h_Y, *h_Y_ref;
float *d_X = 0;
float *d_Y = 0;

const float alpha = 1.0f;
int i;

cublasHandle_t handle;

/* Initialize CUBLAS */
printf("simpleCUBLAS test running..n");
cublasCreate(&handle);

/* Allocate host memory for the matrices */
h_X = (float *)malloc(N * sizeof(h_X[0]));
h_Y = (float *)malloc(N * sizeof(h_Y[0]));
h_Y_ref = (float *)malloc(N * sizeof(h_Y_ref[0]));

/* Fill the matrices with test data */
for (i = 0; i < N; i++)
{
    h_X[i] = rand() / (float)RAND_MAX;
    h_Y[i] = rand() / (float)RAND_MAX;
    h_Y_ref[i] = h_Y[i];
}

/* Allocate device memory for the matrices */
cudaMalloc((void **)&d_X, N * sizeof(d_X[0]));
cudaMalloc((void **)&d_Y, N * sizeof(d_Y[0]));

/* Initialize the device matrices with the host matrices */
cublasSetVector(N, sizeof(h_X[0]), h_X, 1, d_X, 1);
cublasSetVector(N, sizeof(h_Y[0]), h_Y, 1, d_Y, 1);

/* Performs operation using plain C code */
saxpy(N, alpha, h_X, h_Y_ref);

/* Performs operation using cublas */
cublasSaxpy(handle, N, &alpha, d_X, 1, d_Y, 1);

/* Read the result back */
cublasGetVector(N, sizeof(h_Y[0]), d_Y, 1, h_Y, 1);

/* Check result against reference */
for (i = 0; i < N; ++i)
    printf("CPU res = %f t GPU res = %f t diff = %f n", h_Y_ref[i], h_Y[i], h_Y_r

/* Memory clean up */
free(h_X); free(h_Y); free(h_Y_ref);
cudaFree(d_X); cudaFree(d_Y);

/* Shutdown */

```

```
    cublasDestroy(handle);  
}
```

!!! Note “Note” Please note: cuBLAS has its own function for data transfers between CPU and GPU memory:

- [cublasSetVector](<http://docs.nvidia.com/cuda/cublas/index.html#cublasSetVector>)! [external link]
- [cublasGetVector](<http://docs.nvidia.com/cuda/cublas/index.html#cublasGetVector>)! [external link]

To compile the code using NVCC compiler a “-lcublas” compiler flag has to be specified:

```
$ module load cuda  
$ nvcc -lcublas test_cublas.cu -o test_cublas_nvcc
```

To compile the same code with GCC:

```
$ module load cuda  
$ gcc -std=c99 test_cublas.c -o test_cublas_icc -lcublas -lcudart
```

To compile the same code with Intel compiler:

```
$ module load cuda intel  
$ icc -std=c99 test_cublas.c -o test_cublas_icc -lcublas -lcudart
```

Intel MKL

Intel Math Kernel Library

Intel Math Kernel Library (Intel MKL) is a library of math kernel subroutines, extensively threaded and optimized for maximum performance. Intel MKL provides these basic math kernels:

- BLAS (level 1, 2, and 3) and LAPACK linear algebra routines, offering vector, vector-matrix, and matrix-matrix operations.
- The PARDISO direct sparse solver, an iterative sparse solver, and supporting sparse BLAS (level 1, 2, and 3) routines for solving sparse systems of equations.
- ScaLAPACK distributed processing linear algebra routines for Linux* and Windows* operating systems, as well as the Basic Linear Algebra Communications Subprograms (BLACS) and the Parallel Basic Linear Algebra Subprograms (PBLAS).
- Fast Fourier transform (FFT) functions in one, two, or three dimensions with support for mixed radices (not limited to sizes that are powers of 2), as well as distributed versions of these functions.
- Vector Math Library (VML) routines for optimized mathematical operations on vectors.
- Vector Statistical Library (VSL) routines, which offer high-performance vectorized random number generators (RNG) for several probability distributions, convolution and correlation routines, and summary statistics functions.
- Data Fitting Library, which provides capabilities for spline-based approximation of functions, derivatives and integrals of functions, and search.
- Extended Eigensolver, a shared memory version of an eigensolver based on the Feast Eigenvalue Solver.

For details see the Intel MKL Reference Manual [\[1\]](#).

Intel MKL version 13.5.192 is available on Anselm

```
$ module load mkl
```

The module sets up environment variables, required for linking and running mkl enabled applications. The most important variables are the \$MKLROOT, \$MKL_INC_DIR, \$MKL_LIB_DIR and \$MKL_EXAMPLES

!!! Note “Note” The MKL library may be linked using any compiler. With intel compiler use -mkl option to link default threaded MKL.

Interfaces

The MKL library provides number of interfaces. The fundamental once are the LP64 and ILP64. The Intel MKL ILP64 libraries use the 64-bit integer type (necessary for indexing large arrays, with more than $2^{31}-1$ elements), whereas the LP64 libraries index arrays with the 32-bit integer type.

Interface	Integer type
LP64	32-bit, int, integer(kind=4), MPI_INT
ILP64	64-bit, long int, integer(kind=8), MPI_INT64

Linking

Linking MKL libraries may be complex. Intel mkl link line advisor [\[2\]](#) helps. See also examples below.

You will need the mkl module loaded to run the mkl enabled executable. This may be avoided, by compiling library search paths into the executable. Include `rpath` on the compile line:

```
$ icc .... -Wl,-rpath=$LIBRARY_PATH ...
```

Threading

!!! Note “Note” Advantage in using the MKL library is that it brings threaded parallelization to applications that are otherwise not parallel.

For this to work, the application must link the threaded MKL library (default). Number and behaviour of MKL threads may be controlled via the OpenMP environment variables, such as `OMP_NUM_THREADS` and `KMP_AFFINITY`. `MKL_NUM_THREADS` takes precedence over `OMP_NUM_THREADS`

```
$ export OMP_NUM_THREADS=16
$ export KMP_AFFINITY=granularity=fine,compact,1,0
```

The application will run with 16 threads with affinity optimized for fine grain parallelization.

Examples

Number of examples, demonstrating use of the MKL library and its linking is available on Anselm, in the `$MKL_EXAMPLES` directory. In the examples below, we demonstrate linking MKL to Intel and GNU compiled program for multi-threaded matrix multiplication.

Working with examples

```
$ module load intel
$ module load mkl
$ cp -a $MKL_EXAMPLES/cblas /tmp/
$ cd /tmp/cblas

$ make sointel64 function=cblas_dgemm
```

In this example, we compile, link and run the `cblas_dgemm` example, demonstrating use of MKL example suite installed on Anselm.

Example: MKL and Intel compiler

```
$ module load intel
$ module load mkl
$ cp -a $MKL_EXAMPLES/cblas /tmp/
$ cd /tmp/cblas
$
$ icc -w source/cblas_dgemmx.c source/common_func.c -mkl -o cblas_dgemmx.x
$ ./cblas_dgemmx.x data/cblas_dgemmx.d
```

In this example, we compile, link and run the `cblas_dgemm` example, demonstrating use of MKL with `icc -mkl` option. Using the `-mkl` option is equivalent to:

```
$ icc -w source/cblas_dgemmx.c source/common_func.c -o cblas_dgemmx.x
-I$MKL_INC_DIR -L$MKL_LIB_DIR -lmkl_intel_lp64 -lmkl_intel_thread -lmkl_core -liomp5
```

In this example, we compile and link the `cblas_dgemm` example, using LP64 interface to threaded MKL and Intel OMP threads implementation.

Example: MKL and GNU compiler

```
$ module load gcc
$ module load mkl
$ cp -a $MKL_EXAMPLES/cblas /tmp/
$ cd /tmp/cblas

$ gcc -w source/cblas_dgemmx.c source/common_func.c -o cblas_dgemmx.x
-lmkl_intel_lp64 -lmkl_gnu_thread -lmkl_core -lgomp -lm

$ ./cblas_dgemmx.x data/cblas_dgemmx.d
```

In this example, we compile, link and run the `cblas_dgemm` example, using LP64 interface to threaded MKL and gnu OMP threads implementation.

MKL and MIC accelerators

The MKL is capable to automatically offload the computations to the MIC accelerator. See section Intel XeonPhi for details.

Further reading

Read more on Intel website [🔗](#), in particular the MKL users guide [🔗](#).

Intel Debugger

Debugging serial applications

The intel debugger version 13.0 is available, via module intel. The debugger works for applications compiled with C and C++ compiler and the ifort fortran 77/90/95 compiler. The debugger provides java GUI environment. Use X display for running the GUI.

```
$ module load intel
$ idb
```

The debugger may run in text mode. To debug in text mode, use

```
$ idbc
```

To debug on the compute nodes, module intel must be loaded. The GUI on compute nodes may be accessed using the same way as in the GUI section

Example:

```
$ qsub -q qexp -l select=1:ncpus=16 -X -I
qsub: waiting for job 19654.srv11 to start
qsub: job 19654.srv11 ready

$ module load intel
$ module load java
$ icc -O0 -g myprog.c -o myprog.x
$ idb ./myprog.x
```

In this example, we allocate 1 full compute node, compile program myprog.c with debugging options -O0 -g and run the idb debugger interactively on the myprog.x executable. The GUI access is via X11 port forwarding provided by the PBS workload manager.

Debugging parallel applications

Intel debugger is capable of debugging multithreaded and MPI parallel programs as well.

Small number of MPI ranks

For debugging small number of MPI ranks, you may execute and debug each rank in separate xterm terminal (do not forget the X display. Using Intel MPI, this may be done in following way:

```
$ qsub -q qexp -l select=2:ncpus=16 -X -I
qsub: waiting for job 19654.srv11 to start
qsub: job 19655.srv11 ready

$ module load intel impi
$ mpirun -ppn 1 -hostfile $PBS_NODEFILE --enable-x xterm -e idbc ./mympprog.x
```

In this example, we allocate 2 full compute node, run xterm on each node and start idb debugger in command line mode, debugging two ranks of mympprog.x application. The xterm will pop up for each rank, with idb prompt ready. The example is not limited to use of Intel MPI

Large number of MPI ranks

Run the idb debugger from within the MPI debug option. This will cause the debugger to bind to all ranks and provide aggregated outputs across the ranks, pausing execution automatically just after startup. You may then set break points and step the execution manually. Using Intel MPI:

```
$ qsub -q qexp -l select=2:ncpus=16 -X -I
qsub: waiting for job 19654.srv11 to start
qsub: job 19655.srv11 ready

$ module load intel impi
$ mpirun -n 32 -idb ./mympiprogram.x
```

Debugging multithreaded application

Run the idb debugger in GUI mode. The menu Parallel contains number of tools for debugging multiple threads. One of the most useful tools is the **Serialize Execution** tool, which serializes execution of concurrent threads for easy orientation and identification of concurrency related bugs.

Further information

Exhaustive manual on idb features and usage is published at Intel website [🔗](#)

Intel IPP

Intel Integrated Performance Primitives

Intel Integrated Performance Primitives, version 7.1.1, compiled for AVX vector instructions is available, via module `ipp`. The IPP is a very rich library of highly optimized algorithmic building blocks for media and data applications. This includes signal, image and frame processing algorithms, such as FFT, FIR, Convolution, Optical Flow, Hough transform, Sum, MinMax, as well as cryptographic functions, linear algebra functions and many more.

!!! Note “Note” Check out IPP before implementing own math functions for data processing, it is likely already there.

```
$ module load ipp
```

The module sets up environment variables, required for linking and running ipp enabled applications.

IPP example

```
#include "ipp.h"
#include <stdio.h>
int main(int argc, char* argv[])
{
    const IppLibraryVersion *lib;
    Ipp64u fm;
    IppStatus status;

    status= ippInit();           //IPP initialization with the best optimization
    if( status != ippStsNoErr ) {
        printf("IppInit() Error:n");
        printf("%sn", ippGetStatusString(status) );
        return -1;
    }

    //Get version info
    lib = ippiGetLibVersion();
    printf("%s %sn", lib->Name, lib->Version);

    //Get CPU features enabled with selected library level
    fm=ippGetEnabledCpuFeatures();
    printf("SSE      :%cn", (fm>1)&1?'Y':'N');
    printf("SSE2     :%cn", (fm>2)&1?'Y':'N');
    printf("SSE3      :%cn", (fm>3)&1?'Y':'N');
    printf("SSSE3     :%cn", (fm>4)&1?'Y':'N');
    printf("SSE41     :%cn", (fm>6)&1?'Y':'N');
    printf("SSE42     :%cn", (fm>7)&1?'Y':'N');
    printf("AVX       :%cn", (fm>8)&1?'Y':'N');
    printf("AVX2      :%cn", (fm>15)&1?'Y':'N' );
    printf("-----n");
    printf("OS Enabled AVX :%cn", (fm>9)&1?'Y':'N');
    printf("AES          :%cn", (fm>10)&1?'Y':'N');
    printf("CLMUL        :%cn", (fm>11)&1?'Y':'N');
```



```

printf("RDRAND          :%cn", (fm>13)&1?'Y':'N');
printf("F16C           :%cn", (fm>14)&1?'Y':'N');

return 0;
}

```

Compile above example, using any compiler and the ipp module.

```

$ module load intel
$ module load ipp

$ icc testipp.c -o testipp.x -lippi -lipps -lippcore

```

You will need the ipp module loaded to run the ipp enabled executable. This may be avoided, by compiling library search paths into the executable

```

$ module load intel
$ module load ipp

$ icc testipp.c -o testipp.x -Wl,-rpath=$LIBRARY_PATH -lippi -lipps -lippcore

```

Code samples and documentation

Intel provides number of Code Samples for IPP [🔗](#), illustrating use of IPP.

Read full documentation on IPP on Intel website, [🔗](#) in particular the IPP Reference manual. [🔗](#)

Intel TBB

Intel Threading Building Blocks

Intel Threading Building Blocks (Intel TBB) is a library that supports scalable parallel programming using standard ISO C++ code. It does not require special languages or compilers. To use the library, you specify tasks, not threads, and let the library map tasks onto threads in an efficient manner. The tasks are executed by a runtime scheduler and may be offloaded to MIC accelerator.

Intel TBB version 4.1 is available on Anselm

```
$ module load tbb
```

The module sets up environment variables, required for linking and running tbb enabled applications.

!!! Note “Note” Link the tbb library, using -ltbb

Examples

Number of examples, demonstrating use of TBB and its built-in scheduler is available on Anselm, in the \$TBB_EXAMPLES directory.

```
$ module load intel
$ module load tbb
$ cp -a $TBB_EXAMPLES/common $TBB_EXAMPLES/parallel_reduce /tmp/
$ cd /tmp/parallel_reduce/primes
$ icc -O2 -DNDEBUG -o primes.x main.cpp primes.cpp -ltbb
$ ./primes.x
```

In this example, we compile, link and run the primes example, demonstrating use of parallel task-based reduce in computation of prime numbers.

You will need the tbb module loaded to run the tbb enabled executable. This may be avoided, by compiling library search paths into the executable.

```
$ icc -O2 -o primes.x main.cpp primes.cpp -Wl,-rpath=$LIBRARY_PATH -ltbb
```

Further reading

Read more on Intel website, http://software.intel.com/sites/products/documentation/doclib/tbb_sa/help/index.htm 

Intel Parallel Studio

The Anselm cluster provides following elements of the Intel Parallel Studio XE

Intel Parallel Studio XE
Intel Compilers
Intel Debugger
Intel MKL Library
Intel Integrated Performance Primitives Library
Intel Threading Building Blocks Library

Intel compilers

The Intel compilers version 13.1.3 are available, via module intel. The compilers include the icc C and C++ compiler and the ifort fortran 77/90/95 compiler.

```
$ module load intel
$ icc -v
$ ifort -v
```

Read more at the Intel Compilers page.

Intel debugger

The intel debugger version 13.0 is available, via module intel. The debugger works for applications compiled with C and C++ compiler and the ifort fortran 77/90/95 compiler. The debugger provides java GUI environment. Use X display for running the GUI.

```
$ module load intel
$ idb
```

Read more at the Intel Debugger page.

Intel Math Kernel Library

Intel Math Kernel Library (Intel MKL) is a library of math kernel subroutines, extensively threaded and optimized for maximum performance. Intel MKL unites and provides these basic components: BLAS, LAPACK, ScaLapack, PARDISO, FFT, VML, VSL, Data fitting, Feast Eigensolver and many more.

```
$ module load mkl
```

Read more at the Intel MKL page.

Intel Integrated Performance Primitives

Intel Integrated Performance Primitives, version 7.1.1, compiled for AVX is available, via module ipp. The IPP is a library of highly optimized algorithmic building blocks for media and data applications. This includes signal, image and frame processing algorithms, such as FFT, FIR, Convolution, Optical Flow, Hough transform, Sum, MinMax and many more.

```
$ module load ipp
```

Read more at the Intel IPP page.

Intel Threading Building Blocks

Intel Threading Building Blocks (Intel TBB) is a library that supports scalable parallel programming using standard ISO C++ code. It does not require special languages or compilers. It is designed to promote scalable data parallel programming. Additionally, it fully supports nested parallelism, so you can build larger parallel components from smaller parallel components. To use the library, you specify tasks, not threads, and let the library map tasks onto threads in an efficient manner.

```
$ module load tbb
```

Read more at the Intel TBB page.

Intel Compilers

The Intel compilers version 13.1.1 are available, via module intel. The compilers include the icc C and C++ compiler and the ifort fortran 77/90/95 compiler.

```
$ module load intel
$ icc -v
$ ifort -v
```

The intel compilers provide for vectorization of the code, via the AVX instructions and support threading parallelization via OpenMP


For maximum performance on the Anselm cluster, compile your programs using the AVX instructions, with reporting where the vectorization was used. We recommend following compilation options for high performance

```
$ icc -ipo -O3 -vec -xAVX -vec-report1 myprog.c mysubroutines.c -o myprog.x
$ ifort -ipo -O3 -vec -xAVX -vec-report1 myprog.f mysubroutines.f -o myprog.x
```

In this example, we compile the program enabling interprocedural optimizations between source files (-ipo), aggressive loop optimizations (-O3) and vectorization (-vec -xAVX)

The compiler recognizes the omp, simd, vector and ivdep pragmas for OpenMP parallelization and AVX vectorization. Enable the OpenMP parallelization by the **-openmp** compiler switch.

```
$ icc -ipo -O3 -vec -xAVX -vec-report1 -openmp myprog.c mysubroutines.c -o myprog.x
$ ifort -ipo -O3 -vec -xAVX -vec-report1 -openmp myprog.f mysubroutines.f -o myprog.x
```

Read more at <http://software.intel.com/sites/products/documentation/doclib/stdxe/2013/composerxe/compiler/cpp-lin/index.htm> 

Sandy Bridge/Haswell binary compatibility

Anselm nodes are currently equipped with Sandy Bridge CPUs, while Salomon will use Haswell architecture. >The new processors are backward compatible with the Sandy Bridge nodes, so all programs that ran on the Sandy Bridge processors, should also run on the new Haswell nodes. >To get optimal performance out of the Haswell processors a program should make use of the special AVX2 instructions for this processor. One can do this by recompiling codes with the compiler flags >designated to invoke these instructions. For the Intel compiler suite, there are two ways of doing this:

- Using compiler flag (both for Fortran and C): -xCORE-AVX2. This will create a binary with AVX2 instructions, specifically for the Haswell processors. Note that the executable will not run on Sandy Bridge nodes.
- Using compiler flags (both for Fortran and C): -xAVX -xCORE-AVX2. This will generate multiple, feature specific auto-dispatch code paths for Intel® processors, if there is a performance benefit. So this binary will run both on Sandy Bridge and Haswell processors. During runtime it will be decided which path to follow, dependent on which processor you are running on. In general this will result in larger binaries.

FFTW

The discrete Fourier transform in one or more dimensions, MPI parallel

FFTW is a C subroutine library for computing the discrete Fourier transform in one or more dimensions, of arbitrary input size, and of both real and complex data (as well as of even/odd data, i.e. the discrete cosine/sine transforms or DCT/DST). The FFTW library allows for MPI parallel, in-place discrete Fourier transform, with data distributed over number of nodes.

Two versions, **3.3.3** and **2.1.5** of FFTW are available on Anselm, each compiled for **Intel MPI** and **OpenMPI** using **intel** and **gnu** compilers. These are available via modules:

Version	Parallelization
---------	-----------------

FFTW3	pthread, gcc3.3.3OpenMP
-------	----------------------------

FFTW3	pthread, icc3.3.3OpenMP
-------	----------------------------

FFTW2	pthread gcc2.1.5
-------	---------------------

FFTW2	pthread icc2.1.5
-------	---------------------

FFTW3	OpenMPI gcc3.3.3
-------	---------------------

FFTW3	intel icc3.3.3MPI
-------	----------------------

FFTW2	OpenMPI gcc2.1.5
-------	---------------------

FFTW2	intelMPI gcc2.1.5
-------	----------------------

```
$ module load fftw3
```

The module sets up environment variables, required for linking and running fftw enabled applications. Make sure that the choice of fftw module is consistent with your choice of MPI library. Mixing MPI of different implementations may have unpredictable results.

Example

```
#include <fftw3-mpi.h>
int main(int argc, char **argv)
{
    const ptrdiff_t N0 = 100, N1 = 1000;
    fftw_plan plan;
```

```

fftw_complex *data;
ptrdiff_t alloc_local, local_n0, local_0_start, i, j;

MPI_Init(&argc, &argv);
fftw_mpi_init();

/* get local data size and allocate */
alloc_local = fftw_mpi_local_size_2d(N0, N1, MPI_COMM_WORLD,
                                     &local_n0, &local_0_start);
data = fftw_alloc_complex(alloc_local);

/* create plan for in-place forward DFT */
plan = fftw_mpi_plan_dft_2d(N0, N1, data, data, MPI_COMM_WORLD,
                             FFTW_FORWARD, FFTW_ESTIMATE);

/* initialize data */
for (i = 0; i < local_n0; ++i) for (j = 0; j < N1; ++j)
{   data[i*N1 + j][0] = i;
    data[i*N1 + j][1] = j; }

/* compute transforms, in-place, as many times as desired */
fftw_execute(plan);

fftw_destroy_plan(plan);

MPI_Finalize();
}

```

Load modules and compile:

```

$ module load impi intel
$ module load fftw3-mpi

$ mpicc testfftw3mpi.c -o testfftw3mpi.x -Wl,-rpath=$LIBRARY_PATH -lfftw3_mpi

```

Run the example as Intel MPI program.

Read more on FFTW usage on the FFTW website. [🔗](#)

PETSc

PETSc is a suite of building blocks for the scalable solution of scientific and engineering applications modelled by partial differential equations. It supports MPI, shared memory, and GPUs through CUDA or OpenCL, as well as hybrid MPI-shared memory or MPI-GPU parallelism.

Introduction

PETSc (Portable, Extensible Toolkit for Scientific Computation) is a suite of building blocks (data structures and routines) for the scalable solution of scientific and engineering applications modelled by partial differential equations. It allows thinking in terms of high-level objects (matrices) instead of low-level objects (raw arrays). Written in C language but can also be called from FORTRAN, C++, Python and Java codes. It supports MPI, shared memory, and GPUs through CUDA or OpenCL, as well as hybrid MPI-shared memory or MPI-GPU parallelism.

Resources

- project webpage [🔗](#)
- documentation [🔗](#)
 - PETSc Users Manual (PDF) [🔗](#)
 - index of all manual pages [🔗](#)
- PRACE Video Tutorial [part1](#) [🔗](#), [part2](#) [🔗](#), [part3](#) [🔗](#), [part4](#) [🔗](#), [part5](#) [🔗](#)

Modules

You can start using PETSc on Anselm by loading the PETSc module. Module names obey this pattern:

```
# module load petsc/version-compiler-mpi-blas-variant, e.g.  
module load petsc/3.4.4-icc-mpi-mkl-opt
```

where **variant** is replaced by one of {**dbg**, **opt**, **threads-dbg**, **threads-opt**}. The **opt** variant is compiled without debugging information (no **-g** option) and with aggressive compiler optimizations (**-O3 -xAVX**). This variant is suitable for performance measurements and production runs. In all other cases use the debug (**dbg**) variant, because it contains debugging information, performs validations and self-checks, and provides a clear stack trace and message in case of an error. The other two variants **threads-dbg** and **threads-opt** are **dbg** and **opt**, respectively, built with OpenMP and pthreads threading support [🔗](#).

External libraries

PETSc needs at least MPI, BLAS and LAPACK. These dependencies are currently satisfied with Intel MPI and Intel MKL in Anselm **petsc** modules.

PETSc can be linked with a plethora of external numerical libraries [🔗](#), extending PETSc functionality, e.g. direct linear system solvers, preconditioners or partitioners. See below a list of libraries currently included in Anselm **petsc** modules.

All these libraries can be used also alone, without PETSc. Their static or shared program libraries are available in `$PETSC_DIR/$PETSC_ARCH/lib` and header files in `$PETSC_DIR/$PETSC_ARCH/include`.

PETSC_DIR and PETSC_ARCH are environment variables pointing to a specific PETSc instance based on the petsc module loaded.

Libraries linked to PETSc on Anselm (as of 11 April 2015)

- dense linear algebra
 - Elemental 
- sparse linear system solvers
 - Intel MKL Pardiso 
 - MUMPS 
 - PaStiX 
 - SuiteSparse 
 - SuperLU 
 - SuperLU_Dist 
- input/output
 - ExodusII 
 - HDF5 
 - NetCDF 
- partitioning
 - Chaco 
 - METIS 
 - ParMETIS 
 - PT-Scotch 
- preconditioners & multigrid
 - Hypre 
 - Trilinos ML 
 - SPAI - Sparse Approximate Inverse 

Intel numerical libraries

Intel libraries for high performance in numerical computing

Intel Math Kernel Library

Intel Math Kernel Library (Intel MKL) is a library of math kernel subroutines, extensively threaded and optimized for maximum performance. Intel MKL unites and provides these basic components: BLAS, LAPACK, ScaLapack, PARDISO, FFT, VML, VSL, Data fitting, Feast Eigensolver and many more.

```
$ module load mkl
```

Read more at the Intel MKL page.

Intel Integrated Performance Primitives

Intel Integrated Performance Primitives, version 7.1.1, compiled for AVX is available, via module `ipp`. The IPP is a library of highly optimized algorithmic building blocks for media and data applications. This includes signal, image and frame processing algorithms, such as FFT, FIR, Convolution, Optical Flow, Hough transform, Sum, MinMax and many more.

```
$ module load ipp
```

Read more at the Intel IPP page.

Intel Threading Building Blocks

Intel Threading Building Blocks (Intel TBB) is a library that supports scalable parallel programming using standard ISO C++ code. It does not require special languages or compilers. It is designed to promote scalable data parallel programming. Additionally, it fully supports nested parallelism, so you can build larger parallel components from smaller parallel components. To use the library, you specify tasks, not threads, and let the library map tasks onto threads in an efficient manner.

```
$ module load tbb
```

Read more at the Intel TBB page.

MAGMA for Intel Xeon Phi

Next generation dense algebra library for heterogeneous systems with accelerators

Compiling and linking with MAGMA

To be able to compile and link code with MAGMA library user has to load following module:

```
$ module load magma/1.3.0-mic
```

To make compilation more user friendly module also sets these two environment variables:

!!! Note “Note” MAGMA_INC - contains paths to the MAGMA header files (to be used for compilation step)

!!! Note “Note” MAGMA_LIBS - contains paths to MAGMA libraries (to be used for linking step).

Compilation example:

```
$ icc -mkl -O3 -DHAVE_MIC -DADD_ -Wall $MAGMA_INC -c testing_dgetrf_mic.cpp -o testing
```

```
$ icc -mkl -O3 -DHAVE_MIC -DADD_ -Wall -fPIC -Xlinker -zmuldefs -Wall -DNOCHANGE -DHO
```

Running MAGMA code

MAGMA implementation for Intel MIC requires a MAGMA server running on accelerator prior to executing the user application. The server can be started and stopped using following scripts:

!!! Note “Note” To start MAGMA server use: `$MAGMAROOT/start_magma_server`

!!! Note “Note” To stop the server use: `$MAGMAROOT/stop_magma_server`

!!! Note “Note” For deeper understanding how the MAGMA server is started, see the following script: `$MAGMAROOT/launch_anselm_from_mic.sh`

To test if the MAGMA server runs properly we can run one of examples that are part of the MAGMA installation:

```
[user@cn204 ~]$ $MAGMAROOT/testing/testing_dgetrf_mic
```

```
[user@cn204 ~]$ export OMP_NUM_THREADS=16
```

```
[lriha@cn204 ~]$ $MAGMAROOT/testing/testing_dgetrf_mic
```

```
Usage: /apps/libs/magma-mic/magmamic-1.3.0/testing/testing_dgetrf_mic [options] [-h|-
```

M	N	CPU GFlop/s (sec)		MAGMA GFlop/s (sec)		PA-LU /(A *N)
1088	1088	---	(---)	13.93	(0.06)	---
2112	2112	---	(---)	77.85	(0.08)	---
3136	3136	---	(---)	183.21	(0.11)	---
4160	4160	---	(---)	227.52	(0.21)	---
5184	5184	---	(---)	258.61	(0.36)	---
6208	6208	---	(---)	333.12	(0.48)	---
7232	7232	---	(---)	416.52	(0.61)	---
8256	8256	---	(---)	446.97	(0.84)	---

9280	9280	---	(---)	461.15	(1.16)	---
10304	10304	---	(---)	500.70	(1.46)	---

!!! Note “Note” Please note: MAGMA contains several benchmarks and examples that can be found in: **\$MAGMAROOT/testing/**

!!! Note “Note” MAGMA relies on the performance of all CPU cores as well as on the performance of the accelerator. Therefore on Anselm number of CPU OpenMP threads has to be set to 16: **export OMP_NUM_THREADS=16**


See more details at MAGMA home page [🔗](#).

References

[1] MAGMA MIC: Linear Algebra Library for Intel Xeon Phi Coprocessors, Jack Dongarra et. al, http://icl.utk.edu/projectsfiles/magma/pubs/24-MAGMA_MIC_03.pdf [🔗](#)

HDF5

Hierarchical Data Format library. Serial and MPI parallel version.

HDF5 (Hierarchical Data Format)  is a general purpose library and file format for storing scientific data. HDF5 can store two primary objects: datasets and groups. A dataset is essentially a multidimensional array of data elements, and a group is a structure for organizing objects in an HDF5 file. Using these two basic objects, one can create and store almost any kind of scientific data structure, such as images, arrays of vectors, and structured and unstructured grids. You can also mix and match them in HDF5 files according to your needs.

Versions **1.8.11** and **1.8.13** of HDF5 library are available on Anselm, compiled for **Intel MPI** and **OpenMPI** using **intel** and **gnu** compilers. These are available via modules:

VersionParallelization

HDF5 pthread


icc

se-

rial

```
$ module load hdf5-parallel
```

The module sets up environment variables, required for linking and running HDF5 enabled applications. Make sure that the choice of HDF5 module is consistent with your choice of MPI library. Mixing MPI of different implementations may have unpredictable results.

!!! Note “Note” Be aware, that GCC version of **HDF5 1.8.11** has serious performance issues, since it’s compiled with -O0 optimization flag. This version is provided only for testing of code compiled only by GCC and IS NOT recommended for production computations. For more informations, please see: <http://www.hdfgroup.org/ftp/HDF5/prev-releases/ReleaseFiles/release5-1811> 

All GCC versions of ****HDF5 1.8.13**** are not affected by the bug, are compiled with -O3 optimization

Example

```
#include "hdf5.h"
#define FILE "dset.h5"

int main() {

    hid_t      file_id, dataset_id, dataspace_id;  /* identifiers */
    hsize_t    dims[2];
    herr_t     status;
    int        i, j, dset_data[4][6];

    /* Create a new file using default properties. */
    file_id = H5Fcreate(FILE, H5F_ACC_TRUNC, H5P_DEFAULT, H5P_DEFAULT);

    /* Create the data space for the dataset. */
    dims[0] = 4;
    dims[1] = 6;
```

```

dataspace_id = H5Screate_simple(2, dims, NULL);

/* Initialize the dataset. */
for (i = 0; i < 4; i++)
    for (j = 0; j < 6; j++)
        dset_data[i][j] = i * 6 + j + 1;

/* Create the dataset. */
dataset_id = H5Dcreate2(file_id, "/dset", H5T_STD_I32BE, dataspace_id,
                        H5P_DEFAULT, H5P_DEFAULT, H5P_DEFAULT);

/* Write the dataset. */
status = H5Dwrite(dataset_id, H5T_NATIVE_INT, H5S_ALL, H5S_ALL, H5P_DEFAULT,
                  dset_data);

status = H5Dread(dataset_id, H5T_NATIVE_INT, H5S_ALL, H5S_ALL, H5P_DEFAULT,
                  dset_data);

/* End access to the dataset and release resources used by it. */
status = H5Dclose(dataset_id);

/* Terminate access to the data space. */
status = H5Sclose(dataspace_id);

/* Close the file. */
status = H5Fclose(file_id);
}

```

Load modules and compile:

```

$ module load intel impi
$ module load hdf5-parallel

$ mpicc hdf5test.c -o hdf5test.x -Wl,-rpath=$LIBRARY_PATH $HDF5_INC $HDF5_SHLIB

```

Run the example as Intel MPI program.

For further informations, please see the website: <http://www.hdfgroup.org/HDF5/> 

Trilinos

Packages for large scale scientific and engineering problems. Provides MPI and hybrid parallelization.


Introduction

Trilinos is a collection of software packages for the numerical solution of large scale scientific and engineering problems. It is based on C++ and features modern object-oriented design. Both serial as well as parallel computations based on MPI and hybrid parallelization are supported within Trilinos packages.

Installed packages

Current Trilinos installation on ANSELM contains (among others) the following main packages

- **Epetra** - core linear algebra package containing classes for manipulation with serial and distributed vectors, matrices, and graphs. Dense linear solvers are supported via interface to BLAS and LAPACK (Intel MKL on ANSELM). Its extension **EpetraExt** contains e.g. methods for matrix-matrix multiplication.
- **Tpetra** - next-generation linear algebra package. Supports 64bit indexing and arbitrary data type using C++ templates.
- **Belos** - library of various iterative solvers (CG, block CG, GMRES, block GMRES etc.).
- **Amesos** - interface to direct sparse solvers.
- **Anasazi** - framework for large-scale eigenvalue algorithms.
- **IFPACK** - distributed algebraic preconditioner (includes e.g. incomplete LU factorization)
- **Teuchos** - common tools packages. This package contains classes for memory management, output, performance monitoring, BLAS and LAPACK wrappers etc.

For the full list of Trilinos packages, descriptions of their capabilities, and user manuals see <http://trilinos.sandia.gov>. 

Installed version

Currently, Trilinos in version 11.2.3 compiled with Intel Compiler is installed on ANSELM.

Compiling against Trilinos

First, load the appropriate module:

```
$ module load trilinos
```

For the compilation of CMake-aware project, Trilinos provides the `FIND_PACKAGE(Trilinos)` capability, which makes it easy to build against Trilinos, including linking against the correct list of libraries. For details, see http://trilinos.sandia.gov/Finding_Trilinos.txt 

For compiling using simple makefiles, Trilinos provides `Makefile.export` system, which allows users to include important Trilinos variables directly into their makefiles. This can be done simply by inserting the following line into the makefile:

```
include Makefile.export.Trilinos
```

or

include Makefile.export.<package>

if you are interested only in a specific Trilinos package. This will give you access to the variables such as Trilinos_CXX_COMPILER, Trilinos_INCLUDE_DIRS, Trilinos_LIBRARY_DIRS etc. For the detailed description and example makefile see http://trilinos.sandia.gov/Export_Makefile.txt.

GSL

The GNU Scientific Library. Provides a wide range of mathematical routines.

Introduction

The GNU Scientific Library (GSL) provides a wide range of mathematical routines such as random number generators, special functions and least-squares fitting. There are over 1000 functions in total. The routines have been written from scratch in C, and present a modern Applications Programming Interface (API) for C programmers, allowing wrappers to be written for very high level languages.

The library covers a wide range of topics in numerical computing. Routines are available for the following areas:

Complex Numbers	Roots of Polynomials
Special Functions	Vectors and Matrices
Permutations	Combinations
Sorting	BLAS Support
Linear Algebra	CBLAS Library
Fast Fourier Transforms	Eigensystems
Random Numbers	Quadrature
Random Distributions	Quasi-Random Sequences
Histograms	Statistics
Monte Carlo Integration	N-Tuples
Differential Equations	Simulated Annealing
Numerical Differentiation	Interpolation
Series Acceleration	Chebyshev Approximations
Root-Finding	Discrete Hankel Transforms
Least-Squares Fitting	Minimization
IEEE Floating-Point	Physical Constants
Basis Splines	Wavelets

Modules

The GSL 1.16 is available on Anselm, compiled for GNU and Intel compiler. These variants are available via modules:

Module	Compiler
gsl/1.16-gcc	gcc 4.8.6
gsl/1.16-icc(default)	icc

```
$ module load gsl
```

The module sets up environment variables, required for linking and running GSL enabled applications. This particular command loads the default module, which is gsl/1.16-icc

Linking

Load an appropriate gsl module. Link using **-lgsl** switch to link your code against GSL. The GSL depends on cblas API to BLAS library, which must be supplied for linking. The BLAS may be provided, for example from the MKL library, as well as from the BLAS GSL library (-lgslcblas). Using the MKL is recommended.

Compiling and linking with Intel compilers

```
$ module load intel
$ module load gsl
$ icc myprog.c -o myprog.x -Wl,-rpath=$LIBRARY_PATH -mkl -lgsl
```

Compiling and linking with GNU compilers

```
$ module load gcc
$ module load mkl
$ module load gsl/1.16-gcc
$ gcc myprog.c -o myprog.x -Wl,-rpath=$LIBRARY_PATH -lmkl_intel_lp64 -lmkl_gnu_thread
```

Example

Following is an example of discrete wavelet transform implemented by GSL:

```
#include <stdio.h>
#include <math.h>
#include <gsl/gsl_sort.h>
#include <gsl/gsl_wavelet.h>

int
main (int argc, char **argv)
{
    int i, n = 256, nc = 20;
    double *data = malloc (n * sizeof (double));
    double *abscoeff = malloc (n * sizeof (double));
    size_t *p = malloc (n * sizeof (size_t));
```

```

gsl_wavelet *w;
gsl_wavelet_workspace *work;

w = gsl_wavelet_alloc (gsl_wavelet_daubechies, 4);
work = gsl_wavelet_workspace_alloc (n);

for (i=0; i<n; i++)
data[i] = sin (3.141592654*(double)i/256.0);

gsl_wavelet_transform_forward (w, data, 1, n, work);

for (i = 0; i < n; i++)
{
    abscoeff[i] = fabs (data[i]);
}

gsl_sort_index (p, abscoeff, 1, n);

for (i = 0; (i + nc) < n; i++)
    data[p[i]] = 0;

gsl_wavelet_transform_inverse (w, data, 1, n, work);

for (i = 0; i < n; i++)
{
    printf ("%gn", data[i]);
}

gsl_wavelet_free (w);
gsl_wavelet_workspace_free (work);

free (data);
free (abscoeff);
free (p);
return 0;
}

```

Load modules and compile:

```

$ module load intel gsl
icc dwt.c -o dwt.x -Wl,-rpath=$LIBRARY_PATH -mkl -lgsl

```

In this example, we compile the dwt.c code using the Intel compiler and link it to the MKL and GSL library, note the -mkl and -lgsl options. The library search path is compiled in, so that no modules are necessary to run the code.

MPI

Setting up MPI Environment

The Anselm cluster provides several implementations of the MPI library:

MPI	Thread
Li-	sup-
brary	port

The	Partial
highly	thread
opti-	sup-
mized	port
and	up
sta-	to
ble	MPI_THREAD_SERIALIZED

bul-
lxmpi 1.2.4.1

The	Full
In-	thread
tel	sup-
MPI	port
4.1	up
	to
	MPI_THREAD_MULTIPLE

The	Full
Open-	thread
MPI	sup-
1.6.5	port
	up
	to
	MPI_THREAD_MULTIPLE,
	BLCR
	c/r
	sup-
	port

The	Full
Open-	thread
MPI	sup-
1.8.1	port
	up
	to
	MPI_THREAD_MULTIPLE,
	MPI-
	3.0
	sup-
	port

MPI Library	Thread support
mpich2	Full support up to MPI_THREAD_MULTIPLE, BLCR c/r support

MPI libraries are activated via the environment modules.

Look up section modulefiles/mpi in module avail

```
$ module avail
----- /opt/modules/modulefiles/mpi -----
bullxmpi/bullxmpi-1.2.4.1  mvapich2/1.9-icc
impi/4.0.3.008             openmpi/1.6.5-gcc(default)
impi/4.1.0.024             openmpi/1.6.5-gcc46
impi/4.1.0.030             openmpi/1.6.5-icc
impi/4.1.1.036(default)    openmpi/1.8.1-gcc
openmpi/1.8.1-gcc46
mvapich2/1.9-gcc(default)  openmpi/1.8.1-gcc49
mvapich2/1.9-gcc46         openmpi/1.8.1-icc
```

There are default compilers associated with any particular MPI implementation. The defaults may be changed, the MPI libraries may be used in conjunction with any compiler. The defaults are selected via the modules in following way

Module	MPI	Compiler suite
PrgEnv-gnu	bullxmpi-1.2.4.1	bullxmpi-4.4.6 GNU
PrgEnv-intel	Intel MPI 4.1.1	Intel 13.1.1

Module	Compiler suite	
	MPI	
bullxmpi	bullxmpi 1.2.4.1	none, se- lect via mod- ule
impi	Intel MPI 4.1.1	none, se- lect via mod- ule
openmpi	OpenMPI 1.6.5	GNU com- pil- ers 4.8.1, GNU com- pil- ers 4.4.6, Intel Com- pil- ers
openmpi	OpenMPI 1.8.1	GNU com- pil- ers 4.8.1, GNU com- pil- ers 4.4.6, GNU com- pil- ers 4.9.0, Intel Com- pil- ers

	MPI	Compiler suite
Module		
mvapich2	MPICH2	GNU
	1.9	com- pil- ers 4.8.1, GNU com- pil- ers 4.4.6, Intel Com- pil- ers

Examples:

```
$ module load openmpi
```

In this example, we activate the latest openmpi with latest GNU compilers

To use openmpi with the intel compiler suite, use

```
$ module load intel
$ module load openmpi/1.6.5-icc
```

In this example, the openmpi 1.6.5 using intel compilers is activated

Compiling MPI Programs

!!! Note “Note” After setting up your MPI environment, compile your program using one of the mpi wrappers

```
$ mpicc -v
$ mpif77 -v
$ mpif90 -v
```

Example program:

```
// helloworld_mpi.c
#include <stdio.h>

#include<mpi.h>

int main(int argc, char **argv) {

    int len;
    int rank, size;
    char node[MPI_MAX_PROCESSOR_NAME];
```

```

// Initiate MPI
MPI_Init(&argc, &argv);
MPI_Comm_rank(MPI_COMM_WORLD,&rank);
MPI_Comm_size(MPI_COMM_WORLD,&size);

// Get hostname and print
MPI_Get_processor_name(node,&len);
printf("Hello world! from rank %d of %d on host %sn",rank,size,node);

// Finalize and exit
MPI_Finalize();

return 0;
}

```

Compile the above example with

```
$ mpicc helloworld_mpi.c -o helloworld_mpi.x
```

Running MPI Programs

!!! Note “Note” The MPI program executable must be compatible with the loaded MPI module. Always compile and execute using the very same MPI module.

It is strongly discouraged to mix mpi implementations. Linking an application with one MPI implementation and running mpirun/mpiexec from other implementation may result in unexpected errors.

The MPI program executable must be available within the same path on all nodes. This is automatically fulfilled on the /home and /scratch filesystem. You need to preload the executable, if running on the local scratch /lscratch filesystem.

Ways to run MPI programs

Optimal way to run an MPI program depends on its memory requirements, memory access pattern and communication pattern.

!!! Note “Note” Consider these ways to run an MPI program:

1. One MPI process per node, 16 threads per process
2. Two MPI processes per node, 8 threads per process
3. 16 MPI processes per node, 1 thread per process.

One MPI process per node, using 16 threads, is most useful for memory demanding applications, that make good use of processor cache memory and are not memory bound. This is also a preferred way for communication intensive applications as one process per node enjoys full bandwidth access to the network interface.


Two MPI processes per node, using 8 threads each, bound to processor socket is most useful for memory bandwidth bound applications such as BLAS1 or FFT, with scalable memory demand. However, note that the two processes will share access to the network interface. The 8 threads and socket binding should ensure maximum memory access bandwidth and minimize communication, migration and numa effect overheads.

!!! Note “Note” Important! Bind every OpenMP thread to a core!

In the previous two cases with one or two MPI processes per node, the operating system might still migrate OpenMP threads between cores. You want to avoid this by setting the KMP_AFFINITY or GOMP_CPU_AFFINITY environment variables.

16 MPI processes per node, using 1 thread each bound to processor core is most suitable for highly scalable applications with low communication demand.

Running OpenMPI

The **bullxmpi-1.2.4.1** and **OpenMPI 1.6.5**  are both based on OpenMPI. Read more on how to run OpenMPI based MPI.

Running MPICH2

The **Intel MPI** and **mpich2 1.9** are MPICH2 based implementations. Read more on how to run MPICH2 based MPI.

The Intel MPI may run on the Intel Xeon Phi accelerators as well. Read more on how to run Intel MPI on accelerators.

Running MPICH2

MPICH2 program execution

The MPICH2 programs use mpd daemon or ssh connection to spawn processes, no PBS support is needed. However the PBS allocation is required to access compute nodes. On Anselm, the **Intel MPI** and **mpich2 1.9** are MPICH2 based MPI implementations.

Basic usage

!!! Note “Note” Use the mpirun to execute the MPICH2 code.

Example:

```
$ qsub -q qexp -l select=4:ncpus=16 -I
qsub: waiting for job 15210.srv11 to start
qsub: job 15210.srv11 ready

$ module load impi

$ mpirun -ppn 1 -hostfile $PBS_NODEFILE ./helloworld_mpi.x
Hello world! from rank 0 of 4 on host cn17
Hello world! from rank 1 of 4 on host cn108
Hello world! from rank 2 of 4 on host cn109
Hello world! from rank 3 of 4 on host cn110
```

In this example, we allocate 4 nodes via the express queue interactively. We set up the intel MPI environment and interactively run the helloworld_mpi.x program. We request MPI to spawn 1 process per node. Note that the executable helloworld_mpi.x must be available within the same path on all nodes. This is automatically fulfilled on the /home and /scratch filesystem.

You need to preload the executable, if running on the local scratch /lscratch filesystem

```
$ pwd
/lscratch/15210.srv11
$ mpirun -ppn 1 -hostfile $PBS_NODEFILE cp /home/username/helloworld_mpi.x .
$ mpirun -ppn 1 -hostfile $PBS_NODEFILE ./helloworld_mpi.x
Hello world! from rank 0 of 4 on host cn17
Hello world! from rank 1 of 4 on host cn108
Hello world! from rank 2 of 4 on host cn109
Hello world! from rank 3 of 4 on host cn110
```

In this example, we assume the executable helloworld_mpi.x is present on shared home directory. We run the cp command via mpirun, copying the executable from shared home to local scratch. Second mpirun will execute the binary in the /lscratch/15210.srv11 directory on nodes cn17, cn108, cn109 and cn110, one process per node.

!!! Note “Note” MPI process mapping may be controlled by PBS parameters.

The mpirprocs and ompthreads parameters allow for selection of number of running MPI processes per node as well as number of OpenMP threads per MPI process.

One MPI process per node

Follow this example to run one MPI process per node, 16 threads per process. Note that no options to mpirun are needed

```
$ qsub -q qexp -l select=4:ncpus=16:mpiprocs=1:ompthreads=16 -I  
  
$ module load mvapich2  
  
$ mpirun ./helloworld_mpi.x
```

In this example, we demonstrate recommended way to run an MPI application, using 1 MPI processes per node and 16 threads per socket, on 4 nodes.

Two MPI processes per node

Follow this example to run two MPI processes per node, 8 threads per process. Note the options to mpirun for mvapich2. No options are needed for impi.

```
$ qsub -q qexp -l select=4:ncpus=16:mpiprocs=2:ompthreads=8 -I  
  
$ module load mvapich2  
  
$ mpirun -bind-to numa ./helloworld_mpi.x
```

In this example, we demonstrate recommended way to run an MPI application, using 2 MPI processes per node and 8 threads per socket, each process and its threads bound to a separate processor socket of the node, on 4 nodes

16 MPI processes per node

Follow this example to run 16 MPI processes per node, 1 thread per process. Note the options to mpirun for mvapich2. No options are needed for impi.

```
$ qsub -q qexp -l select=4:ncpus=16:mpiprocs=16:ompthreads=1 -I  
  
$ module load mvapich2  
  
$ mpirun -bind-to core ./helloworld_mpi.x
```

In this example, we demonstrate recommended way to run an MPI application, using 16 MPI processes per node, single threaded. Each process is bound to separate processor core, on 4 nodes.

OpenMP thread affinity

!!! Note “Note” Important! Bind every OpenMP thread to a core!

In the previous two examples with one or two MPI processes per node, the operating system might still migrate OpenMP threads between cores. You might want to avoid this by setting these environment variable for GCC OpenMP:

```
$ export GOMP_CPU_AFFINITY="0-15"
```

or this one for Intel OpenMP:

```
$ export KMP_AFFINITY=granularity=fine,compact,1,0
```

As of OpenMP 4.0 (supported by GCC 4.9 and later and Intel 14.0 and later) the following variables may be used for Intel or GCC:

```
$ export OMP_PROC_BIND=true
$ export OMP_PLACES=cores
```

MPICH2 Process Mapping and Binding

The mpirun allows for precise selection of how the MPI processes will be mapped to the computational nodes and how these processes will bind to particular processor sockets and cores.

Machinefile

Process mapping may be controlled by specifying a machinefile input to the mpirun program. Although all implementations of MPI provide means for process mapping and binding, following examples are valid for the impi and mvapich2 only.

Example machinefile

```
cn110.bullx
cn109.bullx
cn108.bullx
cn17.bullx
cn108.bullx
```

Use the machinefile to control process placement

```
$ mpirun -machinefile machinefile helloworld_mpi.x
Hello world! from rank 0 of 5 on host cn110
Hello world! from rank 1 of 5 on host cn109
Hello world! from rank 2 of 5 on host cn108
Hello world! from rank 3 of 5 on host cn17
Hello world! from rank 4 of 5 on host cn108
```

In this example, we see that ranks have been mapped on nodes according to the order in which nodes show in the machinefile

Process Binding

The Intel MPI automatically binds each process and its threads to the corresponding portion of cores on the processor socket of the node, no options needed. The binding is primarily controlled by environment variables. Read more about mpi process binding on Intel website [\[4\]](#). The MPICH2 uses the -bind-to option Use -bind-to numa or -bind-to core to bind the process on single core or entire socket.

Bindings verification

In all cases, binding and threading may be verified by executing

```
$ mpirun -bindto numa numactl --show
$ mpirun -bindto numa echo $OMP_NUM_THREADS
```

Intel MPI on Xeon Phi

The MPI section of Intel Xeon Phi chapter provides details on how to run Intel MPI code on Xeon Phi architecture.

Running OpenMPI

OpenMPI program execution

The OpenMPI programs may be executed only via the PBS Workload manager, by entering an appropriate queue. On Anselm, the **bullxmpi-1.2.4.1** and **OpenMPI 1.6.5** are OpenMPI based MPI implementations.

Basic usage

!!! Note “Note” Use the mpiexec to run the OpenMPI code.

Example:

```
$ qsub -q qexp -l select=4:ncpus=16 -I
qsub: waiting for job 15210.srv11 to start
qsub: job 15210.srv11 ready

$ pwd
/home/username

$ module load openmpi
$ mpiexec -pernode ./helloworld_mpi.x
Hello world! from rank 0 of 4 on host cn17
Hello world! from rank 1 of 4 on host cn108
Hello world! from rank 2 of 4 on host cn109
Hello world! from rank 3 of 4 on host cn110
```

!!! Note “Note” Please be aware, that in this example, the directive **-pernode** is used to run only **one task per node**, which is normally an unwanted behaviour (unless you want to run hybrid code with just one MPI and 16 OpenMP tasks per node). In normal MPI programs **omit the -pernode directive** to run up to 16 MPI tasks per each node.

In this example, we allocate 4 nodes via the express queue interactively. We set up the openmpi environment and interactively run the helloworld_mpi.x program. Note that the executable helloworld_mpi.x must be available within the same path on all nodes. This is automatically fulfilled on the /home and /scratch filesystem.

You need to preload the executable, if running on the local scratch /lscratch filesystem

```
$ pwd
/lscratch/15210.srv11

$ mpiexec -pernode --preload-binary ./helloworld_mpi.x
Hello world! from rank 0 of 4 on host cn17
Hello world! from rank 1 of 4 on host cn108
Hello world! from rank 2 of 4 on host cn109
Hello world! from rank 3 of 4 on host cn110
```

In this example, we assume the executable helloworld_mpi.x is present on compute node cn17 on local scratch. We call the mpiexec with the **--preload-binary** argument (valid for openmpi). The mpiexec will copy the executable from cn17 to the /lscratch/15210.srv11 directory on cn108, cn109 and cn110 and execute the program.

!!! Note “Note” MPI process mapping may be controlled by PBS parameters.

The `mpiprocs` and `ompthreads` parameters allow for selection of number of running MPI processes per node as well as number of OpenMP threads per MPI process.

One MPI process per node

Follow this example to run one MPI process per node, 16 threads per process.

```
$ qsub -q qexp -l select=4:ncpus=16:mpiprocs=1:ompthreads=16 -I  
  
$ module load openmpi  
  
$ mpiexec --bind-to-none ./helloworld_mpi.x
```

In this example, we demonstrate recommended way to run an MPI application, using 1 MPI processes per node and 16 threads per socket, on 4 nodes.

Two MPI processes per node

Follow this example to run two MPI processes per node, 8 threads per process. Note the options to `mpiexec`.

```
$ qsub -q qexp -l select=4:ncpus=16:mpiprocs=2:ompthreads=8 -I  
  
$ module load openmpi  
  
$ mpiexec -bysocket -bind-to-socket ./helloworld_mpi.x
```

In this example, we demonstrate recommended way to run an MPI application, using 2 MPI processes per node and 8 threads per socket, each process and its threads bound to a separate processor socket of the node, on 4 nodes

16 MPI processes per node

Follow this example to run 16 MPI processes per node, 1 thread per process. Note the options to `mpiexec`.

```
$ qsub -q qexp -l select=4:ncpus=16:mpiprocs=16:ompthreads=1 -I  
  
$ module load openmpi  
  
$ mpiexec -bycore -bind-to-core ./helloworld_mpi.x
```

In this example, we demonstrate recommended way to run an MPI application, using 16 MPI processes per node, single threaded. Each process is bound to separate processor core, on 4 nodes.

OpenMP thread affinity

!!! Note “Note” Important! Bind every OpenMP thread to a core!

In the previous two examples with one or two MPI processes per node, the operating system might still migrate OpenMP threads between cores. You might want to avoid this by setting these environment variable for GCC OpenMP:

```
$ export GOMP_CPU_AFFINITY="0-15"
```

or this one for Intel OpenMP:

```
'bash      $ export KMP_AFFINITY=granularity=fine,compact,1,0
```

As of OpenMP 4.0 (supported by GCC 4.9 and later and Intel 14.0 and later) the following variables may be used for Intel or GCC:

```
$ export OMP_PROC_BIND=true
$ export OMP_PLACES=cores
```

OpenMPI Process Mapping and Binding

The `mpiexec` allows for precise selection of how the MPI processes will be mapped to the computational nodes and how these processes will bind to particular processor sockets and cores.

MPI process mapping may be specified by a hostfile or rankfile input to the `mpiexec` program. Although all implementations of MPI provide means for process mapping and binding, following examples are valid for the openmpi only.

Hostfile

Example hostfile

```
cn110.bullx
cn109.bullx
cn108.bullx
cn17.bullx
```

Use the hostfile to control process placement

```
$ mpiexec -hostfile hostfile ./helloworld_mpi.x
Hello world! from rank 0 of 4 on host cn110
Hello world! from rank 1 of 4 on host cn109
Hello world! from rank 2 of 4 on host cn108
Hello world! from rank 3 of 4 on host cn17
```

In this example, we see that ranks have been mapped on nodes according to the order in which nodes show in the hostfile

Rankfile

Exact control of MPI process placement and resource binding is provided by specifying a rankfile

!!! Note “Note” Appropriate binding may boost performance of your application.

Example rankfile

```
rank 0=cn110.bullx slot=1:0,1
rank 1=cn109.bullx slot=0:*
rank 2=cn108.bullx slot=1:1-2
rank 3=cn17.bullx slot=0:1,1:0-2
rank 4=cn109.bullx slot=0:*,1:*
```

This rankfile assumes 5 ranks will be running on 4 nodes and provides exact mapping and binding of the processes to the processor sockets and cores

Explanation: rank 0 will be bounded to cn110, socket1 core0 and core1 rank 1 will be bounded to cn109, socket0, all cores rank 2 will be bounded to cn108, socket1, core1 and core2 rank 3 will be bounded to cn17, socket0 core1, socket1 core0, core1, core2 rank 4 will be bounded to cn109, all cores on both sockets

```
$ mpiexec -n 5 -rf rankfile --report-bindings ./helloworld_mpi.x
[cn17:11180] MCW rank 3 bound to socket 0[core 1] socket 1[core 0-2]: [. B . . . . .]
[cn110:09928] MCW rank 0 bound to socket 1[core 0-1]: [. . . . .] [B B . . . . .]
[cn109:10395] MCW rank 1 bound to socket 0[core 0-7]: [B B B B B B B B] [. . . . .]
[cn108:10406] MCW rank 2 bound to socket 1[core 1-2]: [. . . . .] [. B B . . . . .]
[cn109:10406] MCW rank 4 bound to socket 0[core 0-7] socket 1[core 0-7]: [B B B B B B B B]
Hello world! from rank 3 of 5 on host cn17
Hello world! from rank 1 of 5 on host cn109
Hello world! from rank 0 of 5 on host cn110
Hello world! from rank 4 of 5 on host cn109
Hello world! from rank 2 of 5 on host cn108
```

In this example we run 5 MPI processes (5 ranks) on four nodes. The rankfile defines how the processes will be mapped on the nodes, sockets and cores. The **—report-bindings** option was used to print out the actual process location and bindings. Note that ranks 1 and 4 run on the same node and their core binding overlaps.

It is users responsibility to provide correct number of ranks, sockets and cores.

Bindings verification

In all cases, binding and threading may be verified by executing for example:

```
$ mpiexec -bysocket -bind-to-socket --report-bindings echo
$ mpiexec -bysocket -bind-to-socket numactl --show
$ mpiexec -bysocket -bind-to-socket echo $OMP_NUM_THREADS
```

Changes in OpenMPI 1.8

Some options have changed in OpenMPI version 1.8.

version 1.6.5	version 1.8.1
—bind-to-none	—bind-to none
—bind-to-core	—bind-to core
—bind-to-socket	—bind-to socket
-bysocket	—map-by socket
-bycore	—map-by core
-pernode	—map-by ppr:1:node

MPI4Py (MPI for Python)

Introduction

MPI for Python provides bindings of the Message Passing Interface (MPI) standard for the Python programming language, allowing any Python program to exploit multiple processors.

This package is constructed on top of the MPI-1/2 specifications and provides an object oriented interface which closely follows MPI-2 C++ bindings. It supports point-to-point (sends, receives) and collective (broadcasts, scatters, gathers) communications of any picklable Python object, as well as optimized communications of Python object exposing the single-segment buffer interface (NumPy arrays, builtin bytes/string/array objects).

On Anselm MPI4Py is available in standard Python modules.

Modules

MPI4Py is build for OpenMPI. Before you start with MPI4Py you need to load Python and OpenMPI modules.

```
$ module load python
$ module load openmpi
```

Execution

You need to import MPI to your python program. Include the following line to the python script:

```
from mpi4py import MPI
```

The MPI4Py enabled python programs execute as any other OpenMPI code. The simplest way is to run

```
$ mpiexec python <script>.py
```

For example

```
$ mpiexec python hello_world.py
```

Examples

Hello world!

```
from mpi4py import MPI

comm = MPI.COMM_WORLD

print "Hello! I'm rank %d from %d running in total..." % (comm.rank, comm.size)

comm.Barrier()  # wait for everybody to synchronize
```

Collective Communication with NumPy arrays

```
from mpi4py import MPI
from __future__ import division
import numpy as np

comm = MPI.COMM_WORLD

print("-"*78)
print(" Running on %d cores" % comm.size)
print("-"*78)

comm.Barrier()

# Prepare a vector of N=5 elements to be broadcasted...
N = 5
if comm.rank == 0:
    A = np.arange(N, dtype=np.float64)    # rank 0 has proper data
else:
    A = np.empty(N, dtype=np.float64)    # all other just an empty array

# Broadcast A from rank 0 to everybody
comm.Bcast( [A, MPI.DOUBLE] )

# Everybody should now have the same...
print "[%02d] %s" % (comm.rank, A)
```

Execute the above code as:

```
$ qsub -q qexp -l select=4:ncpus=16:mpiprocs=16:ompthreads=1 -I

$ module load python openmpi

$ mpiexec -bycore -bind-to-core python hello_world.py
```

In this example, we run MPI4Py enabled code on 4 nodes, 16 cores per node (total of 64 processes), each python process is bound to a different core. More examples and documentation can be found on MPI for Python webpage [here](#).

Package ‘parallel’

R-core

May 16, 2013

1 Introduction

Package **parallel** was first included in R 2.14.0. It builds on the work done for CRAN packages **multicore** (Urbanek, 2009–present) and **snow** (Tierney *et al.*, 2003–present) and provides drop-in replacements for most of the functionality of those packages, with integrated handling of random-number generation.

Parallelism can be done in computation at many different levels: this package is principally concerned with ‘coarse-grained parallelization’. At the lowest level, modern CPUs can do several basic operations simultaneously (e.g. integer and floating-point arithmetic), and several implementations of external BLAS libraries use multiple threads to do parts of basic vector/matrix operations in parallel. Several contributed R packages use multiple threads at C level *via* OpenMP or pthreads.

This package handles running much larger chunks of computations in parallel. A typical example is to evaluate the same R function on many different sets of data: often simulated data as in bootstrap computations (or with ‘data’ being the random-number stream). The crucial point is that these chunks of computation are unrelated and do not need to communicate in any way. It is often the case that the chunks take approximately the same length of time. The basic computational model is

- (a) Start up M ‘worker’ processes, and do any initialization needed on the workers.
- (b) Send any data required for each task to the workers.
- (c) Split the task into M roughly equally-sized chunks, and send the chunks (including the R code needed) to the workers.
- (d) Wait for all the workers to complete their tasks, and ask them for their results.
- (e) Repeat steps (b–d) for any further tasks.
- (f) Shut down the worker processes.

Amongst the initializations which may be needed are to load packages and initialize the random-number stream.

There are implementations of this model in the functions `mclapply` and `parLapply` as near-drop-in replacements for `lapply`.

A slightly different model is to split the task into $M_1 > M$ chunks, send the first M chunks to the workers, then repeatedly wait for any worker to complete and send it the next remaining task: see the section on ‘load balancing’.

In principle the workers could be implemented by threads¹ or lightweight processes, but in the

¹only ‘in principle’ since the R interpreter is not thread-safe.

current implementation they are full processes. They can be created in one of three ways:

1. *Via* `system("Rscript")` or similar to launch a new process on the current machine or a similar machine with an identical R installation. This then needs a way to communicate between master and worker processes, which is usually done *via* sockets.

This should be available on all R platforms, although it is conceivable that zealous security measures could block the inter-process communication *via* sockets. Users of Windows and OS X may expect pop-up dialog boxes from the firewall asking if an R process should accept incoming connections.

Following **snow**, a pool of worker processes listening *via* sockets for commands from the master is called a ‘cluster’ of nodes.

2. *Via* forking. *Fork* is a concept² from POSIX operating systems, and should be available on all R platforms except Windows. This creates a new R process by taking a complete copy of the master process, including the workspace and state of the random-number stream. However, the copy will (in any reasonable OS) share memory pages with the master until modified so forking is very fast.

The use of forking was pioneered by package **multicore**.

Note that as it does share the complete process, it also shares any GUI elements, for example an R console and on-screen devices. This can cause havoc.³

There needs to be a way to communicate between master and worker. Once again there are several possibilities since master and workers share memory. In **multicore** the initial fork sends an R expression to be evaluated to the worker, and the master process opens a pipe for reading that is used by the worker to return the results. Both that and creating a cluster of nodes communicating *via* sockets are supported in package **parallel**.

3. Using OS-level facilities to set up a means to send tasks to other members of a group of machines. There are several ways to do that, and for example package **snow** can make use of MPI (‘message passing interface’) using R package **Rmpi**. Communication overheads can dominate computation times in this approach, so it is most often used on tightly-coupled networks of computers with high-speed interconnects.

CRAN packages following this approach include **GridR** (using Condor or Globus) and **Rsgs** (using SGE, currently called ‘Oracle Grid Engine’).

It will not be considered further in this vignette, but those parts of **parallel** which provide **snow**-like functions will accept **snow** clusters including MPI clusters.

The landscape of parallel computing has changed with the advent of shared-memory computers with multiple (and often many) CPU cores. Until the late 2000’s parallel computing was mainly done on clusters of large numbers of single- or dual-CPU computers: nowadays even laptops have two or four cores, and servers with 8 or more cores are commonplace. It is such hardware that package **parallel** is designed to exploit. It can also be used with several computers running the same version of R connected by (reasonable-speed) ethernet: the computers need not be running the same OS.

Note that all these methods of communication use `serialize/unserialize` to send R objects between processes. This has limits (typically hundreds of millions of elements) which a well-designed parallelized algorithm should not approach.

²[http://en.wikipedia.org/wiki/Fork_\(operating_system\)](http://en.wikipedia.org/wiki/Fork_(operating_system))

³Some precautions are taken on Mac OS X: for example the event loops for **R.app** and the **quartz** device are inhibited in the child. This information is available at C level in the **Rboolean** variable `R_isForkedChild`.

2 Numbers of CPUs/cores

In setting up parallel computations it can be helpful to have some idea of the number of CPUs or cores available, but this is a rather slippery concept. Nowadays almost all physical CPUs contain two or more cores that run more-or-less independently (they may share parts of the cache memory, and they do share access to RAM). However, on some processors these cores may themselves be able to run multiple tasks simultaneously, and some OSes (e.g. Windows) have the concept of *logical* CPUs which may exceed the number of cores.

Note that all a program can possibly determine is the total number of CPUs and/or cores available. This is not necessarily the same as the number of CPUs available *to the current user* which may well be restricted by system policies on multi-user systems. Nor does it give much idea of a reasonable number of CPUs to use for the current task: the user may be running many R processes simultaneously, and those processes may themselves be using multiple threads through a multi-threaded BLAS, compiled code using OpenMP or other low-level forms of parallelism. We have even seen instances of **multicore**'s `mclapply` being called recursively,⁴ generating $2n+n^2$ processes on a machine estimated to have $n = 16$ cores.

But in so far as it is a useful guideline, function `detectCores()` tries to determine the number of CPU cores in the machine on which R is running: it has ways to do so on all known current R platforms. What exactly it measures is OS-specific: we try where possible to report the number of physical cores available.

On Windows the default is to report the number of logical CPUs. On modern hardware (e.g. Intel *Core i7*) the latter may not be unreasonable as hyper-threading does give a significant extra throughput. What `detectCores(logical = FALSE)` reports is OS-version-dependent: on recent versions of Windows it reports the number of physical cores but on older versions it might report the number of physical CPU packages.

3 Analogues of apply functions

By far the most common direct applications of packages **multicore** and **snow** have been to provide parallelized replacements of `lapply`, `sapply`, `apply` and related functions.

As analogues of `lapply` there are

```
parLapply(cl, x, FUN, ...)  
mclapply(X, FUN, ..., mc.cores)
```

where `mclapply` is not available⁵ on Windows and has further arguments discussed on its help page. They differ slightly in philosophy: `mclapply` sets up a pool of `mc.cores` workers just for this computation, whereas `parLapply` uses a less ephemeral pool specified by the object `cl` created by a call to `makeCluster` (which *inter alia* specifies the size of the pool). So the workflow is

```
cl <- makeCluster(<size of pool>)  
# one or more parLapply calls  
stopCluster(cl)
```

For matrices there are the rarely used `parApply` and `parCapply` functions, and the more commonly used `parRapply`, a parallel row `apply` for a matrix.

⁴`parallel`: `mclapply` detects this and runs nested calls serially.

⁵except as a stub which simply calls `lapply`.

4 SNOW Clusters

The package contains a slightly revised copy of much of **snow**, and the functions it contains can also be used with clusters created by **snow** (provided the package is on the search path).

Two functions are provided to create SNOW clusters, **makePSOCKcluster** (a streamlined version of **snow::makeSOCKcluster**) and (except on Windows) **makeForkCluster**. They differ only in the way they spawn worker processes: **makePSOCKcluster** uses **Rscript** to launch further copies of R (on the same host or optionally elsewhere) whereas **makeForkCluster** forks the workers on the current host (which thus inherit the environment of the current session).

These functions would normally be called *via* **makeCluster**.

Both **stdout()** and **stderr()** of the workers are redirected, by default being discarded but they can be logged using the **outfile** option. Note that the previous sentence refers to the *connections* of those names, not the C-level file handles. Thus properly written R packages using **Rprintf** will have their output redirected, but not direct C-level output.

A default cluster can be registered by a call to **setDefaultCluster()**: this is then used whenever one of the higher-level functions such as **parApply** is called without an explicit cluster. A little care is needed when repeatedly re-using a pool of workers, as their workspaces will accumulate objects from past usage, and packages may get added to the search path.

If clusters are to be created on a host other than the current machine ('localhost'), **makeCluster** may need to be given more information in the shape of extra arguments.

- If the worker machines are not set up in exactly the same way as the master (for example if they are of a different architecture), use **homogeneous = FALSE** and perhaps set **rscript** to the full path to **Rscript** on the workers.
- The worker machines need to know how to communicate with the master: normally this can be done using the hostname found by **Sys.info()**, but on private networks this need not be the case and **master** may need to be supplied as a name or IP address, e.g. **master = "192.168.1.111"**.
- By default **ssh** is used to launch R on the workers. If it is known by some other name, use e.g. **rshcmd = "plink.exe"** for a Windows box using PUTTY. SSH should be set up to use silent authentication: setups which require a password to be supplied may or may not work.
- Socket communication is done over port a randomly chosen port in the range 11000:11999: site policies might require some other port to be used, in which case set argument **port** or environment variable **R_PARALLEL_PORT**.

5 Forking

Except on Windows, the package contains a copy of **multicore**: there a few names with the added prefix **mc**, e.g. **mccollect** and **mcparallel**. (Package **multicore** used these names, but also the versions without the prefix which are too easily masked: e.g. package **lattice** has a function **parallel**.)

The low-level functions from **multicore** are provided but not exported from the namespace.

There are high-level functions **mcapply** and **pvec**: unlike the versions in **multicore** these default to 2 cores, but this can be controlled by setting **options("mc.cores")**, and that takes its default from environment variable **MC_CORES** when the package is loaded. (Setting this to 1

inhibits parallel operation: there are stub versions of these functions on Windows which force `mc.cores = 1.`)

As from R 2.15.0, `mcmapply` and `mcMap` provide analogues of `mapply` and `Map`.

Note the earlier comments about using forking in a GUI environment.

The parent and forked R processes share the per-session temporary directory `tempdir()`, which can be a problem as a lot of code has assumed it is private to the R process. Further, prior to R 2.14.1 it was possible for `tempfile` in two processes to select the same filename in that temporary directory, and do it sufficiently simultaneously that neither saw it as being in use.

The forked workers share file handles with the master: this means that any output from the worker should go to the same place as `'stdout'` and `'stderr'` of the master process. (This does not work reliably on all OSes: problems have also been noted when forking a session that is processing batch input from `'stdin'`.) Setting argument `mc.silent = TRUE` silences `'stdout'` for the child: `'stderr'` is not affected.

Sharing file handles also impacts graphics devices as forked workers inherit all open graphics devices of the parent: they should not attempt to make use of them.

6 Random-number generation

Some care is needed with parallel computation using (pseudo-)random numbers: the processes/threads which run separate parts of the computation need to run independent (and preferably reproducible) random-number streams. One way to avoid any difficulties is (where possible) to do all the randomization in the master process: this is done where possible in package **boot** (version 1.3-1 and later).

When an R process is started up it takes the random-number seed from the object `.Random.seed` in a saved workspace or constructs one from the clock time and process ID when random-number generation is first used (see the help on `RNG`). Thus worker processes might get the same seed because a workspace containing `.Random.seed` was restored or the random number generator has been used before forking: otherwise these get a non-reproducible seed (but with very high probability a different seed for each worker).

The alternative is to set separate seeds for each worker process in some reproducible way from the seed in the master process. This is generally plenty safe enough, but there have been worries that the random-number streams in the workers might somehow get into step. One approach is to take the seeds a long way apart in the random-number stream: note that random numbers taken a long (fixed) distance apart in a single stream are not necessarily (and often are not) as independent as those taken a short distance apart. Yet another idea (as used by e.g. **JAGS**) is to use different random-number generators for each separate run/process.

Package **parallel** contains an implementation of the ideas of L'Ecuyer *et al.* (2002): this uses a single RNG and make *streams* with seeds 2^{127} steps apart in the random number stream (which has period approximately 2^{191}). This is based on the generator of L'Ecuyer (1999); the reason for choosing that generator⁶ is that it has a fairly long period with a small seed (6 integers), and unlike R's default "Mersenne-Twister" RNG, it is simple to advance the seed by a fixed

⁶apart from the commonality of authors!

number of steps. The generator is the combination of two:

$$\begin{aligned}x_n &= 1403580 \times x_{n-2} - 810728 \times x_{n-3} \mod (2^{32} - 209) \\y_n &= 527612 \times y_{n-1} - 1370589 \times y_{n-3} \mod (2^{32} - 22853) \\z_n &= (x_n - y_n) \mod 4294967087 \\u_n &= z_n / 4294967088 \text{ unless } z_n = 0\end{aligned}$$

The ‘seed’ then consists of $(x_n, x_{n-1}, x_{n-2}, y_n, y_{n-1}, y_{n-2})$, and the recursion for each of x_n and y_n can have pre-computed coefficients for k steps ahead. For $k = 2^{127}$, the seed is advanced by k steps by R call `.Random.seed <- nextRNGStream(.Random.seed)`.

The L’Ecuyer (1999) generator is available in R as from version 2.14.0 *via* `RNGkind("L'Ecuyer-CMRG")`. Thus using the ideas of L’Ecuyer *et al.* (2002) is as simple as

```
RNGkind("L'Ecuyer-CMRG")
set.seed(<something>)
## start M workers
s <- .Random.seed
for (i in 1:M) {
  s <- nextRNGStream(s)
  # send s to worker i as .Random.seed
}
```

and this is implemented for SNOW clusters in function `clusterSetRNGStream`, and as part of `mclapply` and `mclapply` (by default).

Apart from *streams* (2^{127} steps apart), there is the concept of *sub-streams* starting from seeds 2^{76} steps apart. Function `nextRNGSubStream` advances to the next substream.

A direct R interface to the (clunkier) original C implementation is available in CRAN package **rlecuyer** (Sevcikova and Rossini, 2004–present). That works with named streams, each of which have three 6-element seeds associated with them. This can easily be emulated in R by storing `.Random.seed` at suitable times. There is another interface using S4 classes in package **rstream** (Leydold, 2005–present).

7 Load balancing

The introduction mentioned a different strategy which dynamically allocates tasks to workers: this is sometimes known as ‘load balancing’ and is implemented in `mclapply(mc.preschedule = FALSE)`, `clusterApplyLB` and wrappers such as `parLapplyLB` and `clusterMap(.scheduling = "dynamic")`.

Load balancing is potentially advantageous when the tasks take quite dissimilar amounts of computation time, or where the nodes are of disparate capabilities. But some *caveats* are in order:

- (a) Random number streams are allocated to nodes, so if the tasks involve random numbers they are likely to be non-repeatable (as the allocation of tasks to nodes depends on the workloads of the nodes). It would however take only slightly more work to allocate a stream to each task.
- (b) More care is needed in allocating the tasks. If 1000 tasks need to be allocated to 10 nodes, the standard approach send chunks of 100 tasks to each of the nodes. The load-balancing approach sends tasks one at a time to a node, and the communication overhead may be high. So it makes sense to have substantially more tasks than nodes, but not by a factor of 100 (and maybe not by 10).

8 Portability considerations

People wanting to provide parallel facilities in their code need to decide how hard they want to try to be portable and efficient: no approach works optimally on all platforms.

Using `mclapply` is usually the simplest approach, but will run serial versions of the code on Windows. This may suffice where parallel computation is only required for use on a single multi-core Unix-alike server—for `mclapply` can only run on a single shared-memory system. There is fallback to serial use when needed, by setting `mc.cores = 1`.

Using `parLapply` will work everywhere that socket communication works, and can be used, for example, to harness all the CPU cores in a lab of machines that are not otherwise in use. But socket communication may be blocked even when using a single machine and is quite likely to be blocked between machines in a lab. There is not currently any fallback to serial use, nor could there easily be (as the workers start with a different R environment from the one current on the master).

An example of providing access to both approaches as well as serial code is package **boot**, version 1.3-3 or later.

9 Extended examples

```
> library(parallel)
```

Probably the most common use of coarse-grained parallelization in statistics is to do multiple simulation runs, for example to do large numbers of bootstrap replicates or several runs of an MCMC simulation. We show an example of each.

Note that some of the examples will only work serially on Windows and some actually are computer-intensive.

9.1 Bootstrapping

Package **boot** (Canty and Ripley, 1999–present) is support software for the monograph by Davison and Hinkley (1997). Bootstrapping is often used as an example of easy parallelization, and some methods of producing confidence intervals require many thousands of bootstrap samples. As from version 1.3-1 the package itself has parallel support within its main functions, but we illustrate how to use the original (serial) functions in parallel computations.

We consider two examples using the `cd4` dataset from package **boot** where the interest is in the correlation between before and after measurements. The first is a straight simulation, often called a *parametric bootstrap*. The non-parallel form is

```
> library(boot)
> cd4.rg <- function(data, mle) MASS::mvrnorm(nrow(data), mle$m, mle$v)
> cd4.mle <- list(m = colMeans(cd4), v = var(cd4))
> cd4.boot <- boot(cd4, corr, R = 999, sim = "parametric",
+               ran.gen = cd4.rg, mle = cd4.mle)
> boot.ci(cd4.boot, type = c("norm", "basic", "perc"),
+       conf = 0.9, h = atanh, hinv = tanh)
```

To do this with `mclapply` we need to break this into separate runs, and we will illustrate two runs of 500 simulations each:

```

> cd4.rg <- function(data, mle) MASS::mvrnorm(nrow(data), mle$m, mle$v)
> cd4.mle <- list(m = colMeans(cd4), v = var(cd4))
> run1 <- function(...) boot(cd4, corr, R = 500, sim = "parametric",
+                             ran.gen = cd4.rg, mle = cd4.mle)
> mc <- 2 # set as appropriate for your hardware
> ## To make this reproducible:
> set.seed(123, "L'Ecuyer")
> cd4.boot <- do.call(c, mclapply(seq_len(mc), run1) )
> boot.ci(cd4.boot, type = c("norm", "basic", "perc"),
+         conf = 0.9, h = atanh, hinv = tanh)

```

There are many ways to program things like this: often the neatest is to encapsulate the computation in a function, so this is the parallel form of

```

> do.call(c, lapply(seq_len(mc), run1))

```

To run this with `parLapply` we could take a similar approach by

```

> run1 <- function(...) {
+   library(boot)
+   cd4.rg <- function(data, mle) MASS::mvrnorm(nrow(data), mle$m, mle$v)
+   cd4.mle <- list(m = colMeans(cd4), v = var(cd4))
+   boot(cd4, corr, R = 500, sim = "parametric",
+         ran.gen = cd4.rg, mle = cd4.mle)
+ }
> cl <- makeCluster(mc)
> ## make this reproducible
> clusterSetRNGStream(cl, 123)
> library(boot) # needed for c() method on master
> cd4.boot <- do.call(c, parLapply(cl, seq_len(mc), run1) )
> boot.ci(cd4.boot, type = c("norm", "basic", "perc"),
+         conf = 0.9, h = atanh, hinv = tanh)
> stopCluster(cl)

```

Note that whereas with `mclapply` all the packages and objects we use are automatically available on the workers, this is not in general⁷ the case with the `parLapply` approach. There is often a delicate choice of where to do the computations: for example we could compute `cd4.mle` on the workers (as above) or on the master and send the value to the workers. We illustrate the latter by the following code

```

> cl <- makeCluster(mc)
> cd4.rg <- function(data, mle) MASS::mvrnorm(nrow(data), mle$m, mle$v)
> cd4.mle <- list(m = colMeans(cd4), v = var(cd4))
> clusterExport(cl, c("cd4.rg", "cd4.mle"))
> junk <- clusterEvalQ(cl, library(boot)) # discard result
> clusterSetRNGStream(cl, 123)
> res <- clusterEvalQ(cl, boot(cd4, corr, R = 500,
+                             sim = "parametric", ran.gen = cd4.rg, mle = cd4.mle))
> library(boot) # needed for c() method on master
> cd4.boot <- do.call(c, res)
> boot.ci(cd4.boot, type = c("norm", "basic", "perc"),
+         conf = 0.9, h = atanh, hinv = tanh)
> stopCluster(cl)

```

⁷it is with clusters set up with `makeForkCluster`.

Running the double bootstrap on the same problem is far more computer-intensive. The standard version is

```
> R <- 999; M <- 999 ## we would like at least 999 each
> cd4.nest <- boot(cd4, nested.corr, R=R, stype="w", t0=corr(cd4), M=M)
> ## nested.corr is a function in package boot
> op <- par(pty = "s", xaxs = "i", yaxs = "i")
> qqplot((1:R)/(R+1), cd4.nest$t[, 2], pch = ".", asp = 1,
+        xlab = "nominal", ylab = "estimated")
> abline(a = 0, b = 1, col = "grey")
> abline(h = 0.05, col = "grey")
> abline(h = 0.95, col = "grey")
> par(op)
> nominal <- (1:R)/(R+1)
> actual <- cd4.nest$t[, 2]
> 100*nominal[c(sum(actual <= 0.05), sum(actual < 0.95))]
```

which took about 55 secs on one core of an 8-core Linux server.

Using `mclapply` we could use

```
> mc <- 9
> R <- 999; M <- 999; RR <- floor(R/mc)
> run2 <- function(...)
+   cd4.nest <- boot(cd4, nested.corr, R=RR, stype="w", t0=corr(cd4), M=M)
> cd4.nest <- do.call(c, mclapply(seq_len(mc), run2, mc.cores = mc) )
> nominal <- (1:R)/(R+1)
> actual <- cd4.nest$t[, 2]
> 100*nominal[c(sum(actual <= 0.05), sum(actual < 0.95))]
```

which ran in 11 secs (elapsed) using all of that server.

9.2 MCMC runs

Ripley (1988) discusses the maximum-likelihood estimation of the Strauss process, which is done by solving a moment equation

$$E_c T = t$$

where T is the number of R -close pairs and t is the observed value, 30 in the following example. A serial approach to the initial exploration might be

```
> library(spatial)
> towns <- ppinit("towns.dat")
> tget <- function(x, r=3.5) sum(dist(cbind(x$x, x$y)) < r)
> t0 <- tget(towns)
> R <- 1000
> c <- seq(0, 1, 0.1)
> ## res[1] = 0
> res <- c(0, sapply(c[-1], function(c)
+   mean(replicate(R, tget(Strauss(69, c=c, r=3.5))))))
> plot(c, res, type="l", ylab="E t")
> abline(h=t0, col="grey")
```

which takes about 20 seconds today, but many hours when first done in 1985. A parallel version might be

```

> run3 <- function(c) {
+   library(spatial)
+   towns <- ppinit("towns.dat") # has side effects
+   mean(replicate(R, tget(Strauss(69, c=c, r=3.5))))
+ }
> cl <- makeCluster(10, methods = FALSE)
> clusterExport(cl, c("R", "towns", "tget"))
> res <- c(0, parSapply(cl, c[-1], run3)) # 10 tasks
> stopCluster(cl)

```

which took about 4.5 secs, plus 2 secs to set up the cluster. Using a fork cluster (not on Windows) makes the startup much faster and setup easier:

```

> cl <- makeForkCluster(10) # fork after the variables have been set up
> run4 <- function(c) mean(replicate(R, tget(Strauss(69, c=c, r=3.5))))
> res <- c(0, parSapply(cl, c[-1], run4))
> stopCluster(cl)

```

As one might expect, the `mclapply` version is slightly simpler:

```

> run4 <- function(c) mean(replicate(R, tget(Strauss(69, c=c, r=3.5))))
> res <- c(0, unlist(mclapply(c[-1], run4, mc.cores = 10)))

```

If you do not have as many as 10 cores, you might want to consider load-balancing in a task like this as the time taken per simulation does vary with `c`. This can be done using `mclapply(mc.preschedule = FALSE)` or `parSapplyLB`. The disadvantage is that the results would not be reproducible (which does not matter here).

9.3 Package installation

With over 4000 R packages available, it is often helpful to do a comprehensive install in parallel. We provide facilities to do so in `install.packages` *via* a parallel `make`, but that approach may not be suitable.⁸ Some of the tasks take many times longer than others, and we do not know in advance which these are.

We illustrate an approach using package **parallel** which is used on part of the CRAN check farm. Suppose that there is a function `do_one(pkg)` which installs a single package and then returns. Then the task is to run `do_one` on as many of the `M` workers as possible whilst ensuring that all of the direct and indirect dependencies of `pkg` are installed before `pkg` itself. As the installation of a single package can block several others, we do need to allow the number of installs running simultaneously to vary: the following code achieves that, but needs to use low-level functions to do so.

```

> pkgs <- "<names of packages to be installed>"
> M <- 20 # number of parallel installs
> M <- min(M, length(pkgs))
> library(parallel)
> unlink("install_log")
> cl <- makeCluster(M, outfile = "install_log")
> clusterExport(cl, c("tars", "fakes", "gcc")) # variables needed by do_one
> ## set up available via a call to available.packages() for
> ## repositories containing all the packages involved and all their
> ## dependencies.

```

⁸A parallel `make` might not be available, and we have seen a couple of instances of package installation not working correctly when run from `make`.

```

> DL <- utils:::make_dependency_list(pkgs, available, recursive = TRUE)
> DL <- lapply(DL, function(x) x[x %in% pkgs])
> lens <- sapply(DL, length)
> ready <- names(DL[lens == 0L])
> done <- character() # packages already installed
> n <- length(ready)
> submit <- function(node, pkg)
+   parallel::sendCall(cl[[node]], do_one, list(pkg), tag = pkg)
> for (i in 1:min(n, M)) submit(i, ready[i])
> DL <- DL[!names(DL) %in% ready[1:min(n, M)]]
> av <- if(n < M) (n+1L):M else integer() # available workers
> while(length(done) < length(pkgs)) {
+   d <- parallel::recvOneResult(cl)
+   av <- c(av, d$node)
+   done <- c(done, d$tag)
+   OK <- unlist(lapply(DL, function(x) all(x %in% done) ))
+   if (!any(OK)) next
+   p <- names(DL)[OK]
+   m <- min(length(p), length(av)) # >= 1
+   for (i in 1:m) submit(av[i], p[i])
+   av <- av[-(1:m)]
+   DL <- DL[!names(DL) %in% p[1:m]]
+ }

```

9.4 Passing ...

The semantics of ... do not fit well with parallel operation, since lazy evaluation may be delayed until the tasks have been sent to the workers. This is no problem in the forking approach, as the information needed for lazy evaluation will be present in the forked workers.

For **snow**-like clusters the trick is to ensure that ... has any promises forced whilst the information is still available. This is how **boot** does it:

```

>   fn <- function(r) statistic(data, i[r, ], ...)
>   RR <- sum(R)
>   res <- if (ncpus > 1L && (have_mc || have_snow)) {
+     if (have_mc) {
+       parallel::mclapply(seq_len(RR), fn, mc.cores = ncpus)
+     } else if (have_snow) {
+       list(...) # evaluate any promises
+       if (is.null(cl)) {
+         cl <- parallel::makePSOCKcluster(rep("localhost", ncpus))
+         if(RNGkind()[1L] == "L'Ecuyer-CMRG")
+           parallel::clusterSetRNGStream(cl)
+         res <- parallel::parLapply(cl, seq_len(RR), fn)
+         parallel::stopCluster(cl)
+         res
+       } else parallel::parLapply(cl, seq_len(RR), fn)
+     }
+   } else lapply(seq_len(RR), fn)

```

Note that ... is an argument to boot, and so after

```

>   list(...) # evaluate any promises

```

it refers to objects within the evaluation frame of `boot` and hence the environment of `fn` which will therefore be sent to the workers along with `fn`.

10 Differences from earlier versions

The support of parallel RNG differs from `snow`: `multicore` had no support.

10.1 Differences from `multicore`

`multicore` had quite elaborate code to inhibit the Aqua event loop for `R.app` and the event loop for the `quartz` graphics device. This has been replaced by recording a flag in the R executable for a child process.

The version of `detectCores` here is biased towards the number of physical CPUs. This was occasioned by serious problems on Sparc Solaris where `multicore` defaulted to a silly number of processes.

Functions `fork` and `kill` have `mc` prefixes and are not exported. This avoids confusion with other packages (such as package `fork`), and note that `mckill` is not as general as `tools::pskill`.

Aliased functions `collect` and `parallel` are no longer provided.

10.2 Differences from `snow`

`snow` set a timeout that exceeded the maximum value required by POSIX, and did not provide a means to set the timeout on the workers. This resulted in process interlocks on Solaris.

`makeCluster` creates MPI or NWS clusters by calling `snow`.

`makePSOCKcluster` has been streamlined, as package `parallel` is in a known location on all systems, and `Rscript` is these days always available. Logging of workers is set up to append to the file, so multiple processes can be logged.

`parSapply` has been brought into line with `sapply`.

`clusterMap()` gains `SIMPLIFY` and `USE.NAMES` arguments to make it a parallel version of `mapply` and `Map`.

The timing interface has not been copied.


References

- Canty A, Ripley BD (1999–present). “boot: Bootstrap Functions.” URL <http://cran.r-project.org/package=boot>.
- Davison AC, Hinkley DV (1997). *Bootstrap Methods and Their Application*. Cambridge University Press, Cambridge.
- L’Ecuyer P (1999). “Good parameters and implementations for combined multiple recursive random number generators.” *Operations Research*, **47**, 195–164. URL <http://www.imo.umontreal.ca/~lecuyer/myftp/papers/combmrng2.ps>.

- L'Ecuyer P, Simard R, Chen EJ, Kelton WD (2002). "An object-oriented random-number package with many long streams and substreams." *Operations Research*, **50**, 1073–5. URL <http://www.iro.umontreal.ca/~lecuyer/myftp/papers/streams00.pdf>.
- Leydold J (2005–present). "rstream: Streams of random numbers." URL <http://cran.r-project.org/package=rstream>.
- Ripley BD (1988). *Statistical Inference for Spatial Processes*. Cambridge University Press, Cambridge.
- Sevcikova H, Rossini T (2004–present). "rlecuyer: R interface to RNG with multiple streams." URL <http://cran.r-project.org/package=rlecuyer>.
- Tierney L, Rossini AJ, Li N, Sevcikova H (2003–present). "Simple Network of WorkStations for R." URL <http://www.stat.uiowa.edu/~luke/R/cluster/cluster.html>.
- Urbanek S (2009–present). "multicore: Parallel processing of R code on machines with multiple cores or CPUs." URL <http://cran.r-project.org/package=multicore>.

Octave

Introduction

GNU Octave is a high-level interpreted language, primarily intended for numerical computations. It provides capabilities for the numerical solution of linear and nonlinear problems, and for performing other numerical experiments. It also provides extensive graphics capabilities for data visualization and manipulation. Octave is normally used through its interactive command line interface, but it can also be used to write non-interactive programs. The Octave language is quite similar to Matlab so that most programs are easily portable. Read more on <http://www.gnu.org/software/octave/> 

Two versions of octave are available on Anselm, via module

Versionmodule

Octave Octave/3.8.2-
3.8.2, gimkl-
com- 2.11.5
piled
with
GCC
and
Mul-
ti-
threaded
MKL

Octave Octave/4.0.1-
4.0.1, gimkl-
com- 2.11.5
piled
with
GCC
and Mul-
ti-
threaded
MKL

Octave Octave/4.0.0-
4.0.0, foss-
com- 2015g
piled
with >GCC
and
Open-
BLAS

Modules and execution

```
$ module load Octave
```

The octave on Anselm is linked to highly optimized MKL mathematical library. This provides threaded parallelization to many octave kernels, notably the linear algebra subroutines. Octave runs these heavy calculation kernels without any penalty. By default, octave would parallelize to 16 threads. You may control the threads by setting the OMP_NUM_THREADS environment variable.

To run octave interactively, log in with ssh -X parameter for X11 forwarding. Run octave:

```
$ octave
```

To run octave in batch mode, write an octave script, then write a bash jobscript and execute via the qsub command. By default, octave will use 16 threads when running MKL kernels.

```
#!/bin/bash

# change to local scratch directory
cd /lscratch/$PBS_JOBID || exit

# copy input file to scratch
cp $PBS_O_WORKDIR/octcode.m .

# load octave module
module load octave

# execute the calculation
octave -q --eval octcode > output.out

# copy output file to home
cp output.out $PBS_O_WORKDIR/.

#exit
exit
```

This script may be submitted directly to the PBS workload manager via the qsub command. The inputs are in octcode.m file, outputs in output.out file. See the single node jobscript example in the Job execution section [\[4\]](#).

The octave c compiler mkcoctfile calls the GNU gcc 4.8.1 for compiling native c code. This is very useful for running native c subroutines in octave environment.

```
$ mkcoctfile -v
```

Octave may use MPI for interprocess communication This functionality is currently not supported on Anselm cluster. In case you require the octave interface to MPI, please contact Anselm support [\[4\]](#).

Xeon Phi Support

Octave may take advantage of the Xeon Phi accelerators. This will only work on the Intel Xeon Phi accelerated nodes.

Automatic offload support

Octave can accelerate BLAS type operations (in particular the Matrix Matrix multiplications] on the Xeon Phi accelerator, via Automatic Offload using the MKL library

Example

```
$ export OFFLOAD_REPORT=2
$ export MKL_MIC_ENABLE=1
$ module load octave
$ octave -q
octave:1> A=rand(10000); B=rand(10000);
octave:2> tic; C=A*B; toc
[MKL] [MIC --] [AO Function]      DGEMM
[MKL] [MIC --] [AO DGEMM Workdivision]    0.32 0.68
[MKL] [MIC 00] [AO DGEMM CPU Time]      2.896003 seconds
[MKL] [MIC 00] [AO DGEMM MIC Time]      1.967384 seconds
[MKL] [MIC 00] [AO DGEMM CPU->MIC Data]  1347200000 bytes
[MKL] [MIC 00] [AO DGEMM MIC->CPU Data]  2188800000 bytes
Elapsed time is 2.93701 seconds.
```


In this example, the calculation was automatically divided among the CPU cores and the Xeon Phi MIC accelerator, reducing the total runtime from 6.3 secs down to 2.9 secs.

Native support

A version of native Octave is compiled for Xeon Phi accelerators. Some limitations apply for this version:

- Only command line support. GUI, graph plotting etc. is not supported.
- Command history in interactive mode is not supported.

Octave is linked with parallel Intel MKL, so it best suited for batch processing of tasks that utilize BLAS, LAPACK and FFT operations. By default, number of threads is set to 120, you can control this with `> OMP_NUM_THREADS` environment variable.

!!! Note “Note” Calculations that do not employ parallelism (either by using parallel MKL eg. via matrix operations, `fork()` function, parallel package  or other mechanism) will actually run slower than on host CPU.

To use Octave on a node with Xeon Phi:

```
$ ssh mic0 # login to the MIC card
$ source /apps/tools/octave/3.8.2-mic/bin/octave-env.sh # set up environment variables
$ octave -q /apps/tools/octave/3.8.2-mic/example/test0.m # run an example
```

R

Introduction

The R is a language and environment for statistical computing and graphics. R provides a wide variety of statistical (linear and nonlinear modelling, classical statistical tests, time-series analysis, classification, clustering, ...) and graphical techniques, and is highly extensible.

One of R's strengths is the ease with which well-designed publication-quality plots can be produced, including mathematical symbols and formulae where needed. Great care has been taken over the defaults for the minor design choices in graphics, but the user retains full control.

Another convenience is the ease with which the C code or third party libraries may be integrated within R.

Extensive support for parallel computing is available within R.

Read more on <http://www.r-project.org/>, <http://cran.r-project.org/doc/manuals/r-release/R-lang.html>

Modules

The R version 3.0.1 is available on Anselm, along with GUI interface Rstudio

Application	Version
R	R 3.0.1
Rstudio	Rstudio 0.97

```
$ module load R
```

Execution

The R on Anselm is linked to highly optimized MKL mathematical library. This provides threaded parallelization to many R kernels, notably the linear algebra subroutines. The R runs these heavy calculation kernels without any penalty. By default, the R would parallelize to 16 threads. You may control the threads by setting the OMP_NUM_THREADS environment variable.

Interactive execution

To run R interactively, using Rstudio GUI, log in with ssh -X parameter for X11 forwarding. Run rstudio:

```
$ module load Rstudio
$ rstudio
```

Batch execution

To run R in batch mode, write an R script, then write a bash jobscript and execute via the qsub command. By default, R will use 16 threads when running MKL kernels.

Example jobscript:

```
#!/bin/bash

# change to local scratch directory
cd /lscratch/$PBS_JOBID || exit

# copy input file to scratch
cp $PBS_O_WORKDIR/rscript.R .

# load R module
module load R

# execute the calculation
R CMD BATCH rscript.R routput.out

# copy output file to home
cp routput.out $PBS_O_WORKDIR/.

#exit
exit
```

This script may be submitted directly to the PBS workload manager via the qsub command. The inputs are in rscript.R file, outputs in routput.out file. See the single node jobscript example in the Job execution section.

Parallel R

Parallel execution of R may be achieved in many ways. One approach is the implied parallelization due to linked libraries or specially enabled functions, as described above. In the following sections, we focus on explicit parallelization, where parallel constructs are directly stated within the R script.

Package parallel

The package parallel provides support for parallel computation, including by forking (taken from package multicore), by sockets (taken from package snow) and random-number generation.

The package is activated this way:

```
$ R
> library(parallel)
```

More information and examples may be obtained directly by reading the documentation available in R

```
> ?parallel
> library(help = "parallel")
> vignette("parallel")
```

Download the package parallel vignette.

The forking is the most simple to use. Forking family of functions provide parallelized, drop in replacement for the serial apply() family of functions.

!!! Note “Note” Forking via package parallel provides functionality similar to OpenMP construct
omp parallel for

Only cores of single node can be utilized this way!

Forking example:

```
library(parallel)

#integrand function
f <- function(i,h) {
  x <- h*(i-0.5)
  return (4/(1 + x*x))
}

#initialize
size <- detectCores()

while (TRUE)
{
  #read number of intervals
  cat("Enter the number of intervals: (0 quits) ")
  fp<-file("stdin"); n<-scan(fp,nmax=1); close(fp)

  if(n<=0) break

  #run the calculation
  n <- max(n,size)
  h <- 1.0/n

  i <- seq(1,n);
  pi3 <- h*sum(simplify2array(mclapply(i,f,h,mc.cores=size)));

  #print results
  cat(sprintf("Value of PI %16.14f, diff= %16.14fn",pi3,pi3-pi))
}
```

The above example is the classic parallel example for calculating the number π . Note the **detectCores()** and **mclapply()** functions. Execute the example as:

```
$ R --slave --no-save --no-restore -f pi3p.R
```

Every evaluation of the integrand function runs in parallel on different process.

Package Rmpi

!!! Note “Note” package Rmpi provides an interface (wrapper) to MPI APIs.

It also provides interactive R slave environment. On Anselm, Rmpi provides interface to the Open-MPI.

Read more on Rmpi at <http://cran.r-project.org/web/packages/Rmpi/>, reference manual is available at <http://cran.r-project.org/web/packages/Rmpi/Rmpi.pdf>

When using package Rmpi, both openmpi and R modules must be loaded

```
$ module load openmpi
$ module load R
```

Rmpi may be used in three basic ways. The static approach is identical to executing any other MPI program. In addition, there is Rslaves dynamic MPI approach and the mpi.apply approach. In the following section, we will use the number integration example, to illustrate all these concepts.

static Rmpi

Static Rmpi programs are executed via mpiexec, as any other MPI programs. Number of processes is static - given at the launch time.

Static Rmpi example:

```
library(Rmpi)

#integrand function
f <- function(i,h) {
  x <- h*(i-0.5)
  return (4/(1 + x*x))
}

#initialize
invisible(mpi.comm.dup(0,1))
rank <- mpi.comm.rank()
size <- mpi.comm.size()
n<-0

while (TRUE)
{
  #read number of intervals
  if (rank==0) {
    cat("Enter the number of intervals: (0 quits) ")
    fp<-file("stdin"); n<-scan(fp,nmax=1); close(fp)
  }

  #broadcast the intervals
  n <- mpi.bcast(as.integer(n),type=1)

  if(n<=0) break

  #run the calculation
  n <- max(n,size)
  h <- 1.0/n

  i <- seq(rank+1,n,size);
  mypi <- h*sum(sapply(i,f,h));

  pi3 <- mpi.reduce(mypi)

  #print results
  if (rank==0) cat(sprintf("Value of PI %16.14f, diff= %16.14fn",pi3,pi3-pi))
}

mpi.quit()
```

The above is the static MPI example for calculating the number π . Note the **library(Rmpi)** and **mpi.comm.dup()** function calls.

Execute the example as:

```
$ mpiexec R --slave --no-save --no-restore -f pi3.R
```

dynamic Rmpi

Dynamic Rmpi programs are executed by calling the R directly. openmpi module must be still loaded. The R slave processes will be spawned by a function call within the Rmpi program.

Dynamic Rmpi example:

```
#integrand function
f <- function(i,h) {
  x <- h*(i-0.5)
  return (4/(1 + x*x))
}

#the worker function
workerpi <- function()
{
  #initialize
  rank <- mpi.comm.rank()
  size <- mpi.comm.size()
  n<-0

  while (TRUE)
  {
    #read number of intervals
    if (rank==0) {
      cat("Enter the number of intervals: (0 quits) ")
      fp<-file("stdin"); n<-scan(fp,nmax=1); close(fp)
    }

    #broadcast the intervals
    n <- mpi.bcast(as.integer(n),type=1)

    if(n<=0) break

    #run the calculation
    n <- max(n,size)
    h <- 1.0/n

    i <- seq(rank+1,n,size);
    mypi <- h*sum(sapply(i,f,h));

    pi3 <- mpi.reduce(mypi)

    #print results
    if (rank==0) cat(sprintf("Value of PI %16.14f, diff= %16.14fn",pi3,pi3-pi))
  }
}
```



```

#main
library(Rmpi)

cat("Enter the number of slaves: ")
fp<-file("stdin"); ns<-scan(fp,nmax=1); close(fp)

mpi.spawn.Rslaves(nslaves=ns)
mpi.bcast.Robj2slave(f)
mpi.bcast.Robj2slave(workerpi)

mpi.bcast.cmd(workerpi())
workerpi()

mpi.quit()

```

The above example is the dynamic MPI example for calculating the number π . Both master and slave processes carry out the calculation. Note the `mpi.spawn.Rslaves()`, `mpi.bcast.Robj2slave()`** and the `mpi.bcast.cmd()`** function calls.

Execute the example as:

```
$ R --slave --no-save --no-restore -f pi3Rslaves.R
```

mpi.apply Rmpi

`mpi.apply` is a specific way of executing Dynamic Rmpi programs.

!!! Note “Note” `mpi.apply()` family of functions provide MPI parallelized, drop in replacement for the serial `apply()` family of functions.

Execution is identical to other dynamic Rmpi programs.

`mpi.apply` Rmpi example:

```

#integrand function
f <- function(i,h) {
  x <- h*(i-0.5)
  return (4/(1 + x*x))
}

#the worker function
workerpi <- function(rank,size,n)
{
  #run the calculation
  n <- max(n,size)
  h <- 1.0/n

  i <- seq(rank,n,size);
  mypi <- h*sum(sapply(i,f,h));

  return(mypi)
}

#main

```

```

library(Rmpi)

cat("Enter the number of slaves: ")
fp<-file("stdin"); ns<-scan(fp,nmax=1); close(fp)

mpi.spawn.Rslaves(nslaves=ns)
mpi.bcast.Robj2slave(f)
mpi.bcast.Robj2slave(workerpi)

while (TRUE)
{
  #read number of intervals
  cat("Enter the number of intervals: (0 quits) ")
  fp<-file("stdin"); n<-scan(fp,nmax=1); close(fp)
  if(n<=0) break

  #run workerpi
  i=seq(1,2*ns)
  pi3=sum(mpi.parSapply(i,workerpi,2*ns,n))

  #print results
  cat(sprintf("Value of PI %16.14f, diff= %16.14fn",pi3,pi3-pi))
}

mpi.quit()

```

The above is the mpi.apply MPI example for calculating the number π . Only the slave processes carry out the calculation. Note the **mpi.parSapply()**, function call. The package parallel example above may be trivially adapted (for much better performance) to this structure using the mclapply() in place of mpi.parSapply().

Execute the example as:

```
$ R --slave --no-save --no-restore -f pi3parSapply.R
```

Combining parallel and Rmpi

Currently, the two packages can not be combined for hybrid calculations.

Parallel execution

The R parallel jobs are executed via the PBS queue system exactly as any other parallel jobs. User must create an appropriate jobscript and submit via the **qsub**

Example jobscript for static Rmpi parallel R execution, running 1 process per core:

```

#!/bin/bash
#PBS -q qprod
#PBS -N Rjob
#PBS -l select=100:ncpus=16:mpiprocs=16:ompthreads=1

# change to scratch directory
SCRDIR=/scratch/$USER/myjob

```

```
cd $SCRDIR || exit

# copy input file to scratch
cp $PBS_O_WORKDIR/rscript.R .

# load R and openmpi module
module load R
module load openmpi

# execute the calculation
mpiexec -bycore -bind-to-core R --slave --no-save --no-restore -f rscript.R

# copy output file to home
cp routput.out $PBS_O_WORKDIR/.

#exit
exit
```

For more information about jobscripts and MPI execution refer to the Job submission and general MPI sections.

Numerical languages

Interpreted languages for numerical computations and analysis

Introduction

This section contains a collection of high-level interpreted languages, primarily intended for numerical computations.

Matlab

MATLAB® is a high-level language and interactive environment for numerical computation, visualization, and programming.

```
$ module load MATLAB/2015b-EDU  
$ matlab
```

Read more at the Matlab page.

Octave

GNU Octave is a high-level interpreted language, primarily intended for numerical computations. The Octave language is quite similar to Matlab so that most programs are easily portable.

```
$ module load Octave  
$ octave
```

Read more at the Octave page.

R

The R is an interpreted language and environment for statistical computing and graphics.

```
$ module load R  
$ R
```

Read more at the R page.

Matlab 2013-2014

Introduction

!!! Note “Note” This document relates to the old versions R2013 and R2014. For MATLAB 2015, please use this documentation instead.

Matlab is available in the latest stable version. There are always two variants of the release:

- Non commercial or so called EDU variant, which can be used for common research and educational purposes.
- Commercial or so called COM variant, which can be used also for commercial activities. The licenses for commercial variant are much more expensive, so usually the commercial variant has only subset of features compared to the EDU available.

To load the latest version of Matlab load the module

```
$ module load matlab
```

By default the EDU variant is marked as default. If you need other version or variant, load the particular version. To obtain the list of available versions use

```
$ module avail matlab
```

If you need to use the Matlab GUI to prepare your Matlab programs, you can use Matlab directly on the login nodes. But for all computations use Matlab on the compute nodes via PBS Pro scheduler.

If you require the Matlab GUI, please follow the general informations about running graphical applications

Matlab GUI is quite slow using the X forwarding built in the PBS (qsub -X), so using X11 display redirection either via SSH or directly by xauth (please see the “GUI Applications on Compute Nodes over VNC” part) is recommended.

To run Matlab with GUI, use

```
$ matlab
```

To run Matlab in text mode, without the Matlab Desktop GUI environment, use

```
$ matlab -nodesktop -nosplash
```

plots, images, etc... will be still available.

Running parallel Matlab using Distributed Computing Toolbox / Engine

Recommended parallel mode for running parallel Matlab on Anselm is MPIEXEC mode. In this mode user allocates resources through PBS prior to starting Matlab. Once resources are granted the main Matlab instance is started on the first compute node assigned to job by PBS and workers are started on all remaining nodes. User can use both interactive and non-interactive PBS sessions. This mode guarantees that the data processing is not performed on login nodes, but all processing is on compute nodes.

For the performance reasons Matlab should use system MPI. On Anselm the supported MPI implementation for Matlab is Intel MPI. To switch to system MPI user has to override default Matlab setting by creating new configuration file in its home directory. The path and file name has to be exactly the same as in the following listing:

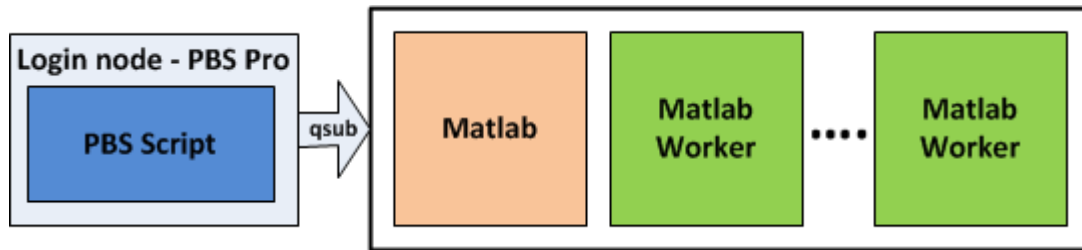


Figure 1: Parallel Matlab

```

$ vim ~/matlab/mpiLibConf.m

function [lib, extras] = mpiLibConf
%MATLAB MPI Library overloading for Infiniband Networks

mpich = '/opt/intel/impi/4.1.1.036/lib64/';

disp('Using Intel MPI 4.1.1.036 over Infiniband')

lib = strcat(mpich, 'libmpich.so');
mpl = strcat(mpich, 'libmpl.so');
opa = strcat(mpich, 'libopa.so');

extras = {};
  
```

System MPI library allows Matlab to communicate through 40Gbps Infiniband QDR interconnect instead of slower 1Gb ethernet network.

!!! Note “Note” Please note: The path to MPI library in “mpiLibConf.m” has to match with version of loaded Intel MPI module. In this example the version 4.1.1.036 of Intel MPI is used by Matlab and therefore module impi/4.1.1.036 has to be loaded prior to starting Matlab.

Parallel Matlab interactive session

Once this file is in place, user can request resources from PBS. Following example shows how to start interactive session with support for Matlab GUI. For more information about GUI based applications on Anselm see.

```

$ xhost +
$ qsub -I -v DISPLAY=$(uname -n):$(echo $DISPLAY | cut -d ':' -f 2) -A NONE-0-0 -q qe
-l feature__matlab__MATLAB=1
  
```

This qsub command example shows how to run Matlab with 32 workers in following configuration: 2 nodes (use all 16 cores per node) and 16 workers = mpirocs per node (-l select=2:ncpus=16:mpiprocs=16). If user requires to run smaller number of workers per node then the “mpiprocs” parameter has to be changed.

The second part of the command shows how to request all necessary licenses. In this case 1 Matlab-EDU license and 32 Distributed Computing Engines licenses.

Once the access to compute nodes is granted by PBS, user can load following modules and start Matlab:

```

cn79$ module load matlab/R2013a-EDU
cn79$ module load impi/4.1.1.036
cn79$ matlab &
  
```

Parallel Matlab batch job

To run matlab in batch mode, write an matlab script, then write a bash jobscript and execute via the qsub command. By default, matlab will execute one matlab worker instance per allocated core.

```
#!/bin/bash
#PBS -A PROJECT ID
#PBS -q qprod
#PBS -l select=2:ncpus=16:mpiprocs=16:ompthreads=1

# change to shared scratch directory
SCR=/scratch/$USER/$PBS_JOBID
mkdir -p $SCR ; cd $SCR || exit

# copy input file to scratch
cp $PBS_O_WORKDIR/matlabcode.m .

# load modules
module load matlab/R2013a-EDU
module load impi/4.1.1.036

# execute the calculation
matlab -nodisplay -r matlabcode > output.out

# copy output file to home
cp output.out $PBS_O_WORKDIR/.
```

This script may be submitted directly to the PBS workload manager via the qsub command. The inputs and matlab script are in matlabcode.m file, outputs in output.out file. Note the missing .m extension in the matlab -r matlabcodefile call, **the .m must not be included**. Note that the **shared /scratch must be used**. Further, it is **important to include quit** statement at the end of the matlabcode.m script.

Submit the jobscript using qsub

```
$ qsub ./jobscript
```

Parallel Matlab program example

The last part of the configuration is done directly in the user Matlab script before Distributed Computing Toolbox is started.

```
sched = findResource('scheduler', 'type', 'mpiexec');
set(sched, 'MpiexecFileName', '/apps/intel/impi/4.1.1/bin/mpirun');
set(sched, 'EnvironmentSetMethod', 'setenv');
```

This script creates scheduler object “sched” of type “mpiexec” that starts workers using mpirun tool. To use correct version of mpirun, the second line specifies the path to correct version of system Intel MPI library.

!!! Note “Note” Please note: Every Matlab script that needs to initialize/use matlabpool has to contain these three lines prior to calling matlabpool(sched, ...) function.

The last step is to start matlabpool with “sched” object and correct number of workers. In this case qsub asked for total number of 32 cores, therefore the number of workers is also set to 32.

```
matlabpool(sched,32);
```

```
... parallel code ...
```

```
matlabpool close
```

The complete example showing how to use Distributed Computing Toolbox is show here.

```
sched = findResource('scheduler', 'type', 'mpiexec');
set(sched, 'MpiexecFileName', '/apps/intel/impi/4.1.1/bin/mpirun')
set(sched, 'EnvironmentSetMethod', 'setenv')
set(sched, 'SubmitArguments', '')
sched

matlabpool(sched,32);

n=2000;

W = rand(n,n);
W = distributed(W);
x = (1:n)';
x = distributed(x);
spmd
[~, name] = system('hostname')

    T = W*x; % Calculation performed on labs, in parallel.
           % T and W are both codistributed arrays here.
end
T;
whos           % T and W are both distributed arrays here.

matlabpool close
quit
```

You can copy and paste the example in a .m file and execute. Note that the matlabpool size should correspond to **total number of cores** available on allocated nodes.

Non-interactive Session and Licenses

If you want to run batch jobs with Matlab, be sure to request appropriate license features with the PBS Pro scheduler, at least the "-l __feature__matlab__MATLAB=1" for EDU variant of Matlab. More information about how to check the license features states and how to request them with PBS Pro, please look here.

In case of non-interactive session please read the following information on how to modify the qsub command to test for available licenses prior getting the resource allocation.

Matlab Distributed Computing Engines start up time

Starting Matlab workers is an expensive process that requires certain amount of time. For your information please see the following table:

compute nodes	number of workers	start-up time[s]
16	256	1008
8	128	534
4	64	333
2	32	210

Matlab

Introduction

Matlab is available in versions R2015a and R2015b. There are always two variants of the release:

- Non commercial or so called EDU variant, which can be used for common research and educational purposes.
- Commercial or so called COM variant, which can be used also for commercial activities. The licenses for commercial variant are much more expensive, so usually the commercial variant has only subset of features compared to the EDU available.

To load the latest version of Matlab load the module

```
$ module load MATLAB
```

By default the EDU variant is marked as default. If you need other version or variant, load the particular version. To obtain the list of available versions use

```
$ module avail MATLAB
```

If you need to use the Matlab GUI to prepare your Matlab programs, you can use Matlab directly on the login nodes. But for all computations use Matlab on the compute nodes via PBS Pro scheduler.

If you require the Matlab GUI, please follow the general informations about running graphical applications.

Matlab GUI is quite slow using the X forwarding built in the PBS (qsub -X), so using X11 display redirection either via SSH or directly by xauth (please see the “GUI Applications on Compute Nodes over VNC” part here) is recommended.

To run Matlab with GUI, use

```
$ matlab
```

To run Matlab in text mode, without the Matlab Desktop GUI environment, use

```
$ matlab -nodesktop -nosplash
```

plots, images, etc... will be still available.

Running parallel Matlab using Distributed Computing Toolbox / Engine

!!! Note “Note” Distributed toolbox is available only for the EDU variant

The MPIEXEC mode available in previous versions is no longer available in MATLAB 2015. Also, the programming interface has changed. Refer to Release Notes [here](#).

Delete previously used file mpiLibConf.m, we have observed crashes when using Intel MPI.

To use Distributed Computing, you first need to setup a parallel profile. We have provided the profile for you, you can either import it in MATLAB command line:

```
>> parallel.importProfile('/apps/all/MATLAB/2015a-EDU/SalomonPBSPro.settings')  
  
ans =  
  
SalomonPBSPro
```

Or in the GUI, go to tab HOME -> Parallel -> Manage Cluster Profiles..., click Import and navigate to:

```
/apps/all/MATLAB/2015a-EDU/SalomonPBSPro.settings
```

With the new mode, MATLAB itself launches the workers via PBS, so you can either use interactive mode or a batch mode on one node, but the actual parallel processing will be done in a separate job started by MATLAB itself. Alternatively, you can use “local” mode to run parallel code on just a single node.

!!! Note “Note” The profile is confusingly named Salomon, but you can use it also on Anselm.

Parallel Matlab interactive session

Following example shows how to start interactive session with support for Matlab GUI. For more information about GUI based applications on Anselm see this page.

```
$ xhost +  
$ qsub -I -v DISPLAY=$(uname -n):$(echo $DISPLAY | cut -d ':' -f 2) -A NONE-0-0 -q qe  
-l feature__matlab__MATLAB=1
```

This qsub command example shows how to run Matlab on a single node.

The second part of the command shows how to request all necessary licenses. In this case 1 Matlab-EDU license and 48 Distributed Computing Engines licenses.

Once the access to compute nodes is granted by PBS, user can load following modules and start Matlab:

```
r1i0n17$ module load MATLAB/2015b-EDU  
r1i0n17$ matlab &
```

Parallel Matlab batch job in Local mode

To run matlab in batch mode, write an matlab script, then write a bash jobscript and execute via the qsub command. By default, matlab will execute one matlab worker instance per allocated core.

```
#!/bin/bash  
#PBS -A PROJECT ID  
#PBS -q qprod  
#PBS -l select=1:ncpus=16:mpiprocs=16:ompthreads=1  
  
# change to shared scratch directory  
SCR=/scratch/work/user/$USER/$PBS_JOBID  
mkdir -p $SCR ; cd $SCR || exit  
  
# copy input file to scratch  
cp $PBS_O_WORKDIR/matlabcode.m .  
  
# load modules  
module load MATLAB/2015a-EDU  
  
# execute the calculation  
matlab -nodisplay -r matlabcode > output.out
```

```
# copy output file to home
cp output.out $PBS_O_WORKDIR/.
```

This script may be submitted directly to the PBS workload manager via the qsub command. The inputs and matlab script are in matlabcode.m file, outputs in output.out file. Note the missing .m extension in the matlab -r matlabcodefile call, **the .m must not be included**. Note that the **shared /scratch must be used**. Further, it is **important to include quit** statement at the end of the matlabcode.m script.

Submit the jobscript using qsub

```
$ qsub ./jobscript
```

Parallel Matlab Local mode program example

The last part of the configuration is done directly in the user Matlab script before Distributed Computing Toolbox is started.

```
cluster = parcluster('local')
```

This script creates scheduler object “cluster” of type “local” that starts workers locally.

!!! Note “Note” Please note: Every Matlab script that needs to initialize/use matlabpool has to contain these three lines prior to calling parpool(sched, ...) function.

The last step is to start matlabpool with “cluster” object and correct number of workers. We have 24 cores per node, so we start 24 workers.

```
parpool(cluster,16);
```

```
... parallel code ...
```

```
parpool close
```

The complete example showing how to use Distributed Computing Toolbox in local mode is shown here.

```
cluster = parcluster('local');
cluster

parpool(cluster,24);

n=2000;

W = rand(n,n);
W = distributed(W);
x = (1:n)';
x = distributed(x);
spmd
[~, name] = system('hostname')

    T = W*x; % Calculation performed on labs, in parallel.
           % T and W are both codistributed arrays here.
end
T;
```

```
whos          % T and W are both distributed arrays here.

parpool close
quit
```

You can copy and paste the example in a .m file and execute. Note that the parpool size should correspond to **total number of cores** available on allocated nodes.

Parallel Matlab Batch job using PBS mode (workers spawned in a separate job)

This mode uses PBS scheduler to launch the parallel pool. It uses the SalomonPBSPro profile that needs to be imported to Cluster Manager, as mentioned before. This method uses MATLAB's PBS Scheduler interface - it spawns the workers in a separate job submitted by MATLAB using qsub.

This is an example of m-script using PBS mode:

```
cluster = parcluster('SalomonPBSPro');
set(cluster, 'SubmitArguments', '-A OPEN-0-0');
set(cluster, 'ResourceTemplate', '-q qprod -l select=10:ncpus=16');
set(cluster, 'NumWorkers', 160);

pool = parpool(cluster, 160);

n=2000;

W = rand(n,n);
W = distributed(W);
x = (1:n)';
x = distributed(x);
spmd
[~, name] = system('hostname')

    T = W*x; % Calculation performed on labs, in parallel.
           % T and W are both codistributed arrays here.
end
whos          % T and W are both distributed arrays here.

% shut down parallel pool
delete(pool)
```

Note that we first construct a cluster object using the imported profile, then set some important options, namely: SubmitArguments, where you need to specify accounting id, and ResourceTemplate, where you need to specify number of nodes to run the job.

You can start this script using batch mode the same way as in Local mode example.

Parallel Matlab Batch with direct launch (workers spawned within the existing job)

This method is a “hack” invented by us to emulate the mpiexec functionality found in previous MATLAB versions. We leverage the MATLAB Generic Scheduler interface, but instead of submitting the workers to PBS, we launch the workers directly within the running job, thus we avoid the issues with master script and workers running in separate jobs (issues with license not available, waiting for the worker's job to spawn etc.)

Please note that this method is experimental.

For this method, you need to use SalomonDirect profile, import it using the same way as SalomonPBSPro

This is an example of m-script using direct mode:

```
parallel.importProfile('/apps/all/MATLAB/2015a-EDU/SalomonDirect.settings')
cluster = parcluster('SalomonDirect');
set(cluster, 'NumWorkers', 48);

pool = parpool(cluster, 48);

n=2000;

W = rand(n,n);
W = distributed(W);
x = (1:n)';
x = distributed(x);
spmd
[~, name] = system('hostname')

    T = W*x; % Calculation performed on labs, in parallel.
        % T and W are both codistributed arrays here.
end
whos      % T and W are both distributed arrays here.

% shut down parallel pool
delete(pool)
```

Non-interactive Session and Licenses

If you want to run batch jobs with Matlab, be sure to request appropriate license features with the PBS Pro scheduler, at least the "-l __feature_matlab_MATLAB=1" for EDU variant of Matlab. More information about how to check the license features states and how to request them with PBS Pro, please look [here](#).

In case of non-interactive session please read the following information on how to modify the qsub command to test for available licenses prior getting the resource allocation.

Matlab Distributed Computing Engines start up time

Starting Matlab workers is an expensive process that requires certain amount of time. For your information please see the following table:

compute nodes	number of workers	start-up time[s]
16	384	831
8	192	807
4	96	483
2	48	16

MATLAB on UV2000

UV2000 machine available in queue “qfat” can be used for MATLAB computations. This is a SMP NUMA machine with large amount of RAM, which can be beneficial for certain types of MATLAB jobs. CPU cores are allocated in chunks of 8 for this machine.

You can use MATLAB on UV2000 in two parallel modes:

Threaded mode

Since this is a SMP machine, you can completely avoid using Parallel Toolbox and use only MATLAB's threading. MATLAB will automatically detect the number of cores you have allocated and will set `maxNumCompThreads` accordingly and certain operations, such as `fft`, `eig`, `svd`, etc. will be automatically run in threads. The advantage of this mode is that you don't need to modify your existing sequential codes.

Local cluster mode

You can also use Parallel Toolbox on UV2000. Use local cluster mode, “SalomonPBSPPro” profile will not work.

ISV Licenses

A guide to managing Independent Software Vendor licences

On Anselm cluster there are also installed commercial software applications, also known as ISV (Independent Software Vendor), which are subjects to licensing. The licenses are limited and their usage may be restricted only to some users or user groups.


Currently Flex License Manager based licensing is supported on the cluster for products Ansys, Comsol and Matlab. More information about the applications can be found in the general software section.

If an ISV application was purchased for educational (research) purposes and also for commercial purposes, then there are always two separate versions maintained and suffix “edu” is used in the name of the non-commercial version.

Overview of the licenses usage

!!! Note “Note” The overview is generated every minute and is accessible from web or command line interface.

Web interface

For each license there is a table, which provides the information about the name, number of available (purchased/licensed), number of used and number of free license features <https://extranet.it4i.cz/anselm/licenses> 

Text interface

For each license there is a unique text file, which provides the information about the name, number of available (purchased/licensed), number of used and number of free license features. The text files are accessible from the Anselm command prompt.

	File
	with
	li-
	cense
Product	state

ansys /apps/user/licenses/ansys_features_state.txt

comsol /apps/user/licenses/comsol_features_state.txt

comsol-/apps/user/licenses/comsol-
edu edu_features_state.txt

matlab /apps/user/licenses/matlab_features_state.txt

matlab-/apps/user/licenses/matlab-
edu edu_features_state.txt

File
with
li-
cense
Productstate

The file has a header which serves as a legend. All the info in the legend starts with a hash (#) so it can be easily filtered when parsing the file via a script.

Example of the Commercial Matlab license state:

```
$ cat /apps/user/licenses/matlab_features_state.txt
# matlab
# -----
# FEATURE                                TOTAL    USED    AVAIL
# -----
MATLAB                                1        1        0
SIMULINK                             1        0        1
Curve_Fitting_Toolbox                 1        0        1
Signal_Blocks                         1        0        1
GADS_Toolbox                          1        0        1
Image_Toolbox                         1        0        1
Compiler                             1        0        1
Neural_Network_Toolbox                1        0        1
Optimization_Toolbox                  1        0        1
Signal_Toolbox                        1        0        1
Statistics_Toolbox                    1        0        1
```

License tracking in PBS Pro scheduler and users usage

Each feature of each license is accounted and checked by the scheduler of PBS Pro. If you ask for certain licences, the scheduler won't start the job until the asked licenses are free (available). This prevents to crash batch jobs, just because of unavailability of the needed licenses.

The general format of the name is:

****feature__APP__FEATURE****

Names of applications (APP):

- ansys
- comsol
- comsol-edu
- matlab
- matlab-edu

To get the FEATURES of a license take a look into the corresponding state file (see above), or use:

Application and List of provided features - ansys \$ grep -v “#” /apps/user/licenses/ansys_features_state.txt | cut -f1 -d’ ‘ - **comsol** \$ grep -v “#” /apps/user/licenses/comsol_features_state.txt | cut -f1 -d’ ‘ - **comsol-edu** \$ grep -v “#” /apps/user/licenses/comsol-edu_features_state.txt | cut -f1 -d’ ‘ - **matlab** \$ grep -v “#” /apps/user/licenses/matlab_features_state.txt | cut -f1 -d’ ‘ - **matlab-edu** \$ grep -v “#” /apps/user/licenses/matlab-edu_features_state.txt | cut -f1 -d’ ‘

Example of PBS Pro resource name, based on APP and FEATURE name:

Application	Feature
ansys	acfd
ansys	aa_r
comsol	COMSOL
comsol	HEATTRANSFER
comsol-COMSOLBATCH	edu
comsol-STRUCTURALMECHANICS	edu
matlab	MATLAB
matlab	Image_Toolbox
matlab-MATLAB_Distrib_Comp_Engine	edu
matlab-Image_Acquisition_Toolbox	edu

Be aware, that the resource names in PBS Pro are CASE SENSITIVE!

Example of qsub statement

Run an interactive PBS job with 1 Matlab EDU license, 1 Distributed Computing Toolbox and 32 Distributed Computing Engines (running on 32 cores):

```
$ qsub -I -q qprod -A PROJECT_ID -l select=2:ncpus=16 -l feature__matlab-edu__MATLAB=
```

The license is used and accounted only with the real usage of the product. So in this example, the general Matlab is used after Matlab is run by the user and not at the time, when the shell of the interactive job is started. Also the Distributed Computing licenses are used at the time, when the user uses the distributed parallel computation in Matlab (e. g. issues pmode start, matlabpool, etc.).

OpenFOAM

A free, open source CFD software package

Introduction

OpenFOAM is a free, open source CFD software package developed by **OpenCFD Ltd** at **ESI Group** and distributed by the **OpenFOAM Foundation**. It has a large user base across most areas of engineering and science, from both commercial and academic organisations.

Homepage: <http://www.openfoam.com/>

Installed version

Currently, several version compiled by GCC/ICC compilers in single/double precision with several version of openmpi are available on Anselm.

For example syntax of available OpenFOAM module is:

< openfoam/2.2.1-icc-openmpi1.6.5-DP >

this means openfoam version 2.2.1 compiled by ICC compiler with openmpi1.6.5 in double precision.

Naming convention of the installed versions is following:

openfoam/<>VERSION>>-<>COMPILER>-<openmpiVERSION>-<PRECISION>

- <VERSION>> - version of openfoam
- <COMPILER> - version of used compiler
- <openmpiVERSION> - version of used openmpi/impi
- <PRECISION> - DP/SP – double/single precision

Available OpenFOAM modules

To check available modules use

```
$ module avail
```

In /opt/modules/modulefiles/engineering you can see installed engineering softwares:

```
----- /opt/modules/modulefiles/engineering -----
ansys/14.5.x          matlab/R2013a-COM      openfoam/
comsol/43b-COM        matlab/R2013a-EDU      openfoam/
comsol/43b-EDU        openfoam/2.2.1-gcc481-openmpi1.6.5-DP  paraview/
lsdyna/7.x.x          openfoam/2.2.1-gcc481-openmpi1.6.5-SP
```

For information how to use modules please look here.

Getting Started

To create OpenFOAM environment on ANSELM give the commands:

```
$ module load openfoam/2.2.1-icc-openmpi1.6.5-DP
```

```
$ source $FOAM_BASHRC
```

!!! Note “Note” Please load correct module with your requirements “compiler - GCC/ICC, precision - DP/SP”.

Create a project directory within the \$HOME/OpenFOAM directory named ><USER>-<OFversion> and create a directory named run within it, e.g. by typing:

```
$ mkdir -p $FOAM_RUN
```

Project directory is now available by typing:

```
$ cd /home/<USER>/OpenFOAM/<USER>-<OFversion>/run
```

<OFversion> - for example <2.2.1>

or

```
$ cd $FOAM_RUN
```

Copy the tutorial examples directory in the OpenFOAM distribution to the run directory:

```
$ cp -r $FOAM_TUTORIALS $FOAM_RUN
```

Now you can run the first case for example incompressible laminar flow in a cavity.

Running Serial Applications

Create a Bash script >test.sh

```
#!/bin/bash
module load openfoam/2.2.1-icc-openmpi1.6.5-DP
source $FOAM_BASHRC

# source to run functions
. $WM_PROJECT_DIR/bin/tools/RunFunctions

cd $FOAM_RUN/tutorials/incompressible/icoFoam/cavity

runApplication blockMesh
runApplication icoFoam
```

Job submission

```
$ qsub -A OPEN-0-0 -q qprod -l select=1:ncpus=16,walltime=03:00:00 test.sh
```

For information about job submission please look here.

Running applications in parallel

Run the second case for example external incompressible turbulent flow - case - motorBike.

First we must run serial application blockMesh and decomposePar for preparation of parallel computation.

!!! Note “Note” Create a Bash scrip test.sh:

```
#!/bin/bash
module load openfoam/2.2.1-icc-openmpi1.6.5-DP
source $FOAM_BASHRC
```

```
# source to run functions
. $WM_PROJECT_DIR/bin/tools/RunFunctions

cd $FOAM_RUN/tutorials/incompressible/simpleFoam/motorBike

runApplication blockMesh
runApplication decomposePar
```

Job submission

```
$ qsub -A OPEN-0-0 -q qprod -l select=1:ncpus=16,walltime=03:00:00 test.sh
```

This job create simple block mesh and domain decomposition. Check your decomposition, and submit parallel computation:

!!! Note “Note” Create a PBS script testParallel.pbs:

```
#!/bin/bash
#PBS -N motorBike
#PBS -l select=2:ncpus=16
#PBS -l walltime=01:00:00
#PBS -q qprod
#PBS -A OPEN-0-0

module load openfoam/2.2.1-icc-openmpi1.6.5-DP
source $FOAM_BASHRC

cd $FOAM_RUN/tutorials/incompressible/simpleFoam/motorBike

nproc = 32

mpirun -hostfile ${PBS_NODEFILE} -np $nproc snappyHexMesh -overwrite -parallel | tee
mpirun -hostfile ${PBS_NODEFILE} -np $nproc potentialFoam -noFunctionObject-writep -p
mpirun -hostfile ${PBS_NODEFILE} -np $nproc simpleFoam -parallel | tee simpleFoam.log
```

nproc – number of subdomains

Job submission

```
$ qsub testParallel.pbs
```

Compile your own solver

Initialize OpenFOAM environment before compiling your solver

```
$ module load openfoam/2.2.1-icc-openmpi1.6.5-DP
$ source $FOAM_BASHRC
$ cd $FOAM_RUN/
```

Create directory applications/solvers in user directory

```
$ mkdir -p applications/solvers
$ cd applications/solvers
```

Copy icoFoam solver’s source files

```
$ cp -r $FOAM_SOLVERS/incompressible/icoFoam/ My_icoFoam
$ cd My_icoFoam
```

Rename icoFoam.C to My_icoFOAM.C

```
$ mv icoFoam.C My_icoFoam.C
```

Edit *>files* file in *Make* directory:

```
icoFoam.C
EXE = $(FOAM_APPBIN)/icoFoam
```

and change to:


```
My_icoFoam.C
EXE = $(FOAM_USER_APPBIN)/My_icoFoam
```

In directory My_icoFoam give the compilation command:

```
$ wmake
```

Have a fun with OpenFOAM :)

LS-DYNA

LS-DYNA  is a multi-purpose, explicit and implicit finite element program used to analyze the nonlinear dynamic response of structures. Its fully automated contact analysis capability, a wide range of constitutive models to simulate a whole range of engineering materials (steels, composites, foams, concrete, etc.), error-checking features and the high scalability have enabled users worldwide to solve successfully many complex problems. Additionally LS-DYNA is extensively used to simulate impacts on structures from drop tests, underwater shock, explosions or high-velocity impacts. Explosive forming, process engineering, accident reconstruction, vehicle dynamics, thermal brake disc analysis or nuclear safety are further areas in the broad range of possible applications. In leading-edge research LS-DYNA is used to investigate the behaviour of materials like composites, ceramics, concrete, or wood. Moreover, it is used in biomechanics, human modelling, molecular structures, casting, forging, or virtual testing.

Anselm provides **1 commercial license of LS-DYNA without HPC** support now.

To run LS-DYNA in batch mode you can utilize/modify the default lsdyna.pbs script and execute it via the qsub command.

```
#!/bin/bash
#PBS -l nodes=1:ppn=16
#PBS -q qprod
#PBS -N $USER-LSDYNA-Project
#PBS -A XX-YY-ZZ


#! Mail to user when job terminate or abort
#PBS -m ae

#!change the working directory (default is home directory)
#cd <working directory> (working directory must exists)
WORK_DIR="/scratch/$USER/work"
cd $WORK_DIR

echo Running on host `hostname`
echo Time is `date`
echo Directory is `pwd`


module load lsdyna

/apps/engineering/lsdyna/lsdyna700s i=input.k
```

Header of the pbs file (above) is common and description can be find on this site. SVS FEM  recommends to utilize sources by keywords: nodes, ppn. These keywords allows to address directly the number of nodes (computers) and cores (ppn) which will be utilized in the job. Also the rest of code assumes such structure of allocated resources.

Working directory has to be created before sending pbs job into the queue. Input file should be in working directory or full path to input file has to be specified. Input file has to be defined by common LS-DYNA .k file which is attached to the LS-DYNA solver via parameter i=

ANSYS Fluent

ANSYS Fluent  software contains the broad physical modeling capabilities needed to model flow, turbulence, heat transfer, and reactions for industrial applications ranging from air flow over an aircraft wing to combustion in a furnace, from bubble columns to oil platforms, from blood flow to semiconductor manufacturing, and from clean room design to wastewater treatment plants. Special models that give the software the ability to model in-cylinder combustion, aeroacoustics, turbomachinery, and multiphase systems have served to broaden its reach.

1. Common way to run Fluent over pbs file

To run ANSYS Fluent in batch mode you can utilize/modify the default fluent.pbs script and execute it via the qsub command.

```
#!/bin/bash
#PBS -S /bin/bash
#PBS -l nodes=2:ppn=16
#PBS -q qprod
#PBS -N $USER-Fluent-Project
#PBS -A XX-YY-ZZ

#! Mail to user when job terminate or abort
#PBS -m ae

#!change the working directory (default is home directory)
#cd <working directory> (working directory must exists)
WORK_DIR="/scratch/$USER/work"
cd $WORK_DIR

echo Running on host `hostname`
echo Time is `date`
echo Directory is `pwd`
echo This jobs runs on the following processors:
echo `cat $PBS_NODEFILE`

#### Load ansys module so that we find the cfx5solve command
module load ansys

# Use following line to specify MPI for message-passing instead
NCORES=`wc -l $PBS_NODEFILE |awk '{print $1}'`

/ansys_inc/v145/fluent/bin/fluent 3d -t$NCORES -cnf=$PBS_NODEFILE -g -i fluent.jou
```

Header of the pbs file (above) is common and description can be find on this site. SVS FEM recommends to utilize sources by keywords: nodes, ppn. These keywords allows to address directly the number of nodes (computers) and cores (ppn) which will be utilized in the job. Also the rest of code assumes such structure of allocated resources.

Working directory has to be created before sending pbs job into the queue. Input file should be in working directory or full path to input file has to be specified. Input file has to be defined by common Fluent journal file which is attached to the Fluent solver via parameter -i fluent.jou

Journal file with definition of the input geometry and boundary conditions and defined process of

solution has e.g. the following structure:

```
/file/read-case aircraft_2m.cas.gz
/solve/init
init
/solve/iterate
10
/file/write-case-dat aircraft_2m-solution
/exit yes
```

The appropriate dimension of the problem has to be set by parameter (2d/3d).

2. Fast way to run Fluent from command line

```
fluent solver_version [FLUENT_options] -i journal_file -pbs
```

This syntax will start the ANSYS FLUENT job under PBS Professional using the qsub command in a batch manner. When resources are available, PBS Professional will start the job and return a job ID, usually in the form of *job_ID.hostname*. This job ID can then be used to query, control, or stop the job using standard PBS Professional commands, such as qstat or qdel. The job will be run out of the current working directory, and all output will be written to the file *fluent.o job_ID*.

3. Running Fluent via user's config file

The sample script uses a configuration file called *pbs_fluent.conf* if no command line arguments are present. This configuration file should be present in the directory from which the jobs are submitted (which is also the directory in which the jobs are executed). The following is an example of what the content of *pbs_fluent.conf* can be:

```
input="example_small.flin"
case="Small-1.65m.cas"
fluent_args="3d -pmyninet"
outfile="fluent_test.out"
mpp="true"
```

The following is an explanation of the parameters:

input is the name of the input file.

case is the name of the .cas file that the input file will utilize.

fluent_args are extra ANSYS FLUENT arguments. As shown in the previous example, you can specify the interconnect by using the -p interconnect command. The available interconnects include ethernet (the default), myrinet, infiniband, vendor, altix, and crayx. The MPI is selected automatically, based on the specified interconnect.

outfile is the name of the file to which the standard output will be sent.

mpp="true" will tell the job script to execute the job across multiple processors.

To run ANSYS Fluent in batch mode with user's config file you can utilize/modify the following script and execute it via the qsub command.

```
#!/bin/sh
#PBS -l nodes=2:ppn=4
#PBS -1 qprod
#PBS -N $USE-Fluent-Project
```

```

#PBS -A XX-YY-ZZ

cd $PBS_O_WORKDIR

#We assume that if they didn't specify arguments then they should use the
#config file if [ "xx${input}${case}${mpp}${fluent_args}zz" = "xxzz" ]; then
  if [ -f pbs_fluent.conf ]; then
    . pbs_fluent.conf
  else
    printf "No command line arguments specified, "
    printf "and no configuration file found.  Exiting n"
  fi
fi

#Augment the ANSYS FLUENT command line arguments case "$mpp" in
true)
  #MPI job execution scenario
  num_nodes='cat $PBS_NODEFILE | sort -u | wc -l'
  cpus='expr $num_nodes * $NCPUS'
  #Default arguments for mpp jobs, these should be changed to suit your
  #needs.
  fluent_args="-t${cpus} $fluent_args -cnf=$PBS_NODEFILE"
  ;;
*)
  #SMP case
  #Default arguments for smp jobs, should be adjusted to suit your
  #needs.
  fluent_args="-t$NCPUS $fluent_args"
  ;;
esac
#Default arguments for all jobs
fluent_args="-ssh -g -i $input $fluent_args"

echo "----- Going to start a fluent job with the following settings:
Input: $input
Case: $case
Output: $outfile
Fluent arguments: $fluent_args"

#run the solver
/ansys_inc/v145/fluent/bin/fluent $fluent_args > $outfile

```

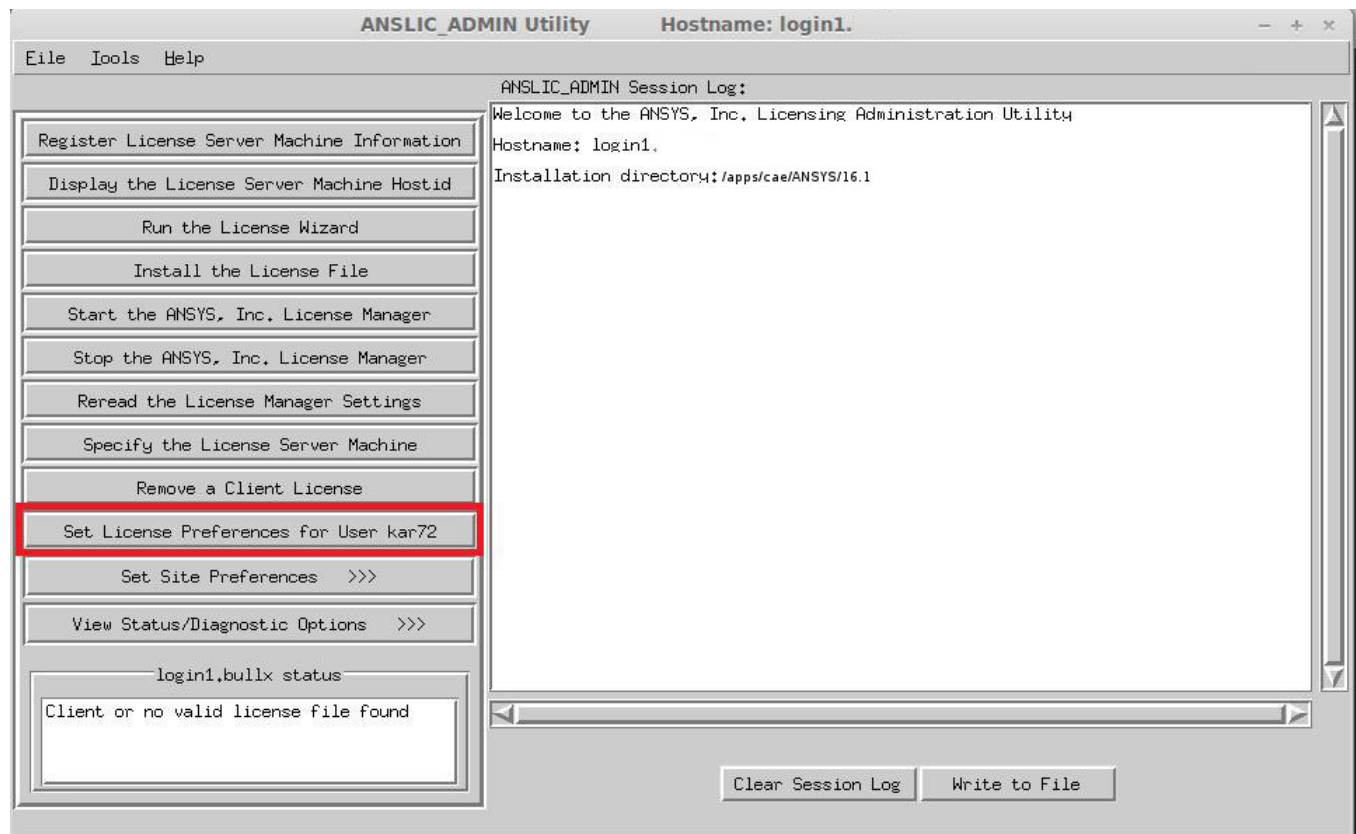
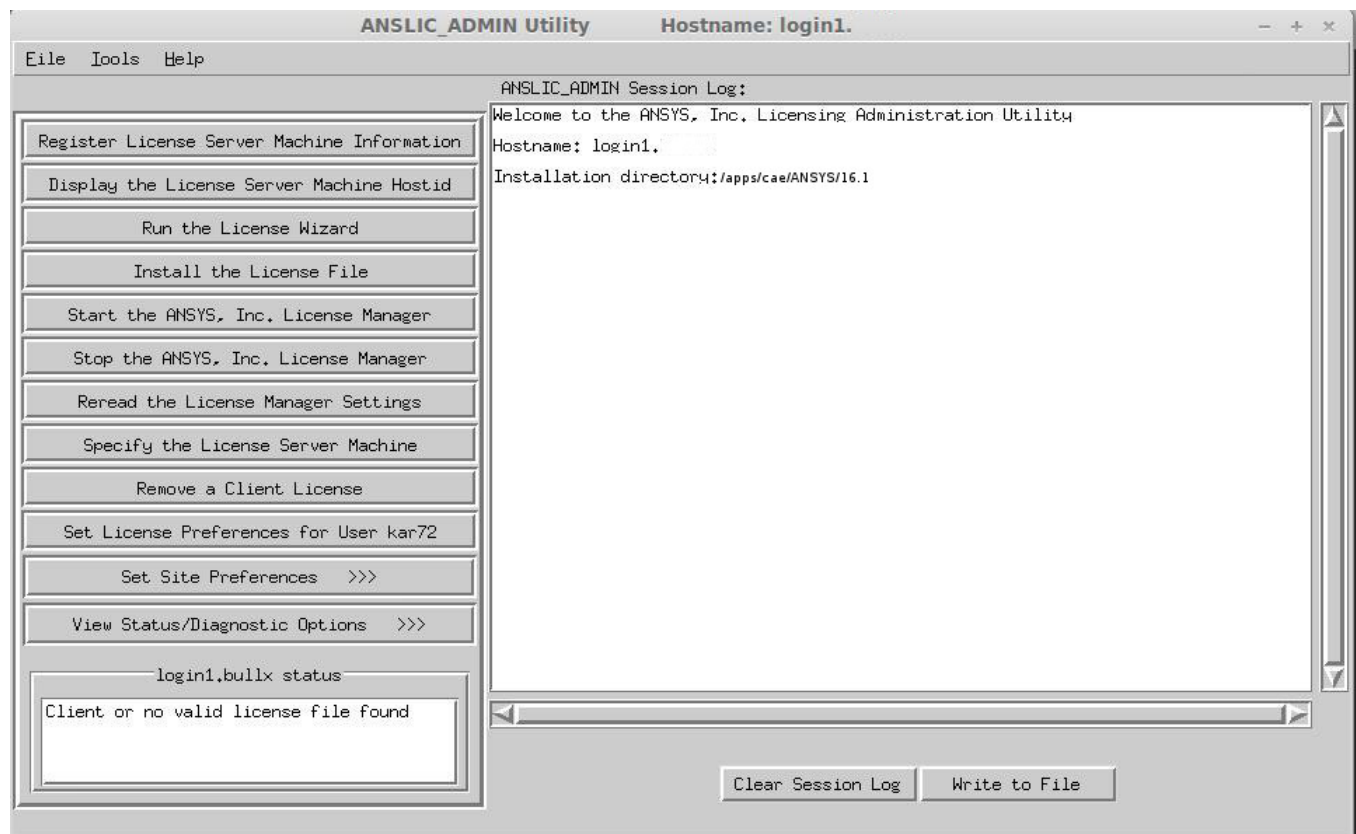
It runs the jobs out of the directory from which they are submitted (PBS_O_WORKDIR).

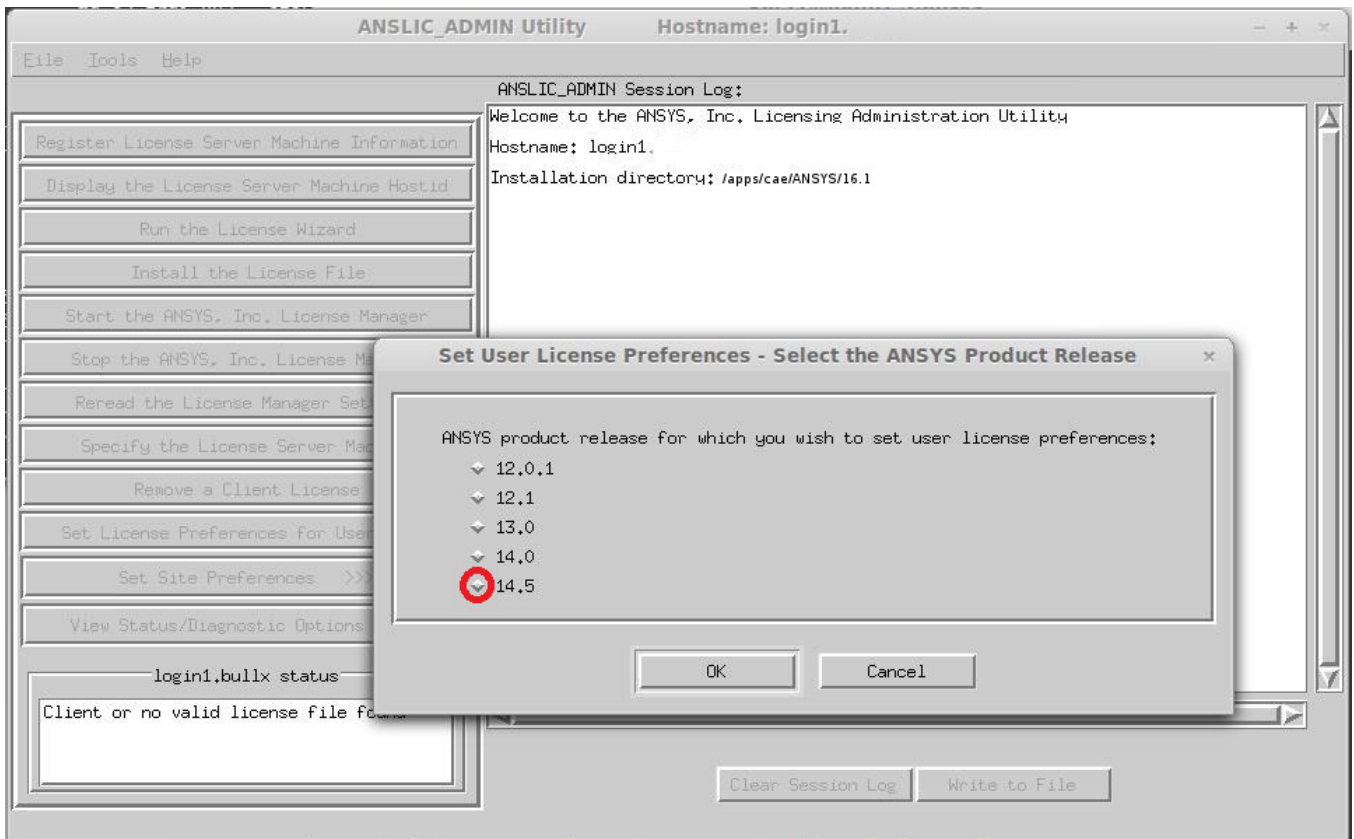
4. Running Fluent in parallel

Fluent could be run in parallel only under Academic Research license. To do so this ANSYS Academic Research license must be placed before ANSYS CFD license in user preferences. To make this change `anslic_admin` utility should be run

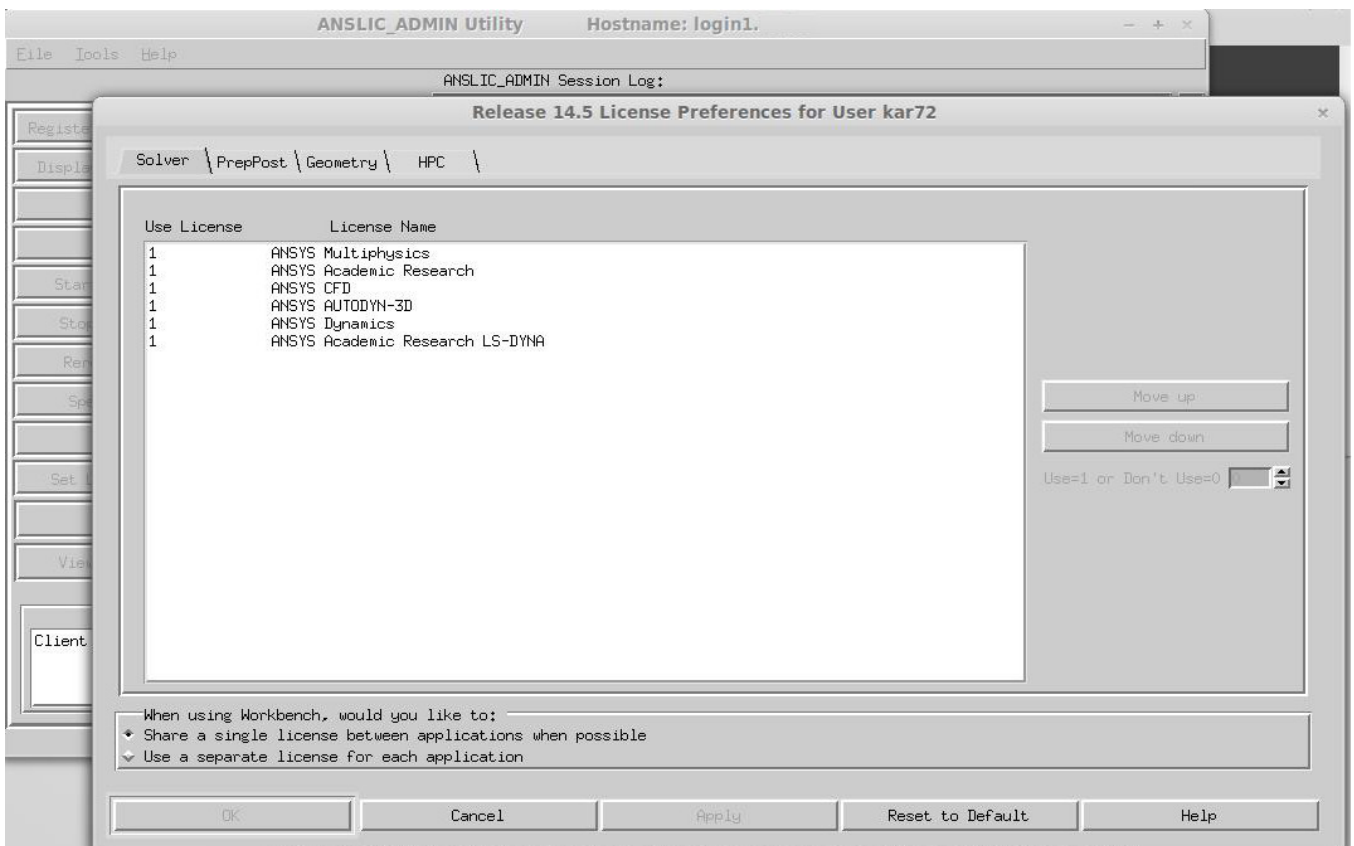
```
/ansys_inc/shared_les/licensing/lic_admin/anslic_admin
```

ANSLIC_ADMIN Utility will be run






ANSYS Academic Research license should be moved up to the top of the list.



ANSYS LS-DYNA

ANSYS LS-DYNA  software provides convenient and easy-to-use access to the technology-rich, time-tested explicit solver without the need to contend with the complex input requirements of this sophisticated program. Introduced in 1996, ANSYS LS-DYNA capabilities have helped customers in numerous industries to resolve highly intricate design issues. ANSYS Mechanical users have been able take advantage of complex explicit solutions for a long time utilizing the traditional ANSYS Parametric Design Language (APDL) environment. These explicit capabilities are available to ANSYS Workbench users as well. The Workbench platform is a powerful, comprehensive, easy-to-use environment for engineering simulation. CAD import from all sources, geometry cleanup, automatic meshing, solution, parametric optimization, result visualization and comprehensive report generation are all available within a single fully interactive modern graphical user environment.

To run ANSYS LS-DYNA in batch mode you can utilize/modify the default `ansysdyna.pbs` script and execute it via the `qsub` command.

```
#!/bin/bash
#PBS -l nodes=2:ppn=16
#PBS -q qprod
#PBS -N $USER-DYNA-Project
#PBS -A XX-YY-ZZ

#! Mail to user when job terminate or abort
#PBS -m ae

#!change the working directory (default is home directory)
#cd <working directory>
WORK_DIR="/scratch/$USER/work"
cd $WORK_DIR

echo Running on host `hostname`
echo Time is `date`
echo Directory is `pwd`
echo This jobs runs on the following processors:
echo `cat $PBS_NODEFILE`

#! Counts the number of processors
NPROCS=`wc -l < $PBS_NODEFILE`

echo This job has allocated $NPROCS nodes

module load ansys

#### Set number of processors per host listing
#### (set to 1 as $PBS_NODEFILE lists each node twice if :ppn=2)
procs_per_host=1
#### Create host list
hl=""
for host in `cat $PBS_NODEFILE`
do
    if [ "$hl" = "" ]
    then hl="$host:$procs_per_host"
    else hl="{hl}:$host:$procs_per_host"
```

```
fi
done
```


```
echo Machines: $hl
```

```
/ansys_inc/v145/ansys/bin/ansys145 -dis -lsdynampp i=input.k -machines $hl
```

Header of the pbs file (above) is common and description can be find on this site. SVS FEM [\[link\]](#) recommends to utilize sources by keywords: nodes, ppn. These keywords allows to address directly the number of nodes (computers) and cores (ppn) which will be utilized in the job. Also the rest of code assumes such structure of allocated resources.

Working directory has to be created before sending pbs job into the queue. Input file should be in working directory or full path to input file has to be specified. Input file has to be defined by common LS-DYNA **.k** file which is attached to the ansys solver via parameter i=

ANSYS MAPDL

ANSYS Multiphysics  software offers a comprehensive product solution for both multiphysics and single-physics analysis. The product includes structural, thermal, fluid and both high- and low-frequency electromagnetic analysis. The product also contains solutions for both direct and sequentially coupled physics problems including direct coupled-field elements and the ANSYS multi-field solver.

To run ANSYS MAPDL in batch mode you can utilize/modify the default mapdl.pbs script and execute it via the qsub command.

```
#!/bin/bash
#PBS -l nodes=2:ppn=16
#PBS -q qprod
#PBS -N $USER-ANSYS-Project
#PBS -A XX-YY-ZZ

#! Mail to user when job terminate or abort
#PBS -m ae

#!change the working directory (default is home directory)
#cd <working directory> (working directory must exists)
WORK_DIR="/scratch/$USER/work"
cd $WORK_DIR

echo Running on host `hostname`
echo Time is `date`
echo Directory is `pwd`
echo This jobs runs on the following processors:
echo `cat $PBS_NODEFILE`

module load ansys

#### Set number of processors per host listing
#### (set to 1 as $PBS_NODEFILE lists each node twice if :ppn=2)
procs_per_host=1
#### Create host list
hl=""
for host in `cat $PBS_NODEFILE`
do
    if [ "$hl" = "" ]
    then hl="$host:$procs_per_host"
    else hl="{hl}:$host:$procs_per_host"
    fi
done

echo Machines: $hl

#-i input.dat includes the input of analysis in APDL format
#-o file.out is output file from ansys where all text outputs will be redirected
#-p the name of license feature (aa_r=ANSYS Academic Research, ane3fl=Multiphysics(comm
/ansys_inc/v145/ansys/bin/ansys145 -b -dis -p aa_r -i input.dat -o file.out -machines $hl
```


Header of the pbs file (above) is common and description can be find on this site. SVS FEM 

recommends to utilize sources by keywords: nodes, ppn. These keywords allows to address directly the number of nodes (computers) and cores (ppn) which will be utilized in the job. Also the rest of code assumes such structure of allocated resources.

Working directory has to be created before sending pbs job into the queue. Input file should be in working directory or full path to input file has to be specified. Input file has to be defined by common APDL file which is attached to the ansys solver via parameter -i

License should be selected by parameter -p. Licensed products are the following: aa_r (ANSYS **Academic** Research), ane3fl (ANSYS Multiphysics)-**Commercial**, aa_r_dy (ANSYS **Academic** AUTODYN) More about licensing here

ANSYS CFX

ANSYS CFX  software is a high-performance, general purpose fluid dynamics program that has been applied to solve wide-ranging fluid flow problems for over 20 years. At the heart of ANSYS CFX is its advanced solver technology, the key to achieving reliable and accurate solutions quickly and robustly. The modern, highly parallelized solver is the foundation for an abundant choice of physical models to capture virtually any type of phenomena related to fluid flow. The solver and its many physical models are wrapped in a modern, intuitive, and flexible GUI and user environment, with extensive capabilities for customization and automation using session files, scripting and a powerful expression language.

To run ANSYS CFX in batch mode you can utilize/modify the default cfx.pbs script and execute it via the qsub command.

```
#!/bin/bash
#PBS -l nodes=2:ppn=16
#PBS -q qprod
#PBS -N $USER-CFX-Project
#PBS -A XX-YY-ZZ

#! Mail to user when job terminate or abort
#PBS -m ae

#!change the working directory (default is home directory)
#cd <working directory> (working directory must exists)
WORK_DIR="/scratch/$USER/work"
cd $WORK_DIR

echo Running on host `hostname`
echo Time is `date`
echo Directory is `pwd`
echo This jobs runs on the following processors:
echo `cat $PBS_NODEFILE`

module load ansys

#### Set number of processors per host listing
#### (set to 1 as $PBS_NODEFILE lists each node twice if :ppn=2)
procs_per_host=1
#### Create host list
hl=""
for host in `cat $PBS_NODEFILE`
do
    if [ "$hl" = "" ]
    then hl="$host:$procs_per_host"
    else hl="{hl}:$host:$procs_per_host"
    fi
done

echo Machines: $hl

#-dev input.def includes the input of CFX analysis in DEF format
#-P the name of preferred license feature (aa_r=ANSYS Academic Research, ane3fl=Multiphys
```

```
/ansys_inc/v145/CFX/bin/cfx5solve -def input.def -size 4 -size-ni 4x -part-large -start-m
```

Header of the pbs file (above) is common and description can be find on this site. SVS FEM recommends to utilize sources by keywords: nodes, ppn. These keywords allows to address directly the number of nodes (computers) and cores (ppn) which will be utilized in the job. Also the rest of code assumes such structure of allocated resources.

Working directory has to be created before sending pbs job into the queue. Input file should be in working directory or full path to input file has to be specified. >Input file has to be defined by common CFX def file which is attached to the cfx solver via parameter -def

License should be selected by parameter -P (Big letter **P**). Licensed products are the following: aa_r (ANSYS **Academic** Research), ane3fl (ANSYS Multiphysics)-**Commercial**. More about licensing [here](#)

Overview of ANSYS Products

SVS FEM  as **ANSYS Channel partner** for Czech Republic provided all ANSYS licenses for ANSELM cluster and supports of all ANSYS Products (Multiphysics, Mechanical, MAPDL, CFX, Fluent, Maxwell, LS-DYNA...) to IT staff and ANSYS users. If you are challenging to problem of ANSYS functionality contact please hotline@svsfem.cz

Anselm provides as commercial as academic variants. Academic variants are distinguished by “**Academic...**” word in the name of license or by two letter preposition “**aa__**” in the license feature name. Change of license is realized on command line respectively directly in user’s pbs file (see individual products). More about licensing here

To load the latest version of any ANSYS product (Mechanical, Fluent, CFX, MAPDL,...) load the module:

```
$ module load ansys
```

ANSYS supports interactive regime, but due to assumed solution of extremely difficult tasks it is not recommended.

If user needs to work in interactive regime we recommend to configure the RSM service on the client machine which allows to forward the solution to the Anselm directly from the client’s Workbench project (see ANSYS RSM service).

Java

Java on ANSELM

Java is available on Anselm cluster. Activate java by loading the java module

```
$ module load java
```

Note that the java module must be loaded on the compute nodes as well, in order to run java on compute nodes.

Check for java version and path

```
$ java -version
$ which java
```

With the module loaded, not only the runtime environment (JRE), but also the development environment (JDK) with the compiler is available.

```
$ javac -version
$ which javac
```

Java applications may use MPI for interprocess communication, in conjunction with OpenMPI. Read more on <http://www.open-mpi.org/faq/?category=java>. This functionality is currently not supported on Anselm cluster. In case you require the java interface to MPI, please contact Anselm support.

Score-P

Introduction

The Score-P measurement infrastructure [\[1\]](#) is a highly scalable and easy-to-use tool suite for profiling, event tracing, and online analysis of HPC applications.

Score-P can be used as an instrumentation tool for Scalasca.

Installed versions

There are currently two versions of Score-P version 1.2.6 modules installed on Anselm :

- scorep/1.2.3-gcc-openmpi, for usage with GNU Compiler and OpenMPI
- scorep/1.2.3-icc-mpi, for usage with Intel Compiler> and Intel MPI>.

Instrumentation

There are three ways to instrument your parallel applications in order to enable performance data collection:

1. Automated instrumentation using compiler
2. Manual instrumentation using API calls
3. Manual instrumentation using directives

Automated instrumentation

is the easiest method. Score-P will automatically add instrumentation to every routine entry and exit using compiler hooks, and will intercept MPI calls and OpenMP regions. This method might, however, produce a large number of data. If you want to focus on profiler a specific regions of your code, consider using the manual instrumentation methods. To use automated instrumentation, simply prepend scorep to your compilation command. For example, replace:

```
$ mpif90 -c foo.f90
$ mpif90 -c bar.f90
$ mpif90 -o myapp foo.o bar.o
```

with:

```
$ scorep mpif90 -c foo.f90
$ scorep mpif90 -c bar.f90
$ scorep mpif90 -o myapp foo.o bar.o
```

Usually your program is compiled using a Makefile or similar script, so it advisable to add the scorep command to your definition of variables CC, CXX, FCC etc.

It is important that scorep is prepended also to the linking command, in order to link with Score-P instrumentation libraries.

Manual instrumentation using API calls

To use this kind of instrumentation, use scorep with switch `-user`. You will then mark regions to be instrumented by inserting API calls.

An example in C/C++ :

```
#include <scorep/SCOREP_User.h>
void foo()
{
    SCOREP_USER_REGION_DEFINE( my_region_handle )
    // more declarations
    SCOREP_USER_REGION_BEGIN( my_region_handle, "foo", SCOREP_USER_REGION_TYPE_COMMON
    // do something
    SCOREP_USER_REGION_END( my_region_handle )
}
```

and Fortran :

```
#include "scorep/SCOREP_User.inc"
subroutine foo
    SCOREP_USER_REGION_DEFINE( my_region_handle )
    ! more declarations
    SCOREP_USER_REGION_BEGIN( my_region_handle, "foo", SCOREP_USER_REGION_TYPE_COMMON
    ! do something
    SCOREP_USER_REGION_END( my_region_handle )
end subroutine foo
```

Please refer to the documentation for description of the API [\[4\]](#).

Manual instrumentation using directives

This method uses POMP2 directives to mark regions to be instrumented. To use this method, use command `scorep -pomp`.

Example directives in C/C++ :

```
void foo(...)
{
    /* declarations */
    #pragma pomp inst begin(foo)
    ...
    if (<condition>)
    {
        #pragma pomp inst altend(foo)
        return;
    }
    ...
    #pragma pomp inst end(foo)
}
```

and in Fortran :

```
subroutine foo(...)
    !declarations
    !POMP$ INST BEGIN(foo)
    ...
    if (<condition>) then
    !POMP$ INST ALTEND(foo)
    return
    end if
```

```
...  
!POMP$ INST END(foo)  
end subroutine foo
```

The directives are ignored if the program is compiled without Score-P. Again, please refer to the documentation [for](#) a more elaborate description.

Allinea Forge (DDT,MAP)

Allinea Forge consist of two tools - debugger DDT and profiler MAP.

Allinea DDT, is a commercial debugger primarily for debugging parallel MPI or OpenMP programs. It also has a support for GPU (CUDA) and Intel Xeon Phi accelerators. DDT provides all the standard debugging features (stack trace, breakpoints, watches, view variables, threads etc.) for every thread running as part of your program, or for every process - even if these processes are distributed across a cluster using an MPI implementation.

Allinea MAP is a profiler for C/C++/Fortran HPC codes. It is designed for profiling parallel code, which uses pthreads, OpenMP or MPI.

License and Limitations for Anselm Users

On Anselm users can debug OpenMP or MPI code that runs up to 64 parallel processes. In case of debugging GPU or Xeon Phi accelerated codes the limit is 8 accelerators. These limitation means that:

- 1 user can debug up 64 processes, or
- 32 users can debug 2 processes, etc.

In case of debugging on accelerators:

- 1 user can debug on up to 8 accelerators, or
- 8 users can debug on single accelerator.

Compiling Code to run with DDT

Modules

Load all necessary modules to compile the code. For example:

```
$ module load intel
$ module load impi ... or ... module load openmpi/X.X.X-icc
```

Load the Allinea DDT module:

```
$ module load Forge
```

Compile the code:

```
$ mpicc -g -O0 -o test_debug test.c
```

```
$ mpif90 -g -O0 -o test_debug test.f
```

Compiler flags

Before debugging, you need to compile your code with theses flags:

!!! Note “Note” - **g** : Generates extra debugging information usable by GDB. -g3 includes even more debugging information. This option is available for GNU and INTEL C/C++ and Fortran compilers.

- ****O0**** : Suppress all optimizations.

Starting a Job with DDT

Be sure to log in with an X window forwarding enabled. This could mean using the -X in the ssh:

```
$ ssh -X username@anselm.it4i.cz
```

Other options is to access login node using VNC. Please see the detailed information on how to use graphic user interface on Anselm

From the login node an interactive session **with X windows forwarding** (-X option) can be started by following command:

```
$ qsub -I -X -A NONE-0-0 -q qexp -lselect=1:ncpus=16:mpiprocs=16,walltime=01:00:00
```

Then launch the debugger with the ddt command followed by the name of the executable to debug:

```
$ ddt test_debug
```

A submission window that appears have a prefilled path to the executable to debug. You can select the number of MPI processors and/or OpenMP threads on which to run and press run. Command line arguments to a program can be entered to the “Arguments” box.

To start the debugging directly without the submission window, user can specify the debugging and execution parameters from the command line. For example the number of MPI processes is set by option “-np 4”. Skipping the dialog is done by “-start” option. To see the list of the “ddt” command line parameters, run “ddt -help”.

```
ddt -start -np 4 ./hello_debug impi
```

Documentation

Users can find original User Guide after loading the DDT module:

```
$DDTPATH/doc/userguide.pdf
```

[1] Discipline, Magic, Inspiration and Science: Best Practice Debugging with Allinea DDT, Workshop conducted at LLNL by Allinea on May 10, 2013, link [🔗](#)

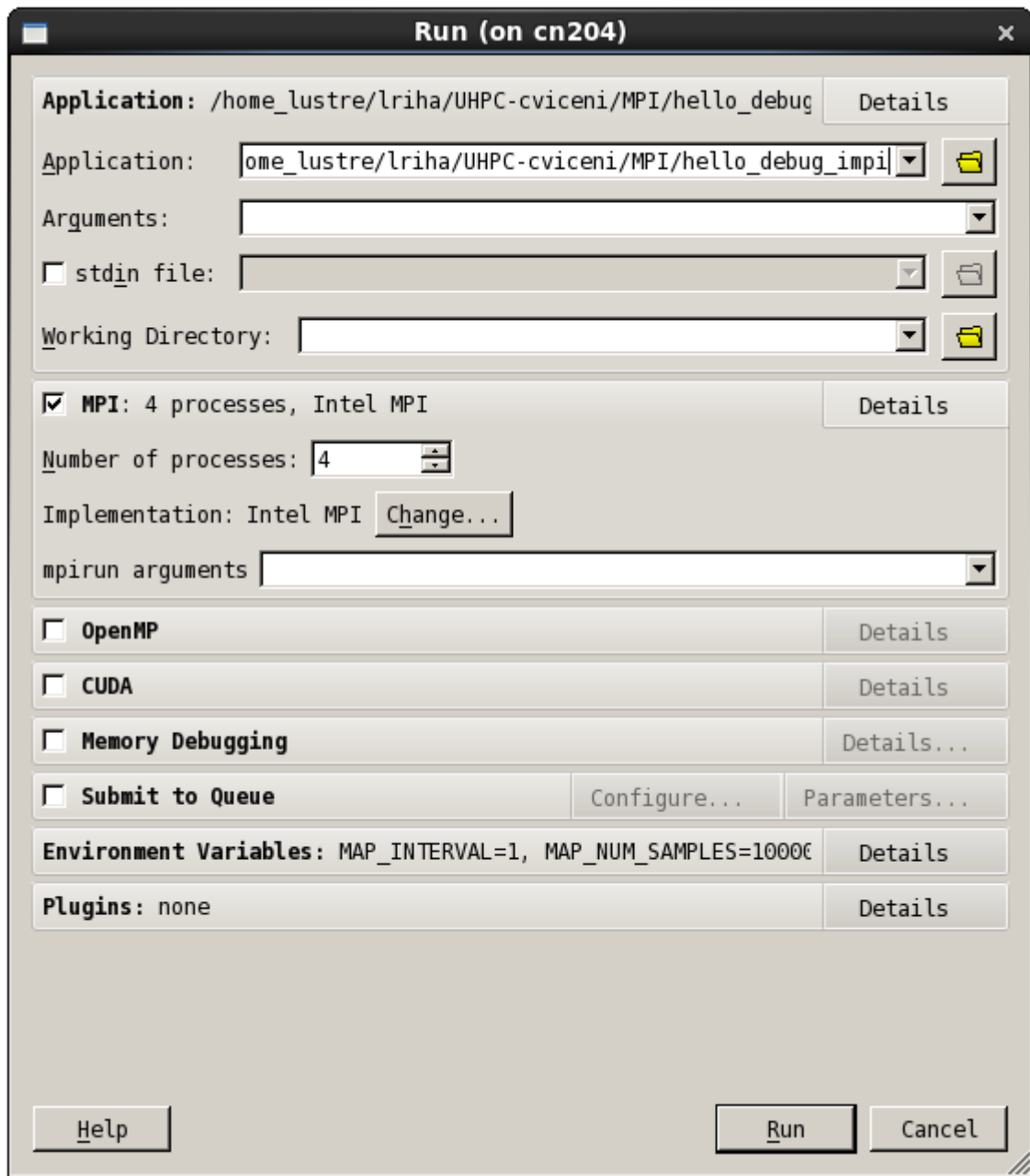


Figure 1:

Allinea Performance Reports

quick application profiling

Introduction

Allinea Performance Reports characterize the performance of HPC application runs. After executing your application through the tool, a synthetic HTML report is generated automatically, containing information about several metrics along with clear behavior statements and hints to help you improve the efficiency of your runs.

The Allinea Performance Reports is most useful in profiling MPI programs.

Our license is limited to 64 MPI processes.

Modules

Allinea Performance Reports version 6.0 is available

```
$ module load PerformanceReports/6.0
```

The module sets up environment variables, required for using the Allinea Performance Reports. This particular command loads the default module, which is performance reports version 4.2.

Usage

!!! Note “Note” Use the the perf-report wrapper on your (MPI) program.

Instead of running your MPI program the usual way, use the the perf report wrapper:

```
$ perf-report mpirun ./mympiprogram.x
```

The mpi program will run as usual. The perf-report creates two additional files, in *.txt* and *.html* format, containing the performance report. Note that demanding MPI codes should be run within the queue system.

Example

In this example, we will be profiling the mympiprogram.x MPI program, using Allinea performance reports. Assume that the code is compiled with intel compilers and linked against intel MPI library:

First, we allocate some nodes via the express queue:

```
$ qsub -q qexp -l select=2:ncpus=16:mpiprocs=16:ompthreads=1 -I
qsub: waiting for job 262197.dm2 to start
qsub: job 262197.dm2 ready
```

Then we load the modules and run the program the usual way:

```
$ module load intel impi allinea-perf-report/4.2
$ mpirun ./mympiprogram.x
```

Now lets profile the code:

```
$ perf-report mpirun ./mympiprogram.x
```

Performance report files `mympprog_32p*.txt`[🔗](#) and `mympprog_32p*.html`[🔗](#) were created. We can see that the code is very efficient on MPI and is CPU bounded.

PAPI

Introduction

Performance Application Programming Interface (PAPI) is a portable interface to access hardware performance counters (such as instruction counts and cache misses) found in most modern architectures. With the new component framework, PAPI is not limited only to CPU counters, but offers also components for CUDA, network, Infiniband etc.

PAPI provides two levels of interface - a simpler, high level interface and more detailed low level interface.

PAPI can be used with parallel as well as serial programs.

Usage

To use PAPI, load module papi:

```
$ module load papi
```

This will load the default version. Execute module avail papi for a list of installed versions.

Utilites

The bin directory of PAPI (which is automatically added to \$PATH upon loading the module) contains various utilites.

papi_avail

Prints which preset events are available on the current CPU. The third column indicated whether the preset event is available on the current CPU.

```
$ papi_avail
Available events and hardware information.
-----
PAPI Version : 5.3.2.0
Vendor string and code : GenuineIntel (1)
Model string and code : Intel(R) Xeon(R) CPU E5-2670 0 @ 2.60GHz (45)
CPU Revision : 7.000000
CPUID Info : Family: 6 Model: 45 Stepping: 7
CPU Max Megahertz : 2601
CPU Min Megahertz : 1200
Hdw Threads per core : 1
Cores per Socket : 8
Sockets : 2
NUMA Nodes : 2
CPUs per Node : 8
Total CPUs : 16
Running in a VM : no
Number Hardware Counters : 11
Max Multiplex Counters : 32
-----
```

Name	Code	Avail	Deriv	Description (Note)
PAPI_L1_DCM	0x80000000	Yes	No	Level 1 data cache misses
PAPI_L1_ICM	0x80000001	Yes	No	Level 1 instruction cache misses
PAPI_L2_DCM	0x80000002	Yes	Yes	Level 2 data cache misses
PAPI_L2_ICM	0x80000003	Yes	No	Level 2 instruction cache misses
PAPI_L3_DCM	0x80000004	No	No	Level 3 data cache misses
PAPI_L3_ICM	0x80000005	No	No	Level 3 instruction cache misses
PAPI_L1_TCM	0x80000006	Yes	Yes	Level 1 cache misses
PAPI_L2_TCM	0x80000007	Yes	No	Level 2 cache misses
PAPI_L3_TCM	0x80000008	Yes	No	Level 3 cache misses
....				

papi__native__avail

Prints which native events are available on the current CPU.

papi__cost

Measures the cost (in cycles) of basic PAPI operations.

papi__mem__info

Prints information about the memory architecture of the current CPU.

PAPI API

PAPI provides two kinds of events:

- **Preset events** is a set of predefined common CPU events, standardized across platforms.
- **Native events** is a set of all events supported by the current hardware. This is a larger set of features than preset. For other components than CPU, only native events are usually available.

To use PAPI in your application, you need to link the appropriate include file.

- papi.h for C
- f77papi.h for Fortran 77
- f90papi.h for Fortran 90
- fpapi.h for Fortran with preprocessor

The include path is automatically added by papi module to \$INCLUDE.

High level API

Please refer to http://icl.cs.utk.edu/projects/papi/wiki/PAPIC:High_Level for a description of the High level API.

Low level API

Please refer to http://icl.cs.utk.edu/projects/papi/wiki/PAPIC:Low_Level for a description of the Low level API.

Timers

PAPI provides the most accurate timers the platform can support. See <http://icl.cs.utk.edu/projects/papi/wiki/PAPIC:Timers>

System information

PAPI can be used to query some system information, such as CPU name and MHz. See http://icl.cs.utk.edu/projects/papi/wiki/PAPIC:System_Information

Example

The following example prints MFLOPS rate of a naive matrix-matrix multiplication:

```
#include <stdlib.h>
#include <stdio.h>
#include "papi.h"
#define SIZE 1000

int main(int argc, char **argv) {
    float matrixa[SIZE][SIZE], matrixb[SIZE][SIZE], mresult[SIZE][SIZE];
    float real_time, proc_time, mflops;
    long long flpins;
    int retval;
    int i,j,k;

    /* Initialize the Matrix arrays */
    for ( i=0; i<SIZE*SIZE; i++){
        mresult[0][i] = 0.0;
        matrixa[0][i] = matrixb[0][i] = rand()*(float)1.1;
    }

    /* Setup PAPI library and begin collecting data from the counters */
    if((retval=PAPI_flops( &real_time, &proc_time, &flpins, &mflops))<PAPI_OK)
        printf("Error!");

    /* A naive Matrix-Matrix multiplication */
    for (i=0;i<SIZE;i++)
        for(j=0;j<SIZE;j++)
            for(k=0;k<SIZE;k++)
                mresult[i][j]=mresult[i][j] + matrixa[i][k]*matrixb[k][j];

    /* Collect the data into the variables passed in */
    if((retval=PAPI_flops( &real_time, &proc_time, &flpins, &mflops))<PAPI_OK)
        printf("Error!");
```

```

    printf("Real_time:t%fnProc_time:t%fnTotal flpins:t%lldnMFLOPS:tt%fn", real_time, pro
    PAPI_shutdown();
    return 0;
}

```

Now compile and run the example :

```

$ gcc matrix.c -o matrix -lpapi
$ ./matrix
Real_time: 8.852785
Proc_time: 8.850000
Total flpins: 6012390908
MFLOPS: 679.366211

```

Let's try with optimizations enabled :

```

$ gcc -O3 matrix.c -o matrix -lpapi
$ ./matrix
Real_time: 0.000020
Proc_time: 0.000000
Total flpins: 6
MFLOPS: inf

```

Now we see a seemingly strange result - the multiplication took no time and only 6 floating point instructions were issued. This is because the compiler optimizations have completely removed the multiplication loop, as the result is actually not used anywhere in the program. We can fix this by adding some “dummy” code at the end of the Matrix-Matrix multiplication routine :

```

for (i=0; i<SIZE;i++)
    for (j=0; j<SIZE; j++)
        if (mresult[i][j] == -1.0) printf("x");

```

Now the compiler won't remove the multiplication loop. (However it is still not that smart to see that the result won't ever be negative). Now run the code again:

```

$ gcc -O3 matrix.c -o matrix -lpapi
$ ./matrix
Real_time: 8.795956
Proc_time: 8.790000
Total flpins: 18700983160
MFLOPS: 2127.529297

```

Intel Xeon Phi

!!! Note “Note” PAPI currently supports only a subset of counters on the Intel Xeon Phi processor compared to Intel Xeon, for example the floating point operations counter is missing.

To use PAPI in Intel Xeon Phi native applications, you need to load module with ” -mic” suffix, for example ” papi/5.3.2-mic” :

```

$ module load papi/5.3.2-mic

```

Then, compile your application in the following way:

```

$ module load intel
$ icc -mmic -Wl,-rpath,/apps/intel/composer_xe_2013.5.192/compiler/lib/mic matrix-mic

```

To execute the application on MIC, you need to manually set LD_LIBRARY_PATH:


```
$ qsub -q qmic -A NONE-0-0 -I
$ ssh mic0
$ export LD_LIBRARY_PATH=/apps/tools/papi/5.4.0-mic/lib/
$ ./matrix-mic
```

Alternatively, you can link PAPI statically (-static flag), then LD_LIBRARY_PATH does not need to be set.




You can also execute the PAPI tools on MIC :

```
$ /apps/tools/papi/5.4.0-mic/bin/papi_native_avail
```

To use PAPI in offload mode, you need to provide both host and MIC versions of PAPI:

```
$ module load papi/5.4.0
$ icc matrix-offload.c -o matrix-offload -offload-option,mic,compiler,"-L$PAPI_HOME-mic/lib"
```

References

1. <http://icl.cs.utk.edu/papi/>  Main project page
2. http://icl.cs.utk.edu/projects/papi/wiki/Main_Page  Wiki
3. <http://icl.cs.utk.edu/papi/docs/>  API Documentation

CUBE

Introduction

CUBE is a graphical performance report explorer for displaying data from Score-P and Scalasca (and other compatible tools). The name comes from the fact that it displays performance data in a three-dimensions :

- **performance metric**, where a number of metrics are available, such as communication time or cache misses,
- **call path**, which contains the call tree of your program
- **system resource**, which contains system's nodes, processes and threads, depending on the parallel programming model.

Each dimension is organized in a tree, for example the time performance metric is divided into Execution time and Overhead time, call path dimension is organized by files and routines in your source code etc.

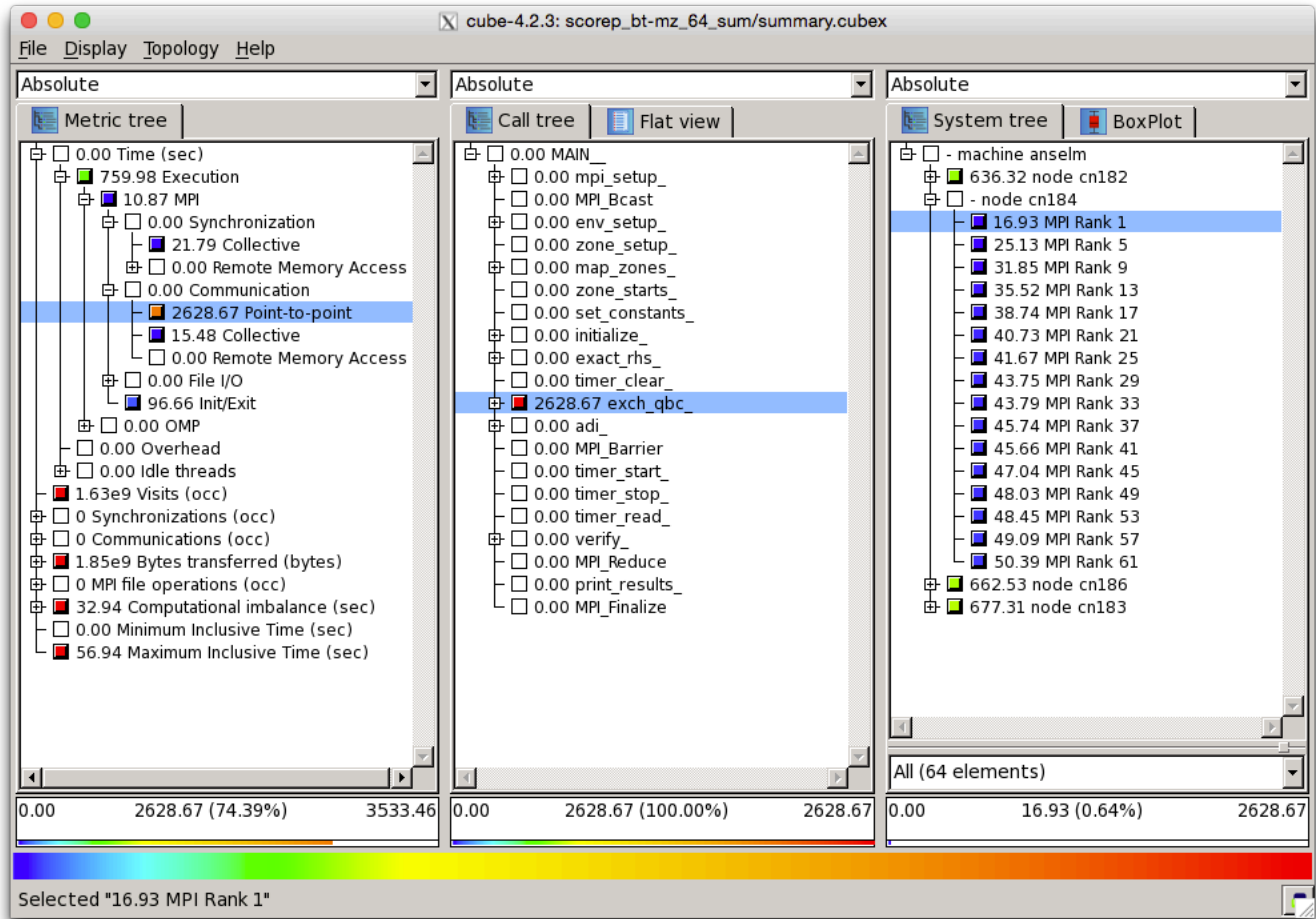


Figure 1:

Figure 1. Screenshot of CUBE displaying data from Scalasca.

Each node in the tree is colored by severity (the color scheme is displayed at the bottom of the window, ranging from the least severe blue to the most severe being red). For example in Figure 1, we can see that most of the point-to-point MPI communication happens in routine `exch_qbc`, colored red.

Installed versions

Currently, there are two versions of CUBE 4.2.3 available as modules :

- cube/4.2.3-gcc, compiled with GCC
- cube/4.2.3-icc, compiled with Intel compiler

Usage

CUBE is a graphical application. Refer to Graphical User Interface documentation for a list of methods to launch graphical applications on Anselm.

!!! Note “Note” Analyzing large data sets can consume large amount of CPU and RAM. Do not perform large analysis on login nodes.

After loading the appropriate module, simply launch cube command, or alternatively you can use scalasca -examine command to launch the GUI. Note that for Scalasca datasets, if you do not analyze the data with scalasca -examine before to opening them with CUBE, not all performance data will be available.

References 1. <http://www.scalasca.org/software/cube-4.x/download.html> 

Debuggers and profilers summary

Introduction

We provide state of the art programmes and tools to develop, profile and debug HPC codes at IT4Innovations. On these pages, we provide an overview of the profiling and debugging tools available on Anslem at IT4I.

Intel debugger

The intel debugger version 13.0 is available, via module intel. The debugger works for applications compiled with C and C++ compiler and the ifort fortran 77/90/95 compiler. The debugger provides java GUI environment. Use X display for running the GUI.

```
$ module load intel
$ idb
```

Read more at the Intel Debugger page.

Allinea Forge (DDT/MAP)

Allinea DDT, is a commercial debugger primarily for debugging parallel MPI or OpenMP programs. It also has a support for GPU (CUDA) and Intel Xeon Phi accelerators. DDT provides all the standard debugging features (stack trace, breakpoints, watches, view variables, threads etc.) for every thread running as part of your program, or for every process even if these processes are distributed across a cluster using an MPI implementation.

```
$ module load Forge
$ forge
```

Read more at the Allinea DDT page.

Allinea Performance Reports

Allinea Performance Reports characterize the performance of HPC application runs. After executing your application through the tool, a synthetic HTML report is generated automatically, containing information about several metrics along with clear behavior statements and hints to help you improve the efficiency of your runs. Our license is limited to 64 MPI processes.

```
$ module load PerformanceReports/6.0
$ perf-report mpirun -n 64 ./my_application argument01 argument02
```

Read more at the Allinea Performance Reports page.

RougeWave Totalview

TotalView is a source- and machine-level debugger for multi-process, multi-threaded programs. Its wide range of tools provides ways to analyze, organize, and test programs, making it easy to isolate and identify problems in individual threads and processes in programs of great complexity.

```
$ module load totalview
$ totalview
```

Read more at the Totalview page.

Vampir trace analyzer

Vampir is a GUI trace analyzer for traces in OTF format.

```
$ module load Vampir/8.5.0  
$ vampir
```

Read more at the Vampir page.

Intel Performance Counter Monitor

Introduction

Intel PCM (Performance Counter Monitor) is a tool to monitor performance hardware counters on Intel® processors, similar to PAPI. The difference between PCM and PAPI is that PCM supports only Intel hardware, but PCM can monitor also uncore metrics, like memory controllers and >QuickPath Interconnect links.

Installed version

Currently installed version 2.6. To load the module, issue:

```
$ module load intelpcm
```

Command line tools

PCM provides a set of tools to monitor system/or application.

pcm-memory

Measures memory bandwidth of your application or the whole system. Usage:

```
$ pcm-memory.x <delay>| [external_program parameters]
```

Specify either a delay of updates in seconds or an external program to monitor. If you get an error about PMU in use, respond “y” and relaunch the program.

Sample output:

Socket 0				Socket 1			
Memory Performance Monitoring				Memory Performance Monitoring			
Mem Ch 0: Reads (MB/s):	2.44			Mem Ch 0: Reads (MB/s):	0.26		
Writes(MB/s):	2.16			Writes(MB/s):	0.08		
Mem Ch 1: Reads (MB/s):	0.35			Mem Ch 1: Reads (MB/s):	0.78		
Writes(MB/s):	0.13			Writes(MB/s):	0.65		
Mem Ch 2: Reads (MB/s):	0.32			Mem Ch 2: Reads (MB/s):	0.21		
Writes(MB/s):	0.12			Writes(MB/s):	0.07		
Mem Ch 3: Reads (MB/s):	0.36			Mem Ch 3: Reads (MB/s):	0.20		
Writes(MB/s):	0.13			Writes(MB/s):	0.07		
NODE0 Mem Read (MB/s):	3.47			NODE1 Mem Read (MB/s):	1.45		
NODE0 Mem Write (MB/s):	2.55			NODE1 Mem Write (MB/s):	0.88		
NODE0 P. Write (T/s) :	31506			NODE1 P. Write (T/s):	9099		
NODE0 Memory (MB/s):	6.02			NODE1 Memory (MB/s):	2.33		
System Read Throughput(MB/s):				4.93			
System Write Throughput(MB/s):				3.43			

```
--                System Memory Throughput(MB/s):                8.35                --
-----|-----
```

pcm-msr

Command pcm-msr.x can be used to read/write model specific registers of the CPU.

pcm-numa

NUMA monitoring utility does not work on Anselm.

pcm-pcie

Can be used to monitor PCI Express bandwidth. Usage: pcm-pcie.x <delay>

pcm-power

Displays energy usage and thermal headroom for CPU and DRAM sockets. Usage: pcm-power.x <delay> | <external program>

pcm

This command provides an overview of performance counters and memory usage. Usage: pcm.x <delay> | <external program>

Sample output :

```
$ pcm.x ./matrix
```

```
Intel(r) Performance Counter Monitor V2.6 (2013-11-04 13:43:31 +0100 ID=db05e43)
```

```
Copyright (c) 2009-2013 Intel Corporation
```

```
Number of physical cores: 16
```

```
Number of logical cores: 16
```

```
Threads (logical cores) per physical core: 1
```

```
Num sockets: 2
```

```
Core PMU (perfmon) version: 3
```

```
Number of core PMU generic (programmable) counters: 8
```

```
Width of generic (programmable) counters: 48 bits
```

```
Number of core PMU fixed counters: 3
```

```
Width of fixed counters: 48 bits
```

```
Nominal core frequency: 2400000000 Hz
```

```
Package thermal spec power: 115 Watt; Package minimum power: 51 Watt; Package maximum power: 115 Watt
```

```
Socket 0: 1 memory controllers detected with total number of 4 channels. 2 QPI ports
```

```
Socket 1: 1 memory controllers detected with total number of 4 channels. 2 QPI ports
```

```
Number of PCM instances: 2
```

```
Max QPI link speed: 16.0 GBytes/second (8.0 GT/second)
```

```
Detected Intel(R) Xeon(R) CPU E5-2665 0 @ 2.40GHz "Intel(r) microarchitecture codenam
```

Executing "./matrix" command:

Exit code: 0

EXEC : instructions per nominal CPU cycle
IPC : instructions per CPU cycle
FREQ : relation to nominal CPU frequency='unhalted clock ticks'/'invariant timer ticks'
AFREQ : relation to nominal CPU frequency while in active state (not in power-saving state)
L3MISS: L3 cache misses
L2MISS: L2 cache misses (including other core's L2 cache *hits*)
L3HIT : L3 cache hit ratio (0.00-1.00)
L2HIT : L2 cache hit ratio (0.00-1.00)
L3CLK : ratio of CPU cycles lost due to L3 cache misses (0.00-1.00), in some cases core's L3 cache is not used
L2CLK : ratio of CPU cycles lost due to missing L2 cache but still hitting L3 cache
READ : bytes read from memory controller (in GBytes)
WRITE : bytes written to memory controller (in GBytes)
TEMP : Temperature reading in 1 degree Celsius relative to the TjMax temperature (in degrees Celsius)

Core (SKT)		EXEC	IPC	FREQ	AFREQ	L3MISS	L2MISS	L3HIT	L2HIT	L3CLK	L2CLK
0	0	0.00	0.64	0.01	0.80	5592	11 K	0.49	0.13	0.32	
1	0	0.00	0.18	0.00	0.69	3086	5552	0.44	0.07	0.48	
2	0	0.00	0.23	0.00	0.81	300	562	0.47	0.06	0.43	
3	0	0.00	0.21	0.00	0.99	437	862	0.49	0.06	0.44	
4	0	0.00	0.23	0.00	0.93	293	559	0.48	0.07	0.42	
5	0	0.00	0.21	0.00	1.00	423	849	0.50	0.06	0.43	
6	0	0.00	0.23	0.00	0.94	285	558	0.49	0.06	0.41	
7	0	0.00	0.18	0.00	0.81	674	1130	0.40	0.05	0.53	
8	1	0.00	0.47	0.01	1.26	6371	13 K	0.51	0.35	0.31	
9	1	2.30	1.80	1.28	1.29	179 K	15 M	0.99	0.59	0.04	
10	1	0.00	0.22	0.00	1.26	315	570	0.45	0.06	0.43	
11	1	0.00	0.23	0.00	0.74	321	579	0.45	0.05	0.45	
12	1	0.00	0.22	0.00	1.25	305	570	0.46	0.05	0.42	
13	1	0.00	0.22	0.00	1.26	336	581	0.42	0.04	0.44	
14	1	0.00	0.22	0.00	1.25	314	565	0.44	0.06	0.43	
15	1	0.00	0.29	0.00	1.19	2815	6926	0.59	0.39	0.29	
<hr/>											
SKT	0	0.00	0.46	0.00	0.79	11 K	21 K	0.47	0.10	0.38	
SKT	1	0.29	1.79	0.16	1.29	190 K	15 M	0.99	0.59	0.05	
<hr/>											
TOTAL	*	0.14	1.78	0.08	1.28	201 K	15 M	0.99	0.59	0.05	

Instructions retired: 1345 M ; Active cycles: 755 M ; Time (TSC): 582 Mticks ; Core 0 temperature: 45.00 C ; Core 1 temperature: 45.00 C ; Core 2 temperature: 45.00 C ; Core 3 temperature: 45.00 C ; Core 4 temperature: 45.00 C ; Core 5 temperature: 45.00 C ; Core 6 temperature: 45.00 C ; Core 7 temperature: 45.00 C ; Core 8 temperature: 45.00 C ; Core 9 temperature: 45.00 C ; Core 10 temperature: 45.00 C ; Core 11 temperature: 45.00 C ; Core 12 temperature: 45.00 C ; Core 13 temperature: 45.00 C ; Core 14 temperature: 45.00 C ; Core 15 temperature: 45.00 C ;

C1 core residency: 0.14 %; C3 core residency: 0.20 %; C6 core residency: 0.00 %; C7 core residency: 0.00 %; C2 package residency: 48.81 %; C3 package residency: 0.00 %; C6 package residency: 0.00 %; C7 package residency: 0.00 %;

PHYSICAL CORE IPC : 1.78 => corresponds to 44.50 % utilization for core 0
Instructions per nominal CPU cycle: 0.14 => corresponds to 3.60 % core utilization overall

Intel(r) QPI data traffic estimation in bytes (data traffic coming to CPU/socket through QPI interface)

		QPI0	QPI1		QPI0	QPI1
SKT	0	0	0		0%	0%
SKT	1	0	0		0%	0%

Total QPI incoming data traffic: 0 QPI data traffic/Memory controller traffic

Intel(r) QPI traffic estimation in bytes (data and non-data traffic outgoing from CPU)

		QPI0	QPI1		QPI0	QPI1
SKT	0	0	0		0%	0%
SKT	1	0	0		0%	0%

Total QPI outgoing data and non-data traffic: 0

SKT 0 package consumed 4.06 Joules

SKT 1 package consumed 9.40 Joules

TOTAL: 13.46 Joules

SKT 0 DIMMs consumed 4.18 Joules

SKT 1 DIMMs consumed 4.28 Joules

TOTAL: 8.47 Joules

Cleaning up

pcm-sensor

Can be used as a sensor for ksysguard GUI, which is currently not installed on Anselm.

API

In a similar fashion to PAPI, PCM provides a C++ API to access the performance counter from within your application. Refer to the doxygen documentation [\[1\]](#) for details of the API.

!!! Note “Note” Due to security limitations, using PCM API to monitor your applications is currently not possible on Anselm. (The application must be run as root user)

Sample program using the API :

```
#include <stdlib.h>
#include <stdio.h>
#include "cpucounters.h"

#define SIZE 1000

using namespace std;
```

```

int main(int argc, char **argv) {
    float matrixa[SIZE][SIZE], matrixb[SIZE][SIZE], mresult[SIZE][SIZE];
    float real_time, proc_time, mflops;
    long long flpins;
    int retval;
    int i,j,k;

    PCM * m = PCM::getInstance();

    if (m->program() != PCM::Success) return 1;

    SystemCounterState before_sstate = getSystemCounterState();

    /* Initialize the Matrix arrays */
    for ( i=0; i<SIZE*SIZE; i++ ){
        mresult[0][i] = 0.0;
        matrixa[0][i] = matrixb[0][i] = rand()*(float)1.1; }

    /* A naive Matrix-Matrix multiplication */
    for (i=0;i<SIZE;i++)
        for(j=0;j<SIZE;j++)
            for(k=0;k<SIZE;k++)
                mresult[i][j]=mresult[i][j] + matrixa[i][k]*matrixb[k][j];

    SystemCounterState after_sstate = getSystemCounterState();

    cout << "Instructions per clock:" << getIPC(before_sstate,after_sstate)
    << "L3 cache hit ratio:" << getL3CacheHitRatio(before_sstate,after_sstate)
    << "Bytes read:" << getBytesReadFromMC(before_sstate,after_sstate);

    for (i=0; i<SIZE;i++)
        for (j=0; j<SIZE; j++)
            if (mresult[i][j] == -1) printf("x");

    return 0;
}

```

Compile it with :

```
$ icc matrix.cpp -o matrix -lpthread -lpcm
```

Sample output:

```

$ ./matrix
Number of physical cores: 16
Number of logical cores: 16
Threads (logical cores) per physical core: 1
Num sockets: 2
Core PMU (perfmon) version: 3
Number of core PMU generic (programmable) counters: 8
Width of generic (programmable) counters: 48 bits
Number of core PMU fixed counters: 3
Width of fixed counters: 48 bits
Nominal core frequency: 2400000000 Hz

```

Package thermal spec power: 115 Watt; **Package** minimum power: 51 Watt; **Package** maximum power: 115 Watt
Socket 0: 1 memory controllers detected with total number of 4 channels. 2 QPI ports detected
Socket 1: 1 memory controllers detected with total number of 4 channels. 2 QPI ports detected
Number of PCM instances: 2
Max QPI link speed: 16.0 GBytes/second (8.0 GT/second)
Instructions per clock:1.7
L3 cache hit ratio:1.0
Bytes read:12513408

References

1. <https://software.intel.com/en-us/articles/intel-performance-counter-monitor-a-better-way-to-measure-performance>
2. <https://software.intel.com/sites/default/files/m/3/2/2/xeon-e5-2600-uncore-guide.pdf> Intel® Xeon® Processor E5-2600 Product Family Uncore Performance Monitoring Guide.
3. <http://intel-pcm-api-documentation.github.io/classPCM.html> API Documentation

Vampir

Vampir is a commercial trace analysis and visualisation tool. It can work with traces in OTF and OTF2 formats. It does not have the functionality to collect traces, you need to use a trace collection tool (such as Score-P) first to collect the traces.

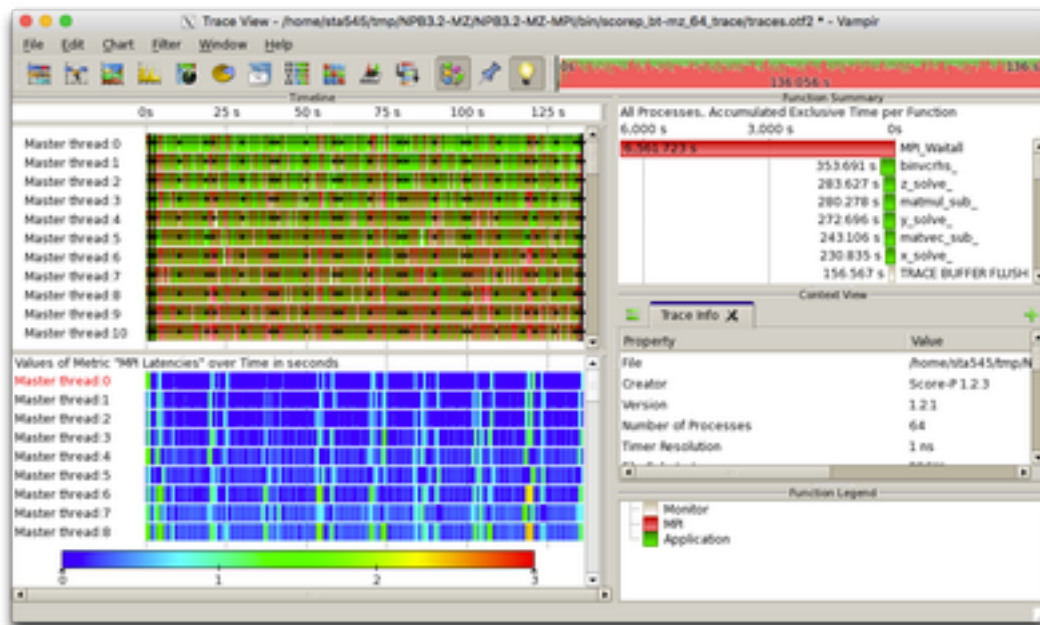


Figure 1:

Installed versions

Version 8.5.0 is currently installed as module Vampir/8.5.0 :

```
$ module load Vampir/8.5.0  
$ vampir &
```

User manual


You can find the detailed user manual in PDF format in `$EBROOTVAMPIR/doc/vampir-manual.pdf`

References

[1]. <https://www.vampir.eu>

Scalasca

Introduction

Scalasca  is a software tool that supports the performance optimization of parallel programs by measuring and analyzing their runtime behavior. The analysis identifies potential performance bottlenecks – in particular those concerning communication and synchronization – and offers guidance in exploring their causes.

Scalasca supports profiling of MPI, OpenMP and hybrid MPI+OpenMP applications.

Installed versions

There are currently two versions of Scalasca 2.0 modules installed on Anselm:

- scalasca2/2.0-gcc-openmpi, for usage with GNU Compiler and OpenMPI,
- scalasca2/2.0-icc-impi, for usage with Intel Compiler and Intel MPI.

Usage

Profiling a parallel application with Scalasca consists of three steps:

1. Instrumentation, compiling the application such way, that the profiling data can be generated.
2. Runtime measurement, running the application with the Scalasca profiler to collect performance data.
3. Analysis of reports

Instrumentation

Instrumentation via ”scalasca -instrument” is discouraged. Use Score-P instrumentation.

Runtime measurement

After the application is instrumented, runtime measurement can be performed with the ”scalasca -analyze” command. The syntax is:

```
scalasca -analyze [scalasca options] [launcher] [launcher options] [program] [program options]
```

An example :

```
$ scalasca -analyze mpirun -np 4 ./mympiprogram
```

Some notable Scalsca options are:

-t Enable trace data collection. By default, only summary data are collected. -e <directory> Specify a directory to save the collected data to. By default, Scalasca saves the data to a directory with prefix scorep_, followed by name of the executable and launch configuration.

!!! Note “Note” Scalasca can generate a huge amount of data, especially if tracing is enabled. Please consider saving the data to a scratch directory.

Analysis of reports

For the analysis, you must have Score-P and CUBE modules loaded. The analysis is done in two steps, first, the data is preprocessed and then CUBE GUI tool is launched.

To launch the analysis, run :

```
scalasca -examine [options] <experiment_directory>
```

If you do not wish to launch the GUI tool, use the “-s” option :

```
scalasca -examine -s <experiment_directory>
```

Alternatively you can open CUBE and load the data directly from here. Keep in mind that in that case the preprocessing is not done and not all metrics will be shown in the viewer.

Refer to CUBE documentation on usage of the GUI viewer.

References

1. <http://www.scalasca.org/> 

Valgrind

Valgrind is a tool for memory debugging and profiling.

About Valgrind

Valgrind is an open-source tool, used mainly for debuggig memory-related problems, such as memory leaks, use of uninitialized memory etc. in C/C++ applications. The toolchain was however extended over time with more functionality, such as debugging of threaded applications, cache profiling, not limited only to C/C++.

Valgind is an extremely useful tool for debugging memory errors such as off-by-one [☞](#). Valgrind uses a virtual machine and dynamic recompilation of binary code, because of that, you can expect that programs being debugged by Valgrind run 5-100 times slower.

The main tools available in Valgrind are :

- **Memcheck**, the original, must used and default tool. Verifies memory access in you program and can detect use of unitialized memory, out of bounds memory access, memory leaks, double free, etc.
- **Massif**, a heap profiler.
- **Hellgrind** and **DRD** can detect race conditions in multi-threaded applications.
- **Cachegrind**, a cache profiler.
- **Callgrind**, a callgraph analyzer.
- For a full list and detailed documentation, please refer to the official Valgrind documentation [☞](#).

Installed versions

There are two versions of Valgrind available on Anselm.

- Version 3.6.0, installed by operating system vendor in /usr/bin/valgrind. This version is available by default, without the need to load any module. This version however does not provide additional MPI support.
- Version 3.9.0 with support for Intel MPI, available in module valgrind/3.9.0-mpi. After loading the module, this version replaces the default valgrind.

Usage

Compile the application which you want to debug as usual. It is advisable to add compilation flags -g (to add debugging information to the binary so that you will see original source code lines in the output) and -O0 (to disable compiler optimizations).

For example, lets look at this C code, which has two problems :

```
#include <stdlib.h>

void f(void)
{
    int* x = malloc(10 * sizeof(int));
    x[10] = 0; // problem 1: heap block overrun
}              // problem 2: memory leak -- x not freed
```

```

int main(void)
{
    f();
    return 0;
}

```

Now, compile it with Intel compiler :

```

$ module add intel
$ icc -g valgrind-example.c -o valgrind-example

```

Now, lets run it with Valgrind. The syntax is :

valgrind [valgrind options] <your program binary> [your program options]

If no Valgrind options are specified, Valgrind defaults to running Memcheck tool. Please refer to the Valgrind documentation for a full description of command line options.

```

$ valgrind ./valgrind-example
==12652== Memcheck, a memory error detector
==12652== Copyright (C) 2002-2013, and GNU GPL'd, by Julian Seward et al.
==12652== Using Valgrind-3.9.0 and LibVEX; rerun with -h for copyright info
==12652== Command: ./valgrind-example
==12652==
==12652== Invalid write of size 4
==12652== at 0x40053E: f (valgrind-example.c:6)
==12652== by 0x40054E: main (valgrind-example.c:11)
==12652== Address 0x5861068 is 0 bytes after a block of size 40 alloc'd
==12652== at 0x4C27AAA: malloc (vg_replace_malloc.c:291)
==12652== by 0x400528: f (valgrind-example.c:5)
==12652== by 0x40054E: main (valgrind-example.c:11)
==12652==
==12652==
==12652== HEAP SUMMARY:
==12652== in use at exit: 40 bytes in 1 blocks
==12652== total heap usage: 1 allocs, 0 frees, 40 bytes allocated
==12652==
==12652== LEAK SUMMARY:
==12652== definitely lost: 40 bytes in 1 blocks
==12652== indirectly lost: 0 bytes in 0 blocks
==12652== possibly lost: 0 bytes in 0 blocks
==12652== still reachable: 0 bytes in 0 blocks
==12652== suppressed: 0 bytes in 0 blocks
==12652== Rerun with --leak-check=full to see details of leaked memory
==12652==
==12652== For counts of detected and suppressed errors, rerun with: -v
==12652== ERROR SUMMARY: 1 errors from 1 contexts (suppressed: 6 from 6)

```

In the output we can see that Valgrind has detected both errors - the off-by-one memory access at line 5 and a memory leak of 40 bytes. If we want a detailed analysis of the memory leak, we need to run Valgrind with `-leak-check=full` option :

```

$ valgrind --leak-check=full ./valgrind-example
==23856== Memcheck, a memory error detector
==23856== Copyright (C) 2002-2010, and GNU GPL'd, by Julian Seward et al.

```



```

==23856== Using Valgrind-3.6.0 and LibVEX; rerun with -h for copyright info
==23856== Command: ./valgrind-example
==23856==
==23856== Invalid write of size 4
==23856== at 0x40067E: f (valgrind-example.c:6)
==23856== by 0x40068E: main (valgrind-example.c:11)
==23856== Address 0x66e7068 is 0 bytes after a block of size 40 alloc'd
==23856== at 0x4C26FDE: malloc (vg_replace_malloc.c:236)
==23856== by 0x400668: f (valgrind-example.c:5)
==23856== by 0x40068E: main (valgrind-example.c:11)
==23856==
==23856==
==23856== HEAP SUMMARY:
==23856== in use at exit: 40 bytes in 1 blocks
==23856== total heap usage: 1 allocs, 0 frees, 40 bytes allocated
==23856==
==23856== 40 bytes in 1 blocks are definitely lost in loss record 1 of 1
==23856== at 0x4C26FDE: malloc (vg_replace_malloc.c:236)
==23856== by 0x400668: f (valgrind-example.c:5)
==23856== by 0x40068E: main (valgrind-example.c:11)
==23856==
==23856== LEAK SUMMARY:
==23856== definitely lost: 40 bytes in 1 blocks
==23856== indirectly lost: 0 bytes in 0 blocks
==23856== possibly lost: 0 bytes in 0 blocks
==23856== still reachable: 0 bytes in 0 blocks
==23856== suppressed: 0 bytes in 0 blocks
==23856==
==23856== For counts of detected and suppressed errors, rerun with: -v
==23856== ERROR SUMMARY: 2 errors from 2 contexts (suppressed: 6 from 6)

```

Now we can see that the memory leak is due to the `malloc()` at line 6.

Usage with MPI

Although Valgrind is not primarily a parallel debugger, it can be used to debug parallel applications as well. When launching your parallel applications, prepend the `valgrind` command. For example :

```
$ mpirun -np 4 valgrind myapplication
```

The default version without MPI support will however report a large number of false errors in the MPI library, such as :

```

==30166== Conditional jump or move depends on uninitialised value(s)
==30166== at 0x4C287E8: strlen (mc_replace_strmem.c:282)
==30166== by 0x55443BD: I_MPI_Processor_model_number (init_interface.c:427)
==30166== by 0x55439E0: I_MPI_Processor_arch_code (init_interface.c:171)
==30166== by 0x558D5AE: MPID_nem impi_init_shm_configuration (mpid_nem impi_extension
==30166== by 0x5598F4C: MPID_nem_init_ckpt (mpid_nem_init.c:566)
==30166== by 0x5598B65: MPID_nem_init (mpid_nem_init.c:489)
==30166== by 0x539BD75: MPIDI_CH3_Init (ch3_init.c:64)
==30166== by 0x5578743: MPID_Init (mpid_init.c:193)
==30166== by 0x554650A: MPIR_Init_thread (initthread.c:539)

```

```

==30166== by 0x553369F: PMPI_Init (init.c:195)
==30166== by 0x4008BD: main (valgrind-example-mpi.c:18)

```

so it is better to use the MPI-enabled valgrind from module. The MPI version requires library /apps/tools/valgrind/3.9.0/impi/lib/valgrind/libmpiwrap-amd64-linux.so, which must be included in the LD_PRELOAD environment variable.

Lets look at this MPI example :

```

#include <stdlib.h>
#include <mpi.h>

int main(int argc, char *argv[])
{
    int *data = malloc(sizeof(int)*99);

    MPI_Init(&argc, &argv);
    MPI_Bcast(data, 100, MPI_INT, 0, MPI_COMM_WORLD);
    MPI_Finalize();

    return 0;
}

```

There are two errors - use of uninitialized memory and invalid length of the buffer. Lets debug it with valgrind :

```

$ module add intel impi
$ mpicc -g valgrind-example-mpi.c -o valgrind-example-mpi
$ module add valgrind/3.9.0-mpi
$ mpirun -np 2 -env LD_PRELOAD /apps/tools/valgrind/3.9.0/mpi/lib/valgrind/libmpiwrap

```

Prints this output : (note that there is output printed for every launched MPI process)

```

==31318== Memcheck, a memory error detector
==31318== Copyright (C) 2002-2013, and GNU GPL'd, by Julian Seward et al.
==31318== Using Valgrind-3.9.0 and LibVEX; rerun with -h for copyright info
==31318== Command: ./valgrind-example-mpi
==31318==
==31319== Memcheck, a memory error detector
==31319== Copyright (C) 2002-2013, and GNU GPL'd, by Julian Seward et al.
==31319== Using Valgrind-3.9.0 and LibVEX; rerun with -h for copyright info
==31319== Command: ./valgrind-example-mpi
==31319==
valgrind MPI wrappers 31319: Active for pid 31319
valgrind MPI wrappers 31319: Try MPIWRAP_DEBUG=help for possible options
valgrind MPI wrappers 31318: Active for pid 31318
valgrind MPI wrappers 31318: Try MPIWRAP_DEBUG=help for possible options
==31319== Unaddressable byte(s) found during client check request
==31319== at 0x4E35974: check_mem_is_addressable_untyped (libmpiwrap.c:960)
==31319== by 0x4E5D0FE: PMPI_Bcast (libmpiwrap.c:908)
==31319== by 0x400911: main (valgrind-example-mpi.c:20)
==31319== Address 0x69291cc is 0 bytes after a block of size 396 alloc'd
==31319== at 0x4C27AAA: malloc (vg_replace_malloc.c:291)
==31319== by 0x4007BC: main (valgrind-example-mpi.c:8)
==31319==
==31318== Uninitialised byte(s) found during client check request

```

```

==31318== at 0x4E3591D: check_mem_is_defined_untyped (libmpiwrap.c:952)
==31318== by 0x4E5D06D: PMPI_Bcast (libmpiwrap.c:908)
==31318== by 0x400911: main (valgrind-example-mpi.c:20)
==31318== Address 0x6929040 is 0 bytes inside a block of size 396 alloc'd
==31318== at 0x4C27AAA: malloc (vg_replace_malloc.c:291)
==31318== by 0x4007BC: main (valgrind-example-mpi.c:8)
==31318==
==31318== Unaddressable byte(s) found during client check request
==31318== at 0x4E3591D: check_mem_is_defined_untyped (libmpiwrap.c:952)
==31318== by 0x4E5D06D: PMPI_Bcast (libmpiwrap.c:908)
==31318== by 0x400911: main (valgrind-example-mpi.c:20)
==31318== Address 0x69291cc is 0 bytes after a block of size 396 alloc'd
==31318== at 0x4C27AAA: malloc (vg_replace_malloc.c:291)
==31318== by 0x4007BC: main (valgrind-example-mpi.c:8)
==31318==
==31318==
==31318== HEAP SUMMARY:
==31318== in use at exit: 3,172 bytes in 67 blocks
==31318== total heap usage: 191 allocs, 124 frees, 81,203 bytes allocated
==31318==
==31319==
==31319== HEAP SUMMARY:
==31319== in use at exit: 3,172 bytes in 67 blocks
==31319== total heap usage: 175 allocs, 108 frees, 48,435 bytes allocated
==31319==
==31318== LEAK SUMMARY:
==31318== definitely lost: 408 bytes in 3 blocks
==31318== indirectly lost: 256 bytes in 1 blocks
==31318== possibly lost: 0 bytes in 0 blocks
==31318== still reachable: 2,508 bytes in 63 blocks
==31318== suppressed: 0 bytes in 0 blocks
==31318== Rerun with --leak-check=full to see details of leaked memory
==31318==
==31318== For counts of detected and suppressed errors, rerun with: -v
==31318== Use --track-origins=yes to see where uninitialised values come from
==31318== ERROR SUMMARY: 2 errors from 2 contexts (suppressed: 4 from 4)
==31319== LEAK SUMMARY:
==31319== definitely lost: 408 bytes in 3 blocks
==31319== indirectly lost: 256 bytes in 1 blocks
==31319== possibly lost: 0 bytes in 0 blocks
==31319== still reachable: 2,508 bytes in 63 blocks
==31319== suppressed: 0 bytes in 0 blocks
==31319== Rerun with --leak-check=full to see details of leaked memory
==31319==
==31319== For counts of detected and suppressed errors, rerun with: -v
==31319== ERROR SUMMARY: 1 errors from 1 contexts (suppressed: 4 from 4)

```

We can see that Valgrind has reported use of uninitialised memory on the master process (which reads the array to be broadcasted) and use of unaddressable memory on both processes.

Intel VTune Amplifier

Introduction

Intel® VTune™ Amplifier, part of Intel Parallel studio, is a GUI profiling tool designed for Intel processors. It offers a graphical performance analysis of single core and multithreaded applications. A highlight of the features:

- Hotspot analysis
- Locks and waits analysis
- Low level specific counters, such as branch analysis and memory bandwidth
- Power usage analysis - frequency and sleep states.

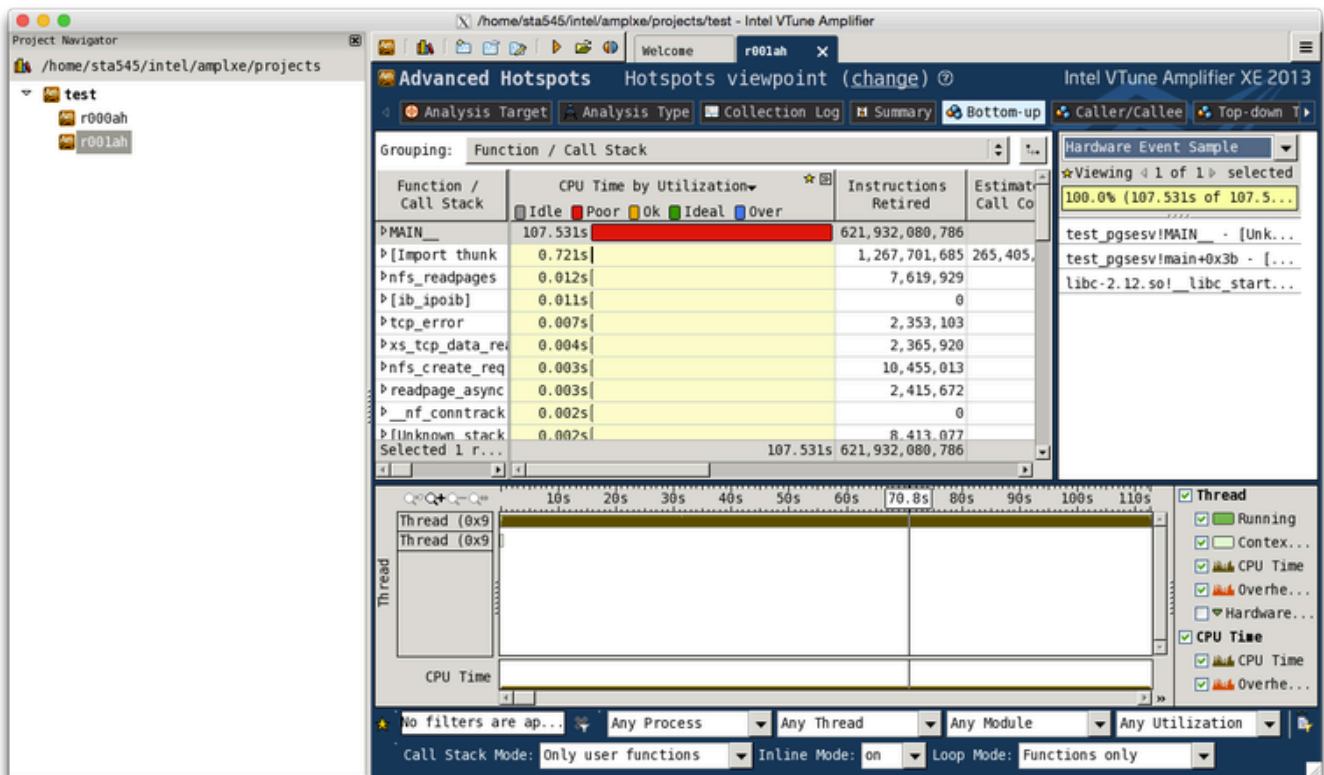


Figure 1: screenshot

Usage

To launch the GUI, first load the module:

```
$ module add VTune/2016_update1
```

and launch the GUI :

```
$ amplxe-gui
```

!!! Note “Note” To profile an application with VTune Amplifier, special kernel modules need to be loaded. The modules are not loaded on Anselm login nodes, thus direct profiling on login nodes is not possible. Use VTune on compute nodes and refer to the documentation on using GUI applications.

The GUI will open in new window. Click on “New Project...” to create a new project. After clicking OK, a new window with project properties will appear. At “Application:”, select the bath to your binary you want to profile (the binary should be compiled with -g flag). Some additional

options such as command line arguments can be selected. At “*Managed code profiling mode:*” select “*Native*” (unless you want to profile managed mode .NET/Mono applications). After clicking *OK*, your project is created.

To run a new analysis, click “*New analysis...*”. You will see a list of possible analysis. Some of them will not be possible on the current CPU (eg. Intel Atom analysis is not possible on Sandy Bridge CPU), the GUI will show an error box if you select the wrong analysis. For example, select “*Advanced Hotspots*”. Clicking on *Start* will start profiling of the application.

Remote Analysis

VTune Amplifier also allows a form of remote analysis. In this mode, data for analysis is collected from the command line without GUI, and the results are then loaded to GUI on another machine. This allows profiling without interactive graphical jobs. To perform a remote analysis, launch a GUI somewhere, open the new analysis window and then click the button “*Command line*” in bottom right corner. It will show the command line needed to perform the selected analysis.

The command line will look like this:

```
/apps/all/VTune/2016_update1/vtune_amplifier_xe_2016.1.1.434111/bin64/amplxe-cl -col
```

Copy the line to clipboard and then you can paste it in your jobscript or in command line. After the collection is run, open the GUI once again, click the menu button in the upper right corner, and select “*Open > Result...*”. The GUI will load the results from the run.

Xeon Phi

!!! Note “Note” This section is outdated. It will be updated with new information soon.

It is possible to analyze both native and offload Xeon Phi applications. For offload mode, just specify the path to the binary. For native mode, you need to specify in project properties:

Application: ssh

Application parameters: mic0 source ~/.profile && /path/to/your/bin


Note that we include source ~/.profile in the command to setup environment paths as described here.

!!! Note “Note” If the analysis is interrupted or aborted, further analysis on the card might be impossible and you will get errors like “ERROR connecting to MIC card”. In this case please contact our support to reboot the MIC card.

You may also use remote analysis to collect data from the MIC and then analyze it in the GUI later :

```
$ amplxe-cl -collect knc-hotspots -no-auto-finalize -- ssh mic0  
"export LD_LIBRARY_PATH=/apps/intel/composer_xe_2015.2.164/compiler/lib/mic/:/apps/in
```

References

1. <https://www.rcac.purdue.edu/tutorials/phi/PerformanceTuningXeonPhi-Tullos.pdf>  Performance Tuning for Intel® Xeon Phi™ Coprocessors

Total View

TotalView is a GUI-based source code multi-process, multi-thread debugger.

License and Limitations for Anselm Users

On Anselm users can debug OpenMP or MPI code that runs up to 64 parallel processes. These limitation means that:

- 1 user can debug up 64 processes, or
- 32 users can debug 2 processes, etc.

Debugging of GPU accelerated codes is also supported.

You can check the status of the licenses here:

```
cat /apps/user/licenses/totalview_features_state.txt
```

# totalview			
# -----			
# FEATURE	TOTAL	USED	AVAIL
# -----			
TotalView_Team	64	0	64
Replay	64	0	64
CUDA	64	0	64

Compiling Code to run with TotalView

Modules

Load all necessary modules to compile the code. For example:

```
module load intel

module load impi    ... or ... module load openmpi/X.X.X-icc
```

Load the TotalView module:

```
module load totalview/8.12
```

Compile the code:

```
mpicc -g -O0 -o test_debug test.c

mpif90 -g -O0 -o test_debug test.f
```

Compiler flags

Before debugging, you need to compile your code with theses flags:

!!! Note “Note” **-g** : Generates extra debugging information usable by GDB. **-g3** includes even more debugging information. This option is available for GNU and INTEL C/C++ and Fortran compilers.

****00**** : Suppress all optimizations.

Starting a Job with TotalView

Be sure to log in with an X window forwarding enabled. This could mean using the -X in the ssh:

```
ssh -X username@anselm.it4i.cz
```

Other options is to access login node using VNC. Please see the detailed information on how to use graphic user interface on Anselm.

From the login node an interactive session with X windows forwarding (-X option) can be started by following command:

```
qsub -I -X -A NONE-0-0 -q qexp -lselect=1:ncpus=16:mpiprocs=16,walltime=01:00:00
```

Then launch the debugger with the totalview command followed by the name of the executable to debug.

Debugging a serial code

To debug a serial code use:

```
totalview test_debug
```

Debugging a parallel code - option 1

To debug a parallel code compiled with **OpenMPI** you need to setup your TotalView environment:

!!! Note “Note” **Please note:** To be able to run parallel debugging procedure from the command line without stopping the debugger in the mpiexec source code you have to add the following function to your ~/.tvdrc file:

```
proc mpi_auto_run_starter {loaded_id} {
    set starter_programs {mpirun mpiexec orterun}
    set executable_name [TV::symbol get $loaded_id full_pathname]
    set file_component [file tail $executable_name]

    if {[lsearch -exact $starter_programs $file_component] != -1} {
        puts "*****"
        puts "Automatically starting $file_component"
        puts "*****"
        dgo
    }
}

# Append this function to TotalView's image load callbacks so that
# TotalView run this program automatically.

dlappend TV::image_load_callbacks mpi_auto_run_starter
```

The source code of this function can be also found in

```
/apps/mpi/openmpi/intel/1.6.5/etc/openmpi-totalview.tcl
```


!!! Note “Note” You can also add only following line to you ~/.tvdr file instead of the entire function: **source /apps/mpi/openmpi/intel/1.6.5/etc/openmpi-totalview.tcl**

You need to do this step only once.

Now you can run the parallel debugger using:

```
mpirun -tv -n 5 ./test_debug
```

When following dialog appears click on “Yes”

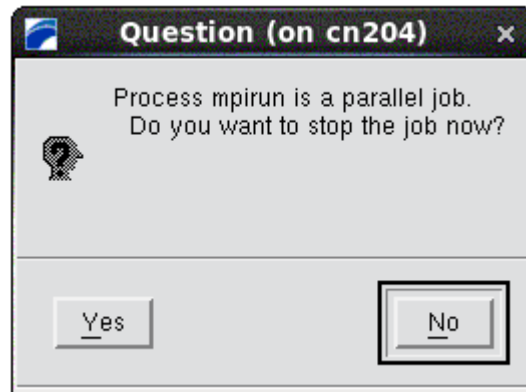


Figure 1:

At this point the main TotalView GUI window will appear and you can insert the breakpoints and start debugging:

Debugging a parallel code - option 2

Other option to start new parallel debugging session from a command line is to let TotalView to execute mpirun by itself. In this case user has to specify a MPI implementation used to compile the source code.

The following example shows how to start debugging session with Intel MPI:

```
module load intel/13.5.192 impi/4.1.1.036 totalview/8/13
```

```
totalview -mpi "Intel MPI-Hydra" -np 8 ./hello_debug impi
```

After running previous command you will see the same window as shown in the screenshot above.

More information regarding the command line parameters of the TotalView can be found TotalView Reference Guide, Chapter 7: TotalView Command Syntax.

Documentation

[1] The TotalView documentation [web page](#) is a good resource for learning more about some of the advanced TotalView features.

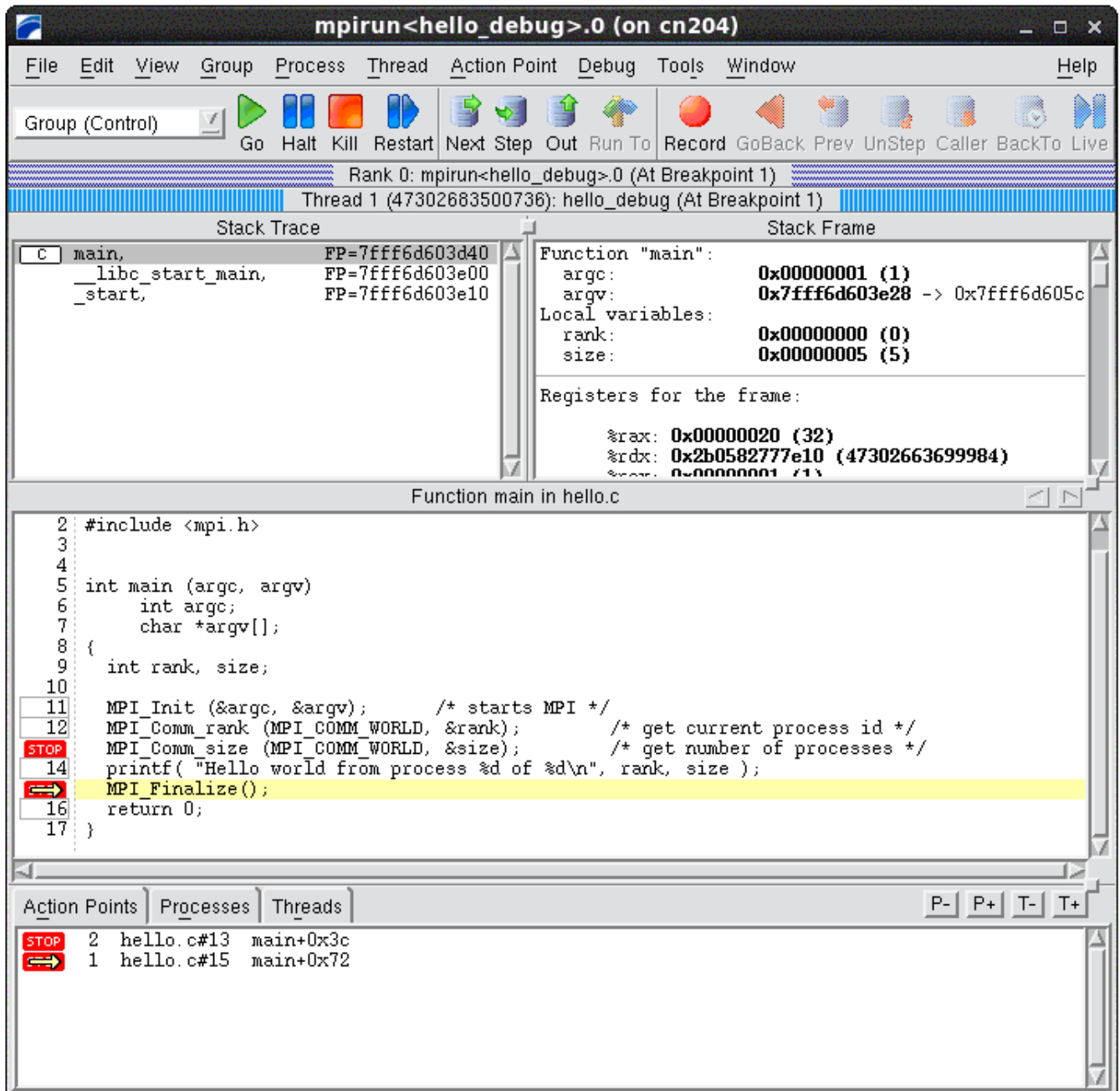


Figure 2:

Molpro

Molpro is a complete system of ab initio programs for molecular electronic structure calculations.

About Molpro

Molpro is a software package used for accurate ab-initio quantum chemistry calculations. More information can be found at the official webpage [\[1\]](#).

License

Molpro software package is available only to users that have a valid license. Please contact support to enable access to Molpro if you have a valid license appropriate for running on our cluster (eg. academic research group licence, parallel execution).

To run Molpro, you need to have a valid license token present in " \$HOME/.molpro/token". You can download the token from Molpro website [\[2\]](#).

Installed version

Currently on Anselm is installed version 2010.1, patch level 45, parallel version compiled with Intel compilers and Intel MPI.

Compilation parameters are default:

Parameter	Value
max number of atoms	200
max number of valence orbitals	300
max number of basis functions	4095
max number of states per symmetry	20
max number of state symmetries	16
max number of records	200
max number of primitives	maxbfm x [2]

Running

Molpro is compiled for parallel execution using MPI and OpenMP. By default, Molpro reads the number of allocated nodes from PBS and launches a data server on one node. On the remaining allocated nodes, compute processes are launched, one process per node, each with 16 threads. You can modify this behavior by using -n, -t and helper-server options. Please refer to the Molpro documentation [\[3\]](#) for more details.

!!! Note "Note" The OpenMP parallelization in Molpro is limited and has been observed to produce limited scaling. We therefore recommend to use MPI parallelization only. This can be achieved by passing option mpirprocs=16:ompthreads=1 to PBS.

You are advised to use the -d option to point to a directory in SCRATCH filesystem. Molpro can produce a large amount of temporary data during its run, and it is important that these are placed in the fast scratch filesystem.

Example jobscript

```
#PBS -A IT4I-0-0
#PBS -q qprod
#PBS -l select=1:ncpus=16:mpiprocs=16:ompthreads=1

cd $PBS_O_WORKDIR

# load Molpro module
module add molpro

# create a directory in the SCRATCH filesystem
mkdir -p /scratch/$USER/$PBS_JOBID

# copy an example input
cp /apps/chem/molpro/2010.1/molprop_2010_1_Linux_x86_64_i8/examples/caffeine_opt_diis

# run Molpro with default options
molpro -d /scratch/$USER/$PBS_JOBID caffeine_opt_diis.com

# delete scratch directory
rm -rf /scratch/$USER/$PBS_JOBID
```

NWChem

High-Performance Computational Chemistry

Introduction

NWChem aims to provide its users with computational chemistry tools that are scalable both in their ability to treat large scientific computational chemistry problems efficiently, and in their use of available parallel computing resources from high-performance parallel supercomputers to conventional workstation clusters.

Homepage 

Installed versions

The following versions are currently installed:

- 6.1.1, not recommended, problems have been observed with this version
- 6.3-rev2-patch1, current release with QMD patch applied. Compiled with Intel compilers, MKL and Intel MPI
- 6.3-rev2-patch1-openmpi, same as above, but compiled with OpenMPI and NWChem provided BLAS instead of MKL. This version is expected to be slower
- 6.3-rev2-patch1-venus, this version contains only libraries for VENUS interface linking. Does not provide standalone NWChem executable

For a current list of installed versions, execute:

```
module avail nwchem
```


Running

NWChem is compiled for parallel MPI execution. Normal procedure for MPI jobs applies. Sample jobscript:

```
#PBS -A IT4I-0-0
#PBS -q qprod
#PBS -l select=1:ncpus=16

module add nwchem/6.3-rev2-patch1
mpirun -np 16 nwchem h2o.nw
```

Options

Please refer to the documentation  and in the input file set the following directives :

- MEMORY : controls the amount of memory NWChem will use
- SCRATCH_DIR : set this to a directory in SCRATCH filesystem (or run the calculation completely in a scratch directory). For certain calculations, it might be advisable to reduce I/O by forcing “direct” mode, eg. “scf direct”

Intel Xeon Phi

A guide to Intel Xeon Phi usage

Intel Xeon Phi can be programmed in several modes. The default mode on Anselm is offload mode, but all modes described in this document are supported.

Intel Utilities for Xeon Phi

To get access to a compute node with Intel Xeon Phi accelerator, use the PBS interactive session

```
$ qsub -I -q qmic -A NONE-0-0
```

To set up the environment module “Intel” has to be loaded

```
$ module load intel/13.5.192
```

Information about the hardware can be obtained by running the micinfo program on the host.

```
$ /usr/bin/micinfo
```

The output of the “micinfo” utility executed on one of the Anselm node is as follows. (note: to get PCIe related details the command has to be run with root privileges)

MicInfo Utility Log

Created Mon Jul 22 00:23:50 2013

System Info

HOST OS	: Linux
OS Version	: 2.6.32-279.5.2.el6.Bull.33.x86_64
Driver Version	: 6720-15
MPSS Version	: 2.1.6720-15
Host Physical Memory	: 98843 MB

Device No: 0, Device Name: mic0

Version

Flash Version	: 2.1.03.0386
SMC Firmware Version	: 1.15.4830
SMC Boot Loader Version	: 1.8.4326
uOS Version	: 2.6.38.8-g2593b11
Device Serial Number	: ADKC30102482

Board

Vendor ID	: 0x8086
Device ID	: 0x2250
Subsystem ID	: 0x2500
Coprocessor Stepping ID	: 3
PCIe Width	: x16
PCIe Speed	: 5 GT/s
PCIe Max payload size	: 256 bytes
PCIe Max read req size	: 512 bytes
Coprocessor Model	: 0x01

```

Coproprocessor Model Ext    : 0x00
Coproprocessor Type         : 0x00
Coproprocessor Family       : 0x0b
Coproprocessor Family Ext   : 0x00
Coproprocessor Stepping     : B1
Board SKU                   : B1PRQ-5110P/5120D
ECC Mode                    : Enabled
SMC HW Revision             : Product 225W Passive CS

```

Cores

```

Total No of Active Cores : 60
Voltage                   : 1032000 uV
Frequency                 : 1052631 kHz

```

Thermal

```

Fan Speed Control        : N/A
Fan RPM                  : N/A
Fan PWM                  : N/A
Die Temp                 : 49 C

```

GDDR

```

GDDR Vendor              : Elpida
GDDR Version             : 0x1
GDDR Density             : 2048 Mb
GDDR Size                : 7936 MB
GDDR Technology          : GDDR5
GDDR Speed               : 5.000000 GT/s
GDDR Frequency           : 2500000 kHz
GDDR Voltage             : 1501000 uV

```

Offload Mode

To compile a code for Intel Xeon Phi a MPSS stack has to be installed on the machine where compilation is executed. Currently the MPSS stack is only installed on compute nodes equipped with accelerators.

```

$ qsub -I -q qmic -A NONE-0-0
$ module load intel/13.5.192

```

For debugging purposes it is also recommended to set environment variable “OFFLOAD_REPORT”. Value can be set from 0 to 3, where higher number means more debugging information.

```
export OFFLOAD_REPORT=3
```

A very basic example of code that employs offload programming technique is shown in the next listing. Please note that this code is sequential and utilizes only single core of the accelerator.

```

$ vim source-offload.cpp

#include <iostream>

int main(int argc, char* argv[])
{
    const int niter = 100000;

```

```

    double result = 0;

    #pragma offload target(mic)
    for (int i = 0; i < niter; ++i) {
        const double t = (i + 0.5) / niter;
        result += 4.0 / (t * t + 1.0);
    }
    result /= niter;
    std::cout << "Pi ~ " << result << '\n';
}

```

To compile a code using Intel compiler run

```
$ icc source-offload.cpp -o bin-offload
```

To execute the code, run the following command on the host

```
./bin-offload
```

Parallelization in Offload Mode Using OpenMP

One way of parallelization a code for Xeon Phi is using OpenMP directives. The following example shows code for parallel vector addition.

```

$ vim ./vect-add

#include <stdio.h>

typedef int T;

#define SIZE 1000

#pragma offload_attribute(push, target(mic))
T in1[SIZE];
T in2[SIZE];
T res[SIZE];
#pragma offload_attribute(pop)

// MIC function to add two vectors
__attribute__((target(mic))) add_mic(T *a, T *b, T *c, int size) {
    int i = 0;
    #pragma omp parallel for
    for (i = 0; i < size; i++)
        c[i] = a[i] + b[i];
}

// CPU function to add two vectors
void add_cpu (T *a, T *b, T *c, int size) {
    int i;
    for (i = 0; i < size; i++)
        c[i] = a[i] + b[i];
}

// CPU function to generate a vector of random numbers

```

```

void random_T (T *a, int size) {
    int i;
    for (i = 0; i < size; i++)
        a[i] = rand() % 10000; // random number between 0 and 9999
}

// CPU function to compare two vectors
int compare(T *a, T *b, T size ){
    int pass = 0;
    int i;
    for (i = 0; i < size; i++){
        if (a[i] != b[i]) {
            printf("Value mismatch at location %d, values %d and %dn",i, a[i], b[i]);
            pass = 1;
        }
    }
    if (pass == 0) printf ("Test passedn"); else printf ("Test Failedn");
    return pass;
}

int main()
{
    int i;
    random_T(in1, SIZE);
    random_T(in2, SIZE);

    #pragma offload target(mic) in(in1,in2) inout(res)
    {

        // Parallel loop from main function
        #pragma omp parallel for
        for (i=0; i<SIZE; i++)
            res[i] = in1[i] + in2[i];

        // or parallel loop is called inside the function
        add_mic(in1, in2, res, SIZE);

    }

    //Check the results with CPU implementation
    T res_cpu[SIZE];
    add_cpu(in1, in2, res_cpu, SIZE);
    compare(res, res_cpu, SIZE);
}

```

During the compilation Intel compiler shows which loops have been vectorized in both host and accelerator. This can be enabled with compiler option “-vec-report2”. To compile and execute the code run

```

$ icc vect-add.c -openmp_report2 -vec-report2 -o vect-add

$ ./vect-add

```


Some interesting compiler flags useful not only for code debugging are:

!!! Note “Note” Debugging

`openmp_report[0|1|2]` - controls the compiler based vectorization diagnostic level
`vec-report[0|1|2]` - controls the OpenMP parallelizer diagnostic level

Performance ooptimization

`xhost` - FOR HOST ONLY - to generate AVX (Advanced Vector Extensions) instructions.

Automatic Offload using Intel MKL Library

Intel MKL includes an Automatic Offload (AO) feature that enables computationally intensive MKL functions called in user code to benefit from attached Intel Xeon Phi coprocessors automatically and transparently.

Behavioral of automatic offload mode is controlled by functions called within the program or by environmental variables. Complete list of controls is listed here[\[4\]](#).

The Automatic Offload may be enabled by either an MKL function call within the code:

```
mkl_mic_enable();
```

or by setting environment variable

```
$ export MKL_MIC_ENABLE=1
```

To get more information about automatic offload please refer to “Using Intel® MKL Automatic Offload on Intel® Xeon Phi™ Coprocessors[\[4\]](#)” white paper or Intel MKL documentation[\[4\]](#).

Automatic offload example

At first get an interactive PBS session on a node with MIC accelerator and load “intel” module that automatically loads “mkl” module as well.

```
$ qsub -I -q qmic -A OPEN-0-0 -l select=1:ncpus=16  
$ module load intel
```

Following example show how to automatically offload an SGEMM (single precision - g dir=“auto”>eneral matrix multiply) function to MIC coprocessor. The code can be copied to a file and compiled without any necessary modification.

```
$ vim sgemm-ao-short.c
```

```
#include <stdio.h>  
#include <stdlib.h>  
#include <malloc.h>  
#include <stdint.h>
```

```
#include "mkl.h"
```

```
int main(int argc, char **argv)  
{
```

```
    float *A, *B, *C; /* Matrices */
```

```
    MKL_INT N = 2560; /* Matrix dimensions */
```

```

MKL_INT LD = N; /* Leading dimension */
int matrix_bytes; /* Matrix size in bytes */
int matrix_elements; /* Matrix size in elements */

float alpha = 1.0, beta = 1.0; /* Scaling factors */
char transa = 'N', transb = 'N'; /* Transposition options */

int i, j; /* Counters */

matrix_elements = N * N;
matrix_bytes = sizeof(float) * matrix_elements;

/* Allocate the matrices */
A = malloc(matrix_bytes); B = malloc(matrix_bytes); C = malloc(matrix_bytes);

/* Initialize the matrices */
for (i = 0; i < matrix_elements; i++) {
    A[i] = 1.0; B[i] = 2.0; C[i] = 0.0;
}

printf("Computing SGEMM on the hostn");
sgemm(&transa, &transb, &N, &N, &N, &alpha, A, &N, B, &N, &beta, C, &N);

printf("Enabling Automatic Offloadn");
/* Alternatively, set environment variable MKL_MIC_ENABLE=1 */
mkl_mic_enable();

int ndevices = mkl_mic_get_device_count(); /* Number of MIC devices */
printf("Automatic Offload enabled: %d MIC devices presentn", ndevices);

printf("Computing SGEMM with automatic workdivisionn");
sgemm(&transa, &transb, &N, &N, &N, &alpha, A, &N, B, &N, &beta, C, &N);

/* Free the matrix memory */
free(A); free(B); free(C);

printf("Donen");

return 0;
}

```

!!! Note “Note” Please note: This example is simplified version of an example from MKL. The expanded version can be found here: [\\$MKL_EXAMPLES/mic_ao/blasr/source/sgemm.c](#)

To compile a code using Intel compiler use:

```
$ icc -mkl sgemm-ao-short.c -o sgemm
```

For debugging purposes enable the offload report to see more information about automatic offloading.

```
$ export OFFLOAD_REPORT=2
```

The output of a code should look similar to following listing, where lines starting with [MKL] are generated by offload reporting:

```

Computing SGEMM on the host
Enabling Automatic Offload
Automatic Offload enabled: 1 MIC devices present
Computing SGEMM with automatic workdivision
[MKL] [MIC --] [AO Function]      SGEMM
[MKL] [MIC --] [AO SGEMM Workdivision]  0.00 1.00
[MKL] [MIC 00] [AO SGEMM CPU Time]      0.463351 seconds
[MKL] [MIC 00] [AO SGEMM MIC Time]      0.179608 seconds
[MKL] [MIC 00] [AO SGEMM CPU->MIC Data] 52428800 bytes
[MKL] [MIC 00] [AO SGEMM MIC->CPU Data] 26214400 bytes
Done

```

Native Mode

In the native mode a program is executed directly on Intel Xeon Phi without involvement of the host machine. Similarly to offload mode, the code is compiled on the host computer with Intel compilers.

To compile a code user has to be connected to a compute with MIC and load Intel compilers module. To get an interactive session on a compute node with an Intel Xeon Phi and load the module use following commands:

```

$ qsub -I -q qmic -A NONE-0-0

$ module load intel/13.5.192

```

!!! Note “Note” Please note that particular version of the Intel module is specified. This information is used later to specify the correct library paths.

To produce a binary compatible with Intel Xeon Phi architecture user has to specify “-mmic” compiler flag. Two compilation examples are shown below. The first example shows how to compile OpenMP parallel code “vect-add.c” for host only:

```

$ icc -xhost -no-offload -fopenmp vect-add.c -o vect-add-host

```

To run this code on host, use:

```

$ ./vect-add-host

```

The second example shows how to compile the same code for Intel Xeon Phi:

```

$ icc -mmic -fopenmp vect-add.c -o vect-add-mic

```

Execution of the Program in Native Mode on Intel Xeon Phi

The user access to the Intel Xeon Phi is through the SSH. Since user home directories are mounted using NFS on the accelerator, users do not have to copy binary files or libraries between the host and accelerator.

To connect to the accelerator run:

```

$ ssh mic0

```

If the code is sequential, it can be executed directly:

```

mic0 $ ~/path_to_binary/vect-add-seq-mic

```

If the code is parallelized using OpenMP a set of additional libraries is required for execution. To locate these libraries new path has to be added to the LD_LIBRARY_PATH environment variable prior to the execution:

```
mic0 $ export LD_LIBRARY_PATH=/apps/intel/composer_xe_2013.5.192/compiler/lib/mic:$LD
```

!!! Note “Note” Please note that the path exported in the previous example contains path to a specific compiler (here the version is 5.192). This version number has to match with the version number of the Intel compiler module that was used to compile the code on the host computer.

For your information the list of libraries and their location required for execution of an OpenMP parallel code on Intel Xeon Phi is:

!!! Note “Note” /apps/intel/composer_xe_2013.5.192/compiler/lib/mic

- libiomp5.so
- libimf.so
- libsvml.so
- libirng.so
- libintlc.so.5

Finally, to run the compiled code use:

```
$ ~/path_to_binary/vect-add-mic
```

OpenCL

OpenCL (Open Computing Language) is an open standard for general-purpose parallel programming for diverse mix of multi-core CPUs, GPU coprocessors, and other parallel processors. OpenCL provides a flexible execution model and uniform programming environment for software developers to write portable code for systems running on both the CPU and graphics processors or accelerators like the Intel® Xeon Phi.

On Anselm OpenCL is installed only on compute nodes with MIC accelerator, therefore OpenCL code can be compiled only on these nodes.

```
module load opencl-sdk opencl-rt
```

Always load “opencl-sdk” (providing devel files like headers) and “opencl-rt” (providing dynamic library libOpenCL.so) modules to compile and link OpenCL code. Load “opencl-rt” for running your compiled code.

There are two basic examples of OpenCL code in the following directory:

```
/apps/intel/opencl-examples/
```

First example “CapsBasic” detects OpenCL compatible hardware, here CPU and MIC, and prints basic information about the capabilities of it.

```
/apps/intel/opencl-examples/CapsBasic/capsbasic
```

To compile and run the example copy it to your home directory, get a PBS interactive session on of the nodes with MIC and run make for compilation. Make files are very basic and shows how the OpenCL code can be compiled on Anselm.

```
$ cp /apps/intel/opencl-examples/CapsBasic/* .  
$ qsub -I -q qmic -A NONE-0-0  
$ make
```

The compilation command for this example is:

```
$ g++ capsbasic.cpp -lOpenCL -o capsbasic -I/apps/intel/opencl/include/
```

After executing the compiled binary file, following output should be displayed.

```
./capsbasic
```

```
Number of available platforms: 1
```

```
Platform names:
```

```
[0] Intel(R) OpenCL [Selected]
```

```
Number of devices available for each type:
```

```
CL_DEVICE_TYPE_CPU: 1
```

```
CL_DEVICE_TYPE_GPU: 0
```

```
CL_DEVICE_TYPE_ACCELERATOR: 1
```

```
** Detailed information for each device **
```

```
CL_DEVICE_TYPE_CPU[0]
```

```
CL_DEVICE_NAME: Intel(R) Xeon(R) CPU E5-2470 0 @ 2.30GHz
```

```
CL_DEVICE_AVAILABLE: 1
```

```
...
```

```
CL_DEVICE_TYPE_ACCELERATOR[0]
```

```
CL_DEVICE_NAME: Intel(R) Many Integrated Core Acceleration Card
```

```
CL_DEVICE_AVAILABLE: 1
```

```
...
```

!!! Note “Note” More information about this example can be found on Intel website: <http://software.intel.com/en-us/vcsample/samples/caps-basic/>

The second example that can be found in “/apps/intel/opencl-examples” directory is General Matrix Multiply. You can follow the the same procedure to download the example to your directory and compile it.

```
$ cp -r /apps/intel/opencl-examples/* .
```

```
$ qsub -I -q qmic -A NONE-0-0
```

```
$ cd GEMM
```

```
$ make
```

The compilation command for this example is:

```
$ g++ cmdoptions.cpp gemm.cpp ../common/basic.cpp ../common/cmdparser.cpp ../common/o
```

To see the performance of Intel Xeon Phi performing the DGEMM run the example as follows:

```
./gemm -d 1
```

```
Platforms (1):
```

```
[0] Intel(R) OpenCL [Selected]
```

```
Devices (2):
```

```
[0] Intel(R) Xeon(R) CPU E5-2470 0 @ 2.30GHz
```

```
[1] Intel(R) Many Integrated Core Acceleration Card [Selected]
```

```
Build program options: "-DT=float -DTILE_SIZE_M=1 -DTILE_GROUP_M=16 -DTILE_SIZE_N=128
```

```
Running gemm_nn kernel with matrix size: 3968x3968
```

```
Memory row stride to ensure necessary alignment: 15872 bytes
```

```
Size of memory region for one matrix: 62980096 bytes
```

```
Using alpha = 0.57599 and beta = 0.872412
```

```

...
Host time: 0.292953 sec.
Host perf: 426.635 GFLOPS
Host time: 0.293334 sec.
Host perf: 426.081 GFLOPS
...

```

!!! Note “Note” Please note: GNU compiler is used to compile the OpenCL codes for Intel MIC. You do not need to load Intel compiler module.

MPI

Environment setup and compilation

Again an MPI code for Intel Xeon Phi has to be compiled on a compute node with accelerator and MPSS software stack installed. To get to a compute node with accelerator use:

```
$ qsub -I -q qmic -A NONE-0-0
```

The only supported implementation of MPI standard for Intel Xeon Phi is Intel MPI. To setup a fully functional development environment a combination of Intel compiler and Intel MPI has to be used. On a host load following modules before compilation:

```
$ module load intel/13.5.192 impi/4.1.1.036
```

To compile an MPI code for host use:

```
bash $ mpiicc -xhost -o mpi-test mpi-test.cbash
```

To compile the same code for Intel Xeon Phi architecture use:

```
$ mpiicc -mmic -o mpi-test-mic mpi-test.c
```

An example of basic MPI version of “hello-world” example in C language, that can be executed on both host and Xeon Phi is (can be directly copy and pasted to a .c file)

```

#include <stdio.h>
#include <mpi.h>

int main (argc, argv)
{
    int argc;
    char *argv[];

    int rank, size;

    int len;
    char node[MPI_MAX_PROCESSOR_NAME];

    MPI_Init (&argc, &argv);          /* starts MPI */
    MPI_Comm_rank (MPI_COMM_WORLD, &rank);    /* get current process id */
    MPI_Comm_size (MPI_COMM_WORLD, &size);    /* get number of processes */

    MPI_Get_processor_name(node,&len);

    printf( "Hello world from process %d of %d on host %s n", rank, size, node );
    MPI_Finalize();

```

```

    return 0;
}

```

MPI programming models

Intel MPI for the Xeon Phi coprocessors offers different MPI programming models:

!!! Note “Note” **Host-only model** - all MPI ranks reside on the host. The coprocessors can be used by using offload pragmas. (Using MPI calls inside offloaded code is not supported.)

****Coprocesor-only model**** - all MPI ranks reside only on the coprocessors.

****Symmetric model**** - the MPI ranks reside on both the host and the coprocessor. Most general M

Host-only model

In this case all environment variables are set by modules, so to execute the compiled MPI program on a single node, use:

```
$ mpirun -np 4 ./mpi-test
```

The output should be similar to:

```

Hello world from process 1 of 4 on host cn207
Hello world from process 3 of 4 on host cn207
Hello world from process 2 of 4 on host cn207
Hello world from process 0 of 4 on host cn207

```

Coprocesor-only model

There are two ways how to execute an MPI code on a single coprocessor: 1.) lunch the program using “**mpirun**” from the coprocessor; or 2.) lunch the task using “**mpiexec.hydra**” from a host.

Execution on coprocessor

Similarly to execution of OpenMP programs in native mode, since the environmental module are not supported on MIC, user has to setup paths to Intel MPI libraries and binaries manually. One time setup can be done by creating a “**.profile**” file in user’s home directory. This file sets up the environment on the MIC automatically once user access to the accelerator through the SSH.

```
$ vim ~/.profile
```

```
PS1='[u@h W]$ '
```

```
export PATH=/usr/bin:/usr/sbin:/bin:/sbin
```

```
#OpenMP
```

```
export LD_LIBRARY_PATH=/apps/intel/composer_xe_2013.5.192/compiler/lib/mic:$LD_LIBRARY_PATH
```

```
#Intel MPI
```

```
export LD_LIBRARY_PATH=/apps/intel/impi/4.1.1.036/mic/lib/:$LD_LIBRARY_PATH
```

```
export PATH=/apps/intel/impi/4.1.1.036/mic/bin/:$PATH
```

!!! Note “Note” Please note: - this file sets up both environmental variable for both MPI and OpenMP libraries. - this file sets up the paths to a particular version of Intel MPI library and particular version of an Intel compiler. These versions have to match with loaded modules.

To access a MIC accelerator located on a node that user is currently connected to, use:

```
$ ssh mic0
```

or in case you need specify a MIC accelerator on a particular node, use:

```
$ ssh cn207-mic0
```

To run the MPI code in parallel on multiple core of the accelerator, use:

```
$ mpirun -np 4 ./mpi-test-mic
```

The output should be similar to:

```
Hello world from process 1 of 4 on host cn207-mic0
Hello world from process 2 of 4 on host cn207-mic0
Hello world from process 3 of 4 on host cn207-mic0
Hello world from process 0 of 4 on host cn207-mic0
```

Execution on host

If the MPI program is launched from host instead of the coprocessor, the environmental variables are not set using the “.profile” file. Therefore user has to specify library paths from the command line when calling “mpiexec”.

First step is to tell mpiexec that the MPI should be executed on a local accelerator by setting up the environmental variable “I_MPI_MIC”

```
$ export I_MPI_MIC=1
```

Now the MPI program can be executed as:

```
$ mpiexec.hydra -genv LD_LIBRARY_PATH /apps/intel/impi/4.1.1.036/mic/lib/ -host mic0
```

or using mpirun

```
$ mpirun -genv LD_LIBRARY_PATH /apps/intel/impi/4.1.1.036/mic/lib/ -host mic0 -n 4 ~/
```

!!! Note “Note” Please note: - the full path to the binary has to specified (here: “>~/mpi-test-mic”) - the LD_LIBRARY_PATH has to match with Intel MPI module used to compile the MPI code

The output should be again similar to:

```
Hello world from process 1 of 4 on host cn207-mic0
Hello world from process 2 of 4 on host cn207-mic0
Hello world from process 3 of 4 on host cn207-mic0
Hello world from process 0 of 4 on host cn207-mic0
```

!!! Note “Note” Please note that the “mpiexec.hydra” requires a file the MIC filesystem. If the file is missing please contact the system administrators. A simple test to see if the file is present is to execute:

```
$ ssh mic0 ls /bin/pmi_proxy
/bin/pmi_proxy
```

Execution on host - MPI processes distributed over multiple accelerators on multiple nodes

To get access to multiple nodes with MIC accelerator, user has to use PBS to allocate the resources. To start interactive session, that allocates 2 compute nodes = 2 MIC accelerators run qsub command with following parameters:


```
$ qsub -I -q qmic -A NONE-0-0 -l select=2:ncpus=16
```

```
$ module load intel/13.5.192 impi/4.1.1.036
```

This command connects user through ssh to one of the nodes immediately. To see the other nodes that have been allocated use:

```
$ cat $PBS_NODEFILE
```

For example:

```
cn204.bullx
cn205.bullx
```

This output means that the PBS allocated nodes cn204 and cn205, which means that user has direct access to “cn204-mic0” and “cn-205-mic0” accelerators.

!!! Note “Note” Please note: At this point user can connect to any of the allocated nodes or any of the allocated MIC accelerators using ssh:

- to connect to the second node : `** $ ssh cn205**`
- to connect to the accelerator on the first node from the first node: `**$ ssh cn204-mic0**` or
- to connect to the accelerator on the second node from the first node: `**$ ssh cn205-mic0**`

At this point we expect that correct modules are loaded and binary is compiled. For parallel execution the mpiexec.hydra is used. Again the first step is to tell mpiexec that the MPI can be executed on MIC accelerators by setting up the environmental variable “I_MPI_MIC”

```
$ export I_MPI_MIC=1
```

The launch the MPI program use:

```
$ mpiexec.hydra -genv LD_LIBRARY_PATH /apps/intel/impi/4.1.1.036/mic/lib/
-genv I_MPI_FABRICS_LIST tcp
-genv I_MPI_FABRICS shm:tcp
-genv I_MPI_TCP_NETMASK=10.1.0.0/16
-host cn204-mic0 -n 4 ~/mpi-test-mic
: -host cn205-mic0 -n 6 ~/mpi-test-mic
```

or using mpirun:

```
$ mpirun -genv LD_LIBRARY_PATH /apps/intel/impi/4.1.1.036/mic/lib/
-genv I_MPI_FABRICS_LIST tcp
-genv I_MPI_FABRICS shm:tcp
-genv I_MPI_TCP_NETMASK=10.1.0.0/16
-host cn204-mic0 -n 4 ~/mpi-test-mic
: -host cn205-mic0 -n 6 ~/mpi-test-mic
```

In this case four MPI processes are executed on accelerator cn204-mic and six processes are executed on accelerator cn205-mic0. The sample output (sorted after execution) is:

```
Hello world from process 0 of 10 on host cn204-mic0
Hello world from process 1 of 10 on host cn204-mic0
Hello world from process 2 of 10 on host cn204-mic0
Hello world from process 3 of 10 on host cn204-mic0
Hello world from process 4 of 10 on host cn205-mic0
Hello world from process 5 of 10 on host cn205-mic0
Hello world from process 6 of 10 on host cn205-mic0
Hello world from process 7 of 10 on host cn205-mic0
```

```
Hello world from process 8 of 10 on host cn205-mic0
Hello world from process 9 of 10 on host cn205-mic0
```

The same way MPI program can be executed on multiple hosts:

```
$ mpiexec.hydra -genv LD_LIBRARY_PATH /apps/intel/impi/4.1.1.036/mic/lib/
-genv I_MPI_FABRICS_LIST tcp
-genv I_MPI_FABRICS shm:tcp
-genv I_MPI_TCP_NETMASK=10.1.0.0/16
-host cn204 -n 4 ~/mpi-test
: -host cn205 -n 6 ~/mpi-test
```

Symmetric model

In a symmetric mode MPI programs are executed on both host computer(s) and MIC accelerator(s). Since MIC has a different architecture and requires different binary file produced by the Intel compiler two different files has to be compiled before MPI program is executed.

In the previous section we have compiled two binary files, one for hosts “**mpi-test**” and one for MIC accelerators “**mpi-test-mic**”. These two binaries can be executed at once using `mpiexec.hydra`:

```
$ mpiexec.hydra
-genv I_MPI_FABRICS_LIST tcp
-genv I_MPI_FABRICS shm:tcp
-genv I_MPI_TCP_NETMASK=10.1.0.0/16
-genv LD_LIBRARY_PATH /apps/intel/impi/4.1.1.036/mic/lib/
-host cn205 -n 2 ~/mpi-test
: -host cn205-mic0 -n 2 ~/mpi-test-mic
```

In this example the first two parameters (line 2 and 3) sets up required environment variables for execution. The third line specifies binary that is executed on host (here `cn205`) and the last line specifies the binary that is execute on the accelerator (here `cn205-mic0`).

The output of the program is:

```
Hello world from process 0 of 4 on host cn205
Hello world from process 1 of 4 on host cn205
Hello world from process 2 of 4 on host cn205-mic0
Hello world from process 3 of 4 on host cn205-mic0
```

The execution procedure can be simplified by using the `mpirun` command with the machine file a a parameter. Machine file contains list of all nodes and accelerators that should used to execute MPI processes.

An example of a machine file that uses 2 >hosts (**cn205** and **cn206**) and 2 accelerators (**cn205-mic0** and **cn206-mic0**) to run 2 MPI processes on each of them:

```
$ cat hosts_file_mix
cn205:2
cn205-mic0:2
cn206:2
cn206-mic0:2
```

In addition if a naming convention is set in a way that the name of the binary for host is “**bin_name**” and the name of the binary for the accelerator is “**bin_name-mic**” then by setting up the environment variable `I_MPI_MIC_POSTFIX` to “**-mic**” user do not have to specify the names of booth binaries. In this case `mpirun` needs just the name of the host

binary file (i.e. “mpi-test”) and uses the suffix to get a name of the binary for accelerator (i.e. “mpi-test-mic”).

```
$ export I_MPI_MIC_POSTFIX=-mic
```

To run the MPI code using mpirun and the machine file “hosts_file_mix” use:

```
$ mpirun
  -genv I_MPI_FABRICS shm:tcp
  -genv LD_LIBRARY_PATH /apps/intel/impi/4.1.1.036/mic/lib/
  -genv I_MPI_FABRICS_LIST tcp
  -genv I_MPI_FABRICS shm:tcp
  -genv I_MPI_TCP_NETMASK=10.1.0.0/16
  -machinefile hosts_file_mix
  ~/mpi-test
```

A possible output of the MPI “hello-world” example executed on two hosts and two accelerators is:

```
Hello world from process 0 of 8 on host cn204
Hello world from process 1 of 8 on host cn204
Hello world from process 2 of 8 on host cn204-mic0
Hello world from process 3 of 8 on host cn204-mic0
Hello world from process 4 of 8 on host cn205
Hello world from process 5 of 8 on host cn205
Hello world from process 6 of 8 on host cn205-mic0
Hello world from process 7 of 8 on host cn205-mic0
```

!!! Note “Note” Please note: At this point the MPI communication between MIC accelerators on different nodes uses 1Gb Ethernet only.

Using the PBS automatically generated node-files

PBS also generates a set of node-files that can be used instead of manually creating a new one every time. Three node-files are generated:

!!! Note “Note” **Host only node-file:**

- /lscratch/\${PBS_JOBID}/nodefile-cn MIC only node-file:
- /lscratch/\${PBS_JOBID}/nodefile-mic Host and MIC node-file:
- /lscratch/\${PBS_JOBID}/nodefile-mix

Please note each host or accelerator is listed only per files. User has to specify how many jobs should be executed per node using “-n” parameter of the mpirun command.

Optimization

For more details about optimization techniques please read Intel document Optimization and Performance Tuning for Intel® Xeon Phi™ Coprocessors [\[4\]](#)


ParaView

An open-source, multi-platform data analysis and visualization application

Introduction

ParaView is an open-source, multi-platform data analysis and visualization application. ParaView users can quickly build visualizations to analyze their data using qualitative and quantitative techniques. The data exploration can be done interactively in 3D or programmatically using ParaView's batch processing capabilities.


ParaView was developed to analyze extremely large datasets using distributed memory computing resources. It can be run on supercomputers to analyze datasets of exascale size as well as on laptops for smaller data.

Homepage : <http://www.paraview.org/> 

Installed version

Currently, version 4.0.1 compiled with GCC 4.8.1 against Bull MPI library and OSMesa 10.0 is installed on Anselm.

Usage

On Anselm, ParaView is to be used in client-server mode. A parallel ParaView server is launched on compute nodes by the user, and client is launched on your desktop PC to control and view the visualization. Download ParaView client application for your OS here: <http://paraview.org/paraview/resources/software.php> . Important : **your version must match the version number installed on Anselm !** (currently v4.0.1)

Launching server

To launch the server, you must first allocate compute nodes, for example

```
$ qsub -I -q qprod -A OPEN-0-0 -l select=2
```

to launch an interactive session on 2 nodes. Refer to Resource Allocation and Job Execution for details.

After the interactive session is opened, load the ParaView module :

```
$ module add paraview
```

Now launch the parallel server, with number of nodes times 16 processes:

```
$ mpirun -np 32 pvserver --use-offscreen-rendering  
Waiting for client...  
Connection URL: cs://cn77:11111  
Accepting connection(s): cn77:11111
```

Note that the server is listening on compute node cn77 in this case, we shall use this information later.

Client connection

Because a direct connection is not allowed to compute nodes on Anselm, you must establish a SSH tunnel to connect to the server. Choose a port number on your PC to be forwarded to ParaView server, for example 12345. If your PC is running Linux, use this command to establish a SSH tunnel:

```
ssh -TN -L 12345:cn77:11111 username@anselm.it4i.cz
```

replace username with your login and cn77 with the name of compute node your ParaView server is running on (see previous step). If you use PuTTY on Windows, load Anselm connection configuration, then go to Connection->SSH->Tunnels to set up the port forwarding. Click Remote radio button. Insert 12345 to Source port textbox. Insert cn77:11111. Click Add button, then Open.

Now launch ParaView client installed on your desktop PC. Select File->Connect..., click Add Server. Fill in the following :

Name : Anselm tunnel Server Type : Client/Server Host : localhost Port : 12345

Click Configure, Save, the configuration is now saved for later use. Now click Connect to connect to the ParaView server. In your terminal where you have interactive session with ParaView server launched, you should see:

```
Client connected.
```

You can now use Parallel ParaView.

Close server

Remember to close the interactive session after you finish working with ParaView server, as it will remain launched even after your client is disconnected and will continue to consume resources.

GPU support

Currently, GPU acceleration is not supported in the server and ParaView will not take advantage of accelerated nodes on Anselm. Support for GPU acceleration might be added in the future.

Remote visualization service

Introduction

The goal of this service is to provide the users a GPU accelerated use of OpenGL applications, especially for pre- and post- processing work, where not only the GPU performance is needed but also fast access to the shared file systems of the cluster and a reasonable amount of RAM.

The service is based on integration of open source tools VirtualGL and TurboVNC together with the cluster's job scheduler PBS Professional.

Currently two compute nodes are dedicated for this service with following configuration for each node:

Visualization node configuration	
CPU	2x Intel Sandy Bridge E5-2670, 2.6GHz
Processor cores	16 (2x8 cores)
RAM	64 GB, min. 4 GB per core
GPU	NVIDIA Quadro 4000, 2GB RAM
Local disk drive	yes - 500 GB
Compute network	InfiniBand QDR

Schematic overview

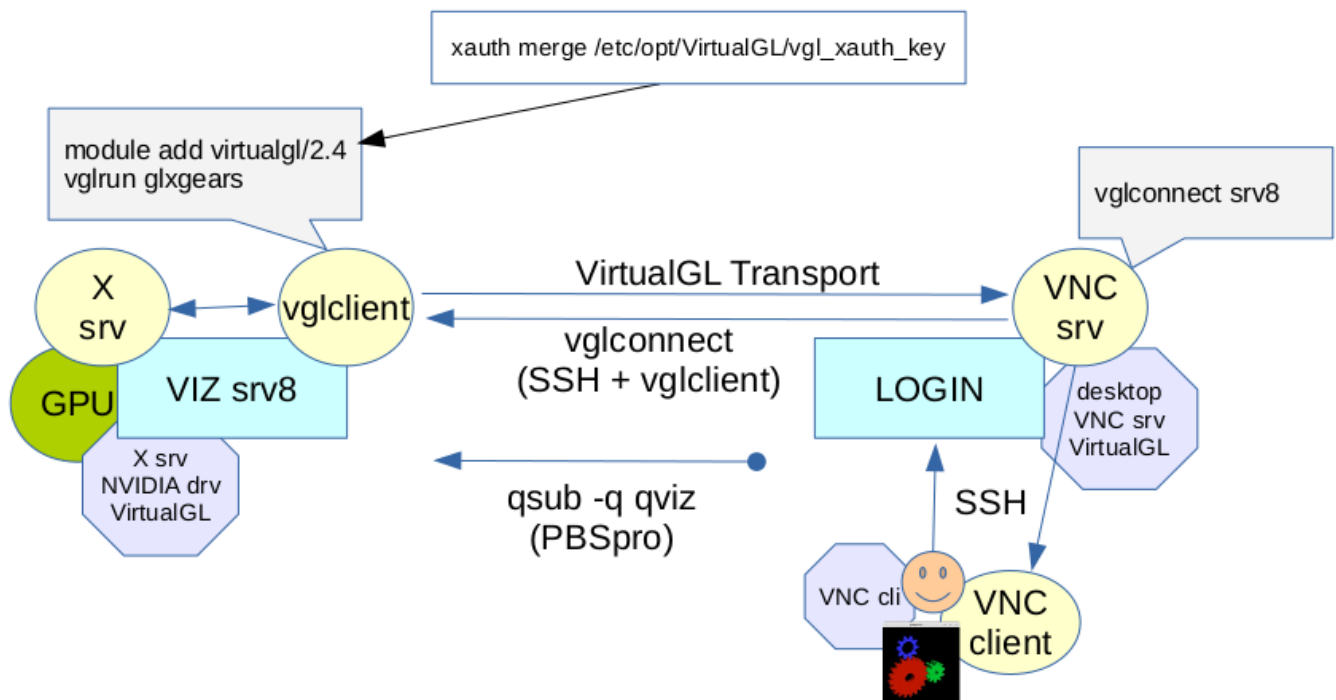


Figure 1: rem_vis_scheme

Legend

User



Hardware node

LOGIN

Logical Component

VNC
client

Software Component

VNC cli

Hardware component

GPU

Application




Command

vglconnect srv8

Figure 2: rem_vis_legend

How to use the service

Setup and start your own TurboVNC server.

TurboVNC is designed and implemented for cooperation with VirtualGL and available for free for all major platforms. For more information and download, please refer to: <http://sourceforge.net/projects/turbovnc/> 

Always use TurboVNC on both sides (server and client) **don't mix TurboVNC and other VNC implementations** (TightVNC, TigerVNC, ...) as the VNC protocol implementation may slightly differ and diminish your user experience by introducing picture artifacts, etc.

The procedure is:

1. Connect to a login node.

Please follow the documentation.

2. Run your own instance of TurboVNC server.

To have the OpenGL acceleration, **24 bit color depth must be used**. Otherwise only the geometry (desktop size) definition is needed.

At first VNC server run you need to define a password.

This example defines desktop with dimensions 1200x700 pixels and 24 bit color depth.

```
$ module load turbovnc/1.2.2
$ vncserver -geometry 1200x700 -depth 24
```

```
Desktop 'TurboVNC: login2:1 (username)' started on display login2:1
```

```
Starting applications specified in /home/username/.vnc/xstartup.turbovnc
Log file is /home/username/.vnc/login2:1.log
```

3. Remember which display number your VNC server runs (you will need it in the future to stop the server).

```
$ vncserver -list
```

TurboVNC server sessions:

```
X DISPLAY # PROCESS ID
:1 23269
```

In this example the VNC server runs on display :1.

4. Remember the exact login node, where your VNC server runs.

```
$ uname -n
login2
```

In this example the VNC server runs on **login2**.

5. Remember on which TCP port your own VNC server is running.

To get the port you have to look to the log file of your VNC server.

```
$ grep -E "VNC.*port" /home/username/.vnc/login2:1.log
20/02/2015 14:46:41 Listening for VNC connections on TCP port 5901
```

In this example the VNC server listens on TCP port **5901**.

6. Connect to the login node where your VNC server runs with SSH to tunnel your VNC session.

Tunnel the TCP port on which your VNC server is listening.

```
$ ssh login2.anselm.it4i.cz -L 5901:localhost:5901
```

x-window-system/ *If you use Windows and Putty, please refer to port forwarding setup in the documentation: x-window-and-vnc#section-12*

7. If you don't have Turbo VNC installed on your workstation.

Get it from: <http://sourceforge.net/projects/turbovnc/>

8. Run TurboVNC Viewer from your workstation.

Mind that you should connect through the SSH tunneled port. In this example it is 5901 on your workstation (localhost).

```
$ vncviewer localhost:5901
```

*If you use Windows version of TurboVNC Viewer, just run the Viewer and use address **localhost:5901**.*

9. Proceed to the chapter “Access the visualization node.”

Now you should have working TurboVNC session connected to your workstation.

10. After you end your visualization session.

Don't forget to correctly shutdown your own VNC server on the login node!

```
$ vncserver -kill :1
```

Access the visualization node

To access the node use a dedicated PBS Professional scheduler queue **qviz**. The queue has following properties:

queue	active project	project resources	nodes min ncpus*	priority authorization walltime	—
—	qviz	Visualization queue	yes none required	2 4 150 no 1 hour / 2 hours	

Currently when accessing the node, each user gets 4 cores of a CPU allocated, thus approximately 16 GB of RAM and 1/4 of the GPU capacity. *If more GPU power or RAM is required, it is recommended to allocate one whole node per user, so that all 16 cores, whole RAM and whole GPU is exclusive. This is currently also the maximum allowed allocation per one user. One hour of work is allocated by default, the user may ask for 2 hours maximum.*

To access the visualization node, follow these steps:

1. In your VNC session, open a terminal and allocate a node using PBSPro qsub command.

This step is necessary to allow you to proceed with next steps.

```
$ qsub -I -q qviz -A PROJECT_ID
```

In this example the default values for CPU cores and usage time are used.

```
$ qsub -I -q qviz -A PROJECT_ID -l select=1:ncpus=16 -l walltime=02:00:00
```

*Substitute **PROJECT_ID** with the assigned project identification string.*

In this example a whole node for 2 hours is requested.

If there are free resources for your request, you will have a shell unning on an assigned node. Please remember the name of the node.

```
$ uname -n  
srv8
```

In this example the visualization session was assigned to node **srv8**.

2. In your VNC session open another terminal (keep the one with interactive PBSPro job open).

Setup the VirtualGL connection to the node, which PBSPro allocated for our job.

```
$ vglconnect srv8
```

You will be connected with created VirtualGL tunnel to the visualization ode, where you will have a shell.

3. Load the VirtualGL module.

```
$ module load virtualgl/2.4
```

4. Run your desired OpenGL accelerated application using VirtualGL script “vglrun”.

```
$ vglrun glxgears
```

Please note, that if you want to run an OpenGL application which is vailable through modules, you need at first load the respective module. . g. to run the **Mentat** OpenGL application from **MARC** software ackage use:

```
$ module load marc/2013.1  
$ vglrun mentat
```

5. After you end your work with the OpenGL application.

Just logout from the visualization node and exit both opened terminals nd end your VNC server session as described above.

Tips and Tricks

If you want to increase the responsibility of the visualization, please adjust your TurboVNC client settings in this way:

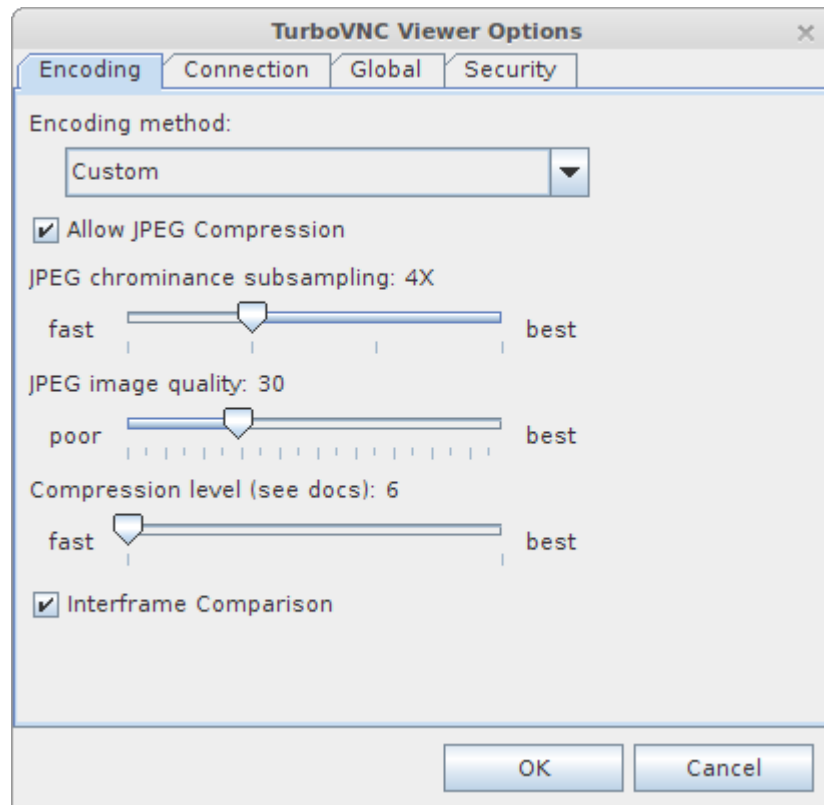


Figure 3: rem_vis_settings

To have an idea how the settings are affecting the resulting picture quality three levels of “JPEG image quality” are demonstrated:

1. JPEG image quality = 30
2. JPEG image quality = 15
3. JPEG image quality = 10

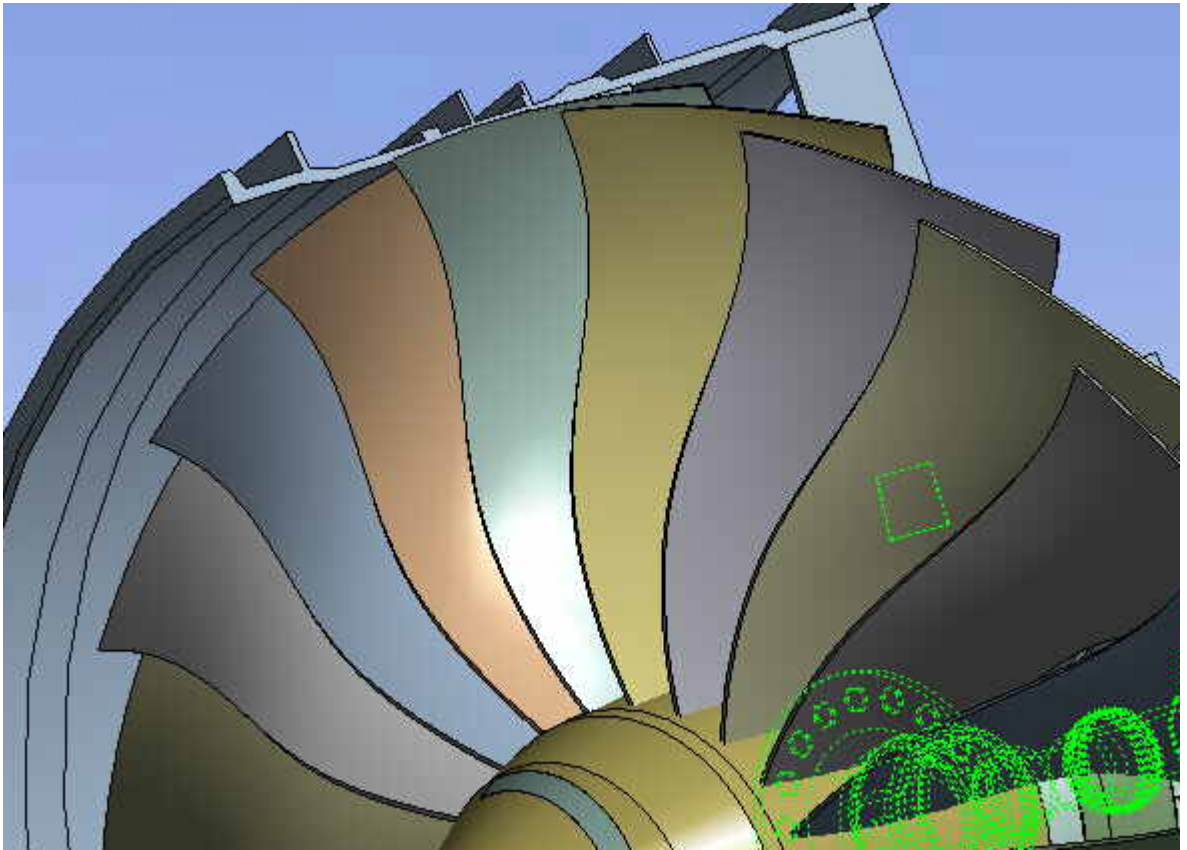


Figure 4: rem_vis_q3

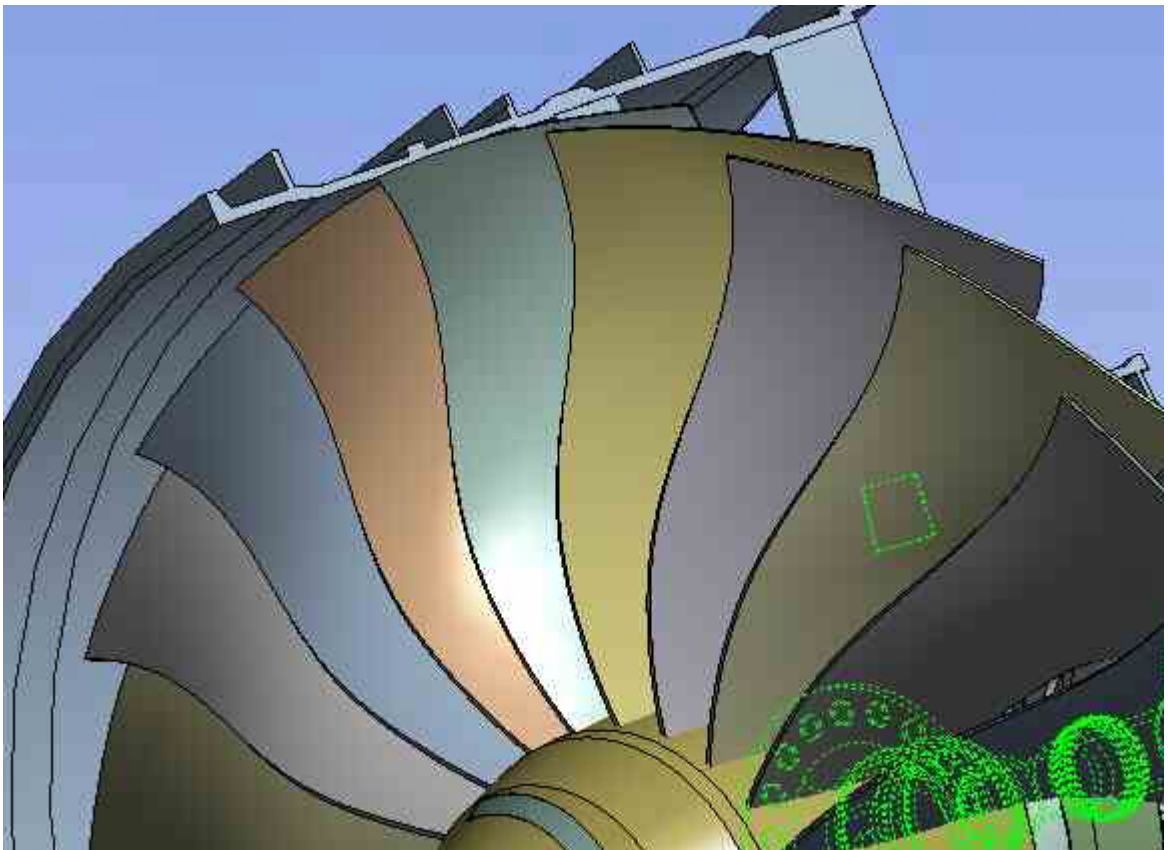


Figure 5: rem_vis_q2

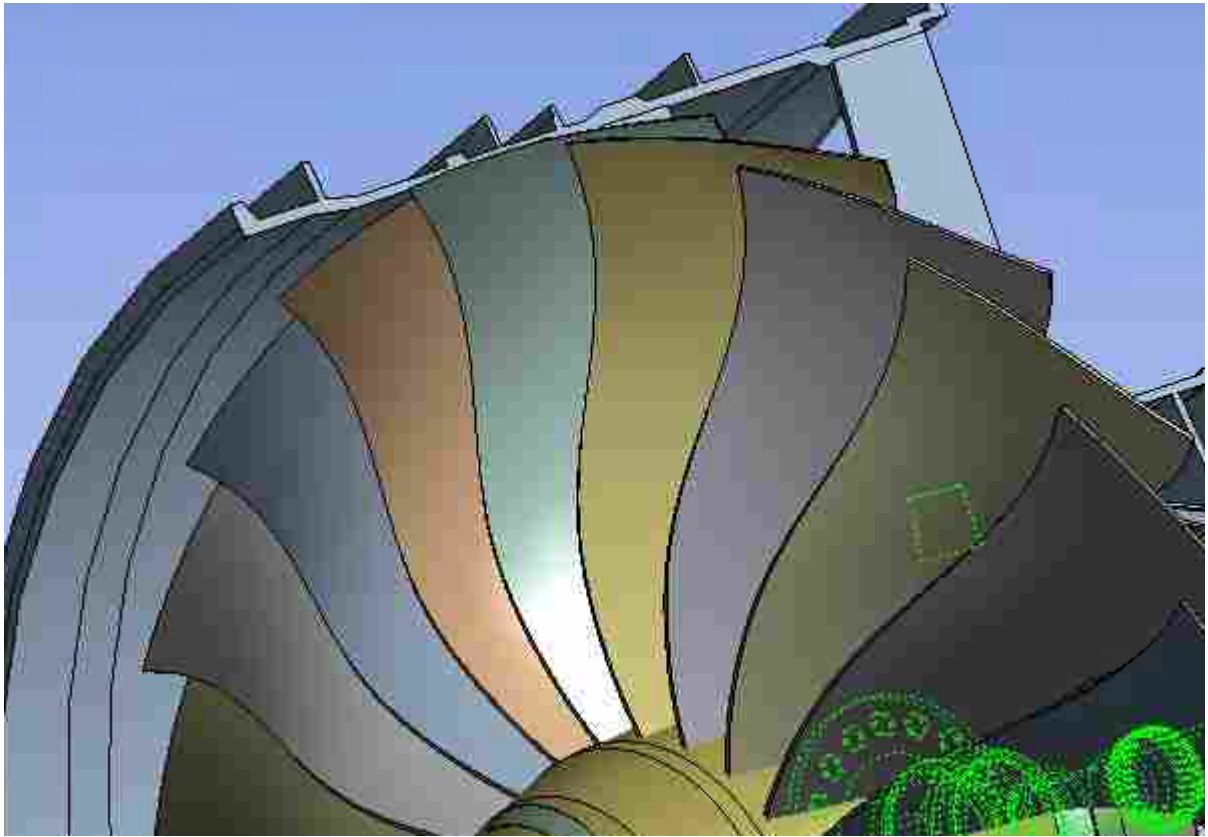


Figure 6: rem_vis_q1

Environment and Modules

Environment Customization

After logging in, you may want to configure the environment. Write your preferred path definitions, aliases, functions and module loads in the `.bashrc` file

```
# ./bashrc

# Source global definitions
if [ -f /etc/bashrc ]; then
    . /etc/bashrc
fi

# User specific aliases and functions
alias qs='qstat -a'
module load PrgEnv-gnu

# Display informations to standard output - only in interactive ssh session
if [ -n "$SSH_TTY" ]
then
    module list # Display loaded modules
fi
```

!!! Note “Note” Do not run commands outputting to standard output (echo, module list, etc) in `.bashrc` for non-interactive SSH sessions. It breaks fundamental functionality (scp, PBS) of your account! Take care for SSH session interactivity for such commands as stated in the previous example.

Application Modules

In order to configure your shell for running particular application on Anselm we use Module package interface.

!!! Note “Note” The modules set up the application paths, library paths and environment variables for running particular application.

We have also second modules repository. This modules repository is created using tool called E

The modules may be loaded, unloaded and switched, according to momentary needs.

To check available modules use

```
$ module avail
```

To load a module, for example the octave module use

```
$ module load octave
```

loading the octave module will set up paths and environment variables of your active shell such that you are ready to run the octave software

To check loaded modules use

```
$ module list
```

To unload a module, for example the octave module use

```
$ module unload octave
```

Learn more on modules by reading the module man page

```
$ man module
```

Following modules set up the development environment

PrgEnv-gnu sets up the GNU development environment in conjunction with the bullx MPI library

PrgEnv-intel sets up the INTEL development environment in conjunction with the Intel MPI library

Application Modules Path Expansion

All application modules on Salomon cluster (and further) will be build using tool called EasyBuild [\[1\]](#). In case that you want to use some applications that are build by EasyBuild already, you have to modify your MODULEPATH environment variable.

```
export MODULEPATH=$MODULEPATH:/apps/easybuild/modules/all/
```

This command expands your searched paths to modules. You can also add this command to the .bashrc file to expand paths permanently. After this command, you can use same commands to list/add/remove modules as is described above.

Resource Allocation and Job Execution

To run a job, computational resources for this particular job must be allocated. This is done via the PBS Pro job workload manager software, which efficiently distributes workloads across the supercomputer. Extensive informations about PBS Pro can be found in the official documentation [here](#), especially in the PBS Pro User's Guide.

Resources Allocation Policy

The resources are allocated to the job in a fairshare fashion, subject to constraints set by the queue and resources available to the Project. The Fairshare at Anselm ensures that individual users may consume approximately equal amount of resources per week. The resources are accessible via several queues for queueing the jobs. The queues provide prioritized and exclusive access to the computational resources. Following queues are available to Anselm users:

- **qexp**, the Express queue
- **qprod**, the Production queue
- **qlong**, the Long queue, regula
- **qnvvidia**, **qmic**, **qfat**, the Dedicated queues
- **qfree**, the Free resource utilization queue

!!! Note “Note” Check the queue status at <https://extranet.it4i.cz/anselm/>

Read more on the Resource AllocationPolicy page.

Job submission and execution

!!! Note “Note” Use the **qsub** command to submit your jobs.

The qsub submits the job into the queue. The qsub command creates a request to the PBS Job manager for allocation of specified resources. The **smallest allocation unit is entire node, 16 cores**, with exception of the qexp queue. The resources will be allocated when available, subject to allocation policies and constraints. **After the resources are allocated the jobscript or interactive shell is executed on first of the allocated nodes.**

Read more on the Job submission and execution page.

Capacity computing

!!! Note “Note” Use Job arrays when running huge number of jobs.

Use GNU Parallel and/or Job arrays when running (many) single core jobs.

In many cases, it is useful to submit huge (100+) number of computational jobs into the PBS queue system. Huge number of (small) jobs is one of the most effective ways to execute embarrassingly parallel calculations, achieving best runtime, throughput and computer utilization. In this chapter, we discuss the the recommended way to run huge number of jobs, including **ways to run huge number of single core jobs**.

Read more on Capacity computing page.

Capacity computing

Introduction

In many cases, it is useful to submit huge (>100+) number of computational jobs into the PBS queue system. Huge number of (small) jobs is one of the most effective ways to execute embarrassingly parallel calculations, achieving best runtime, throughput and computer utilization.

However, executing huge number of jobs via the PBS queue may strain the system. This strain may result in slow response to commands, inefficient scheduling and overall degradation of performance and user experience, for all users. For this reason, the number of jobs is **limited to 100 per user, 1000 per job array**

!!! Note “Note” Please follow one of the procedures below, in case you wish to schedule more than 100 jobs at a time.

- Use Job arrays when running huge number of multithread (bound to one node only) or multinode (multithread across several nodes) jobs
- Use GNU parallel when running single core jobs
- Combine GNU parallel with Job arrays when running huge number of single core jobs

Policy

1. A user is allowed to submit at most 100 jobs. Each job may be a job array.
2. The array size is at most 1000 subjobs.

Job arrays

!!! Note “Note” Huge number of jobs may be easily submitted and managed as a job array.

A job array is a compact representation of many jobs, called subjobs. The subjobs share the same job script, and have the same values for all attributes and resources, with the following exceptions:

- each subjob has a unique index, \$PBS_ARRAY_INDEX
- job Identifiers of subjobs only differ by their indices
- the state of subjobs can differ (R,Q,...etc.)

All subjobs within a job array have the same scheduling priority and schedule as independent jobs. Entire job array is submitted through a single qsub command and may be managed by qdel, qalter, qhold, qrls and qsig commands as a single job.

Shared jobscript

All subjobs in job array use the very same, single jobscript. Each subjob runs its own instance of the jobscript. The instances execute different work controlled by \$PBS_ARRAY_INDEX variable.

Example:

Assume we have 900 input files with name beginning with “file” (e. g. file001, ..., file900). Assume we would like to use each of these input files with program executable myprog.x, each as a separate job.

First, we create a tasklist file (or subjobs list), listing all tasks (subjobs) - all input files in our example:

```
$ find . -name 'file*' > tasklist
```

Then we create jobscript:

```
#!/bin/bash
#PBS -A PROJECT_ID
#PBS -q qprod
#PBS -l select=1:ncpus=16,walltime=02:00:00

# change to local scratch directory
SCR=/lscratch/$PBS_JOBID
mkdir -p $SCR ; cd $SCR || exit

# get individual tasks from tasklist with index from PBS JOB ARRAY
TASK=$(sed -n "${PBS_ARRAY_INDEX}p" $PBS_O_WORKDIR/tasklist)

# copy input file and executable to scratch
cp $PBS_O_WORKDIR/$TASK input ; cp $PBS_O_WORKDIR/myprog.x .

# execute the calculation
./myprog.x < input > output

# copy output file to submit directory
cp output $PBS_O_WORKDIR/$TASK.out
```

In this example, the submit directory holds the 900 input files, executable myprog.x and the jobscript file. As input for each run, we take the filename of input file from created tasklist file. We copy the input file to local scratch /lscratch/*PBS_JOBID*, executethemyprog.xandcopytheoutputfilebackto > thesubmitdire name. The myprog.x runs on one node only and must use threads to run in parallel. Be aware, that if the myprog.x is **not multithreaded**, then all the **jobs are run as single thread programs in sequential** manner. Due to allocation of the whole node, the accounted time is equal to the usage of whole node**, while using only 1/16 of the node!

If huge number of parallel multicore (in means of multinode multithread, e. g. MPI enabled) jobs is needed to run, then a job array approach should also be used. The main difference compared to previous example using one node is that the local scratch should not be used (as it's not shared between nodes) and MPI or other technique for parallel multinode run has to be used properly.

Submit the job array

To submit the job array, use the qsub -J command. The 900 jobs of the example above may be submitted like this:

```
$ qsub -N JOBNAME -J 1-900 jobscript
12345 [].dm2
```

In this example, we submit a job array of 900 subjobs. Each subjob will run on full node and is assumed to take less than 2 hours (please note the #PBS directives in the beginning of the jobscript file, dont' forget to set your valid PROJECT_ID and desired queue).

Sometimes for testing purposes, you may need to submit only one-element array. This is not allowed by PBSPro, but there's a workaround:

```
$ qsub -N JOBNAME -J 9-10:2 jobscript
```

This will only choose the lower index (9 in this example) for submitting/running your job.

Manage the job array

Check status of the job array by the qstat command.

```
$ qstat -a 12345[] .dm2
```

dm2:

Job ID	Username	Queue	Jobname	SessID	NDS	TSK	Req'd Memory	Req'd Time	Elap S	Time
12345[] .dm2	user2	qprod	xx	13516	1	16	--	00:50	B	00:02

The status B means that some subjobs are already running. Check status of the first 100 subjobs by the qstat command.

```
$ qstat -a 12345[1-100] .dm2
```

dm2:

Job ID	Username	Queue	Jobname	SessID	NDS	TSK	Req'd Memory	Req'd Time	Elap S	Time
12345[1] .dm2	user2	qprod	xx	13516	1	16	--	00:50	R	00:02
12345[2] .dm2	user2	qprod	xx	13516	1	16	--	00:50	R	00:02
12345[3] .dm2	user2	qprod	xx	13516	1	16	--	00:50	R	00:01
12345[4] .dm2	user2	qprod	xx	13516	1	16	--	00:50	Q	--
.
,
12345[100] .dm2	user2	qprod	xx	13516	1	16	--	00:50	Q	--

Delete the entire job array. Running subjobs will be killed, queueing subjobs will be deleted.

```
$ qdel 12345[] .dm2
```

Deleting large job arrays may take a while. Display status information for all user's jobs, job arrays, and subjobs.

```
$ qstat -u $USER -t
```

Display status information for all user's subjobs.

```
$ qstat -u $USER -tJ
```

Read more on job arrays in the PBSPro Users guide.

GNU parallel

!!! Note "Note" Use GNU parallel to run many single core tasks on one node.

GNU parallel is a shell tool for executing jobs in parallel using one or more computers. A job can be a single command or a small script that has to be run for each of the lines in the input. GNU parallel is most useful in running single core jobs via the queue system on Anselm.

For more information and examples see the parallel man page:

```
$ module add parallel
$ man parallel
```

GNU parallel jobscript

The GNU parallel shell executes multiple instances of the jobscript using all cores on the node. The instances execute different work, controlled by the \$PARALLEL_SEQ variable.

Example:

Assume we have 101 input files with name beginning with “file” (e. g. file001, ..., file101). Assume we would like to use each of these input files with program executable myprog.x, each as a separate single core job. We call these single core jobs tasks.

First, we create a tasklist file, listing all tasks - all input files in our example:

```
$ find . -name 'file*' > tasklist
```

Then we create jobscript:

```
#!/bin/bash
#PBS -A PROJECT_ID
#PBS -q qprod
#PBS -l select=1:ncpus=16,walltime=02:00:00

[ -z "$PARALLEL_SEQ" ] &&
{ module add parallel ; exec parallel -a $PBS_O_WORKDIR/tasklist $0 ; }

# change to local scratch directory
SCR=/lscratch/$PBS_JOBID/$PARALLEL_SEQ
mkdir -p $SCR ; cd $SCR || exit

# get individual task from tasklist
TASK=$1

# copy input file and executable to scratch
cp $PBS_O_WORKDIR/$TASK input

# execute the calculation
cat input > output

# copy output file to submit directory
cp output $PBS_O_WORKDIR/$TASK.out
```

In this example, tasks from tasklist are executed via the GNU parallel. The jobscript executes multiple instances of itself in parallel, on all cores of the node. Once an instance of jobscript is finished, new instance starts until all entries in tasklist are processed. Currently processed entry of the joblist may be retrieved via \$1 variable. Variable \$TASK expands to one of the input filenames from tasklist. We copy the input file to local scratch, execute the myprog.x and copy the output file back to the submit directory, under the \$TASK.out name.

Submit the job

To submit the job, use the qsub command. The 101 tasks' job of the example above may be submitted like this:

```
$ qsub -N JOBNAME jobscript
12345.dm2
```

In this example, we submit a job of 101 tasks. 16 input files will be processed in parallel. The 101 tasks on 16 cores are assumed to complete in less than 2 hours.

Please note the #PBS directives in the beginning of the jobscript file, don't forget to set your valid PROJECT_ID and desired queue.

Job arrays and GNU parallel

!!! Note “Note” Combine the Job arrays and GNU parallel for best throughput of single core jobs

While job arrays are able to utilize all available computational nodes, the GNU parallel can be used to efficiently run multiple single-core jobs on single node. The two approaches may be combined to utilize all available (current and future) resources to execute single core jobs.

!!! Note “Note” Every subjob in an array runs GNU parallel to utilize all cores on the node

GNU parallel, shared jobscript

Combined approach, very similar to job arrays, can be taken. Job array is submitted to the queuing system. The subjobs run GNU parallel. The GNU parallel shell executes multiple instances of the jobscript using all cores on the node. The instances execute different work, controlled by the \$PBS_JOB_ARRAY and \$PARALLEL_SEQ variables.

Example:

Assume we have 992 input files with name beginning with “file” (e. g. file001, ..., file992). Assume we would like to use each of these input files with program executable myprog.x, each as a separate single core job. We call these single core jobs tasks.

First, we create a tasklist file, listing all tasks - all input files in our example:

```
$ find . -name 'file*' > tasklist
```

Next we create a file, controlling how many tasks will be executed in one subjob

```
$ seq 32 > numtasks
```

Then we create jobscript:

```
#!/bin/bash
#PBS -A PROJECT_ID
#PBS -q qprod
#PBS -l select=1:ncpus=16,walltime=02:00:00

[ -z "$PARALLEL_SEQ" ] &&
{ module add parallel ; exec parallel -a $PBS_O_WORKDIR/numtasks $0 ; }

# change to local scratch directory
SCR=/lscratch/$PBS_JOBID/$PARALLEL_SEQ
mkdir -p $SCR ; cd $SCR || exit

# get individual task from tasklist with index from PBS JOB ARRAY and index from Parallel
IDX=$(( $PBS_ARRAY_INDEX + $PARALLEL_SEQ - 1 ))
TASK=$(sed -n "${IDX}p" $PBS_O_WORKDIR/tasklist)
[ -z "$TASK" ] && exit
```

```
# copy input file and executable to scratch
cp $PBS_O_WORKDIR/$TASK input

# execute the calculation
cat input > output

# copy output file to submit directory
cp output $PBS_O_WORKDIR/$TASK.out
```

In this example, the jobscript executes in multiple instances in parallel, on all cores of a computing node. Variable \$TASK expands to one of the input filenames from tasklist. We copy the input file to local scratch, execute the myprog.x and copy the output file back to the submit directory, under the \$TASK.out name. The numtasks file controls how many tasks will be run per subjob. Once an task is finished, new task starts, until the number of tasks in numtasks file is reached.

!!! Note “Note” Select subjob walltime and number of tasks per subjob carefully

When deciding this values, think about following guiding rules:

1. Let $n=N/16$. Inequality $(n+1) * T < W$ should hold. The N is number of tasks per subjob, T is expected single task walltime and W is subjob walltime. Short subjob walltime improves scheduling and job throughput.
2. Number of tasks should be modulo 16.
3. These rules are valid only when all tasks have similar task walltimes T.

Submit the job array

To submit the job array, use the qsub -J command. The 992 tasks’ job of the example above may be submitted like this:

```
$ qsub -N JOBNAME -J 1-992:32 jobscript
12345 [] .dm2
```

In this example, we submit a job array of 31 subjobs. Note the -J 1-992:**32**, this must be the same as the number sent to numtasks file. Each subjob will run on full node and process 16 input files in parallel, 32 in total per subjob. Every subjob is assumed to complete in less than 2 hours.

Please note the #PBS directives in the beginning of the jobscript file, dont’ forget to set your valid PROJECT_ID and desired queue.

Examples

Download the examples in capacity.zip, illustrating the above listed ways to run huge number of jobs. We recommend to try out the examples, before using this for running production jobs.

Unzip the archive in an empty directory on Anselm and follow the instructions in the README file

```
$ unzip capacity.zip
$ cat README
```

Resources Allocation Policy

Resources Allocation Policy

The resources are allocated to the job in a fairshare fashion, subject to constraints set by the queue and resources available to the Project. The Fairshare at Anselm ensures that individual users may consume approximately equal amount of resources per week. Detailed information in the Job scheduling section. The resources are accessible via several queues for queueing the jobs. The queues provide prioritized and exclusive access to the computational resources. Following table provides the queue partitioning overview:

queue	active project	project resources	nodes	min ncpus*	priority	authorization	>walltime			—										
—		qexp	no	none required	2 reserved, 31 total	including MIC, GPU and FAT nodes	1	>150	no	1h										
		qprod	yes	> 0	>178 nodes w/o accelerator	16	0	no	24/48h		qlong									
											Long queue									
											yes	> 0								
												60 nodes w/o accelerator								
												16	0	no	72/144h		qnvidia, qmic, qfat	Dedicated queues	yes	

!!! Note “Note” **The qfree queue is not free of charge.** Normal accounting applies. However, it allows for utilization of free resources, once a Project exhausted all its allocated computational resources. This does not apply for Directors Discretion’s projects (DD projects) by default. Usage of qfree after exhaustion of DD projects computational resources is allowed after request for this queue.

****The qexp queue is equipped with the nodes not having the very same CPU clock speed.** Should**

- **qexp**, the Express queue: This queue is dedicated for testing and running very small jobs. It is not required to specify a project to enter the qexp. There are 2 nodes always reserved for this queue (w/o accelerator), maximum 8 nodes are available via the qexp for a particular user, from a pool of nodes containing Nvidia accelerated nodes (cn181-203), MIC accelerated nodes (cn204-207) and Fat nodes with 512GB RAM (cn208-209). This enables to test and tune also accelerated code or code with higher RAM requirements. The nodes may be allocated on per core basis. No special authorization is required to use it. The maximum runtime in qexp is 1 hour.
- **qprod**, the Production queue: This queue is intended for normal production runs. It is required that active project with nonzero remaining resources is specified to enter the qprod. All nodes may be accessed via the qprod queue, except the reserved ones. 178 nodes without accelerator are included. Full nodes, 16 cores per node are allocated. The queue runs with medium priority and no special authorization is required to use it. The maximum runtime in qprod is 48 hours.
- **qlong**, the Long queue: This queue is intended for long production runs. It is required that active project with nonzero remaining resources is specified to enter the qlong. Only 60 nodes without acceleration may be accessed via the qlong queue. Full nodes, 16 cores per node are allocated. The queue runs with medium priority and no special authorization is required to use it. The maximum runtime in qlong is 144 hours (three times of the standard qprod time - 3 * 48 h).
- **qnvidia, qmic, qfat**, the Dedicated queues: The queue qnvidia is dedicated to access the Nvidia accelerated nodes, the qmic to access MIC nodes and qfat the Fat nodes. It is required that active project with nonzero remaining resources is specified to enter these queues. 23 nvidia, 4 mic and 2 fat nodes are included. Full nodes, 16 cores per node are allocated. The queues run with very high priority, the jobs will be scheduled before the jobs coming from the qexp queue. An PI needs explicitly ask support for authorization to enter the dedicated queues for all users associated to her/his Project.

- **qfree**, The Free resource queue: The queue qfree is intended for utilization of free resources, after a Project exhausted all its allocated computational resources (Does not apply to DD projects by default. DD projects have to request for permission on qfree after exhaustion of computational resources.). It is required that active project is specified to enter the queue, however no remaining resources are required. Consumed resources will be accounted to the Project. Only 178 nodes without accelerator may be accessed from this queue. Full nodes, 16 cores per node are allocated. The queue runs with very low priority and no special authorization is required to use it. The maximum runtime in qfree is 12 hours.


Notes

The job wall clock time defaults to **half the maximum time**, see table above. Longer wall time limits can be set manually, see examples.

Jobs that exceed the reserved wall clock time (Req'd Time) get killed automatically. Wall clock time limit can be changed for queuing jobs (state Q) using the qalter command, however can not be changed for a running job (state R).

Anselm users may check current queue configuration at <https://extranet.it4i.cz/anselm/queues>.

Queue status

Check the status of jobs, queues and compute nodes at <https://extranet.it4i.cz/anselm/>.

Display the queue status on Anselm:

```
$ qstat -q
```

The PBS allocation overview may be obtained also using the rspbs command.

```
$ rspbs
```

```
Usage: rspbs [options]
```

Options:

```
--version          show program's version number and exit
-h, --help         show this help message and exit
--get-node-ncpu-chart
                   Print chart of allocated ncpus per node
--summary          Print summary
--get-server-details
                   Print server
--get-queues       Print queues
--get-queues-details
                   Print queues details
--get-reservations
                   Print reservations
--get-reservations-details
                   Print reservations details
--get-nodes        Print nodes of PBS complex
--get-nodeset      Print nodeset of PBS complex
--get-nodes-details
                   Print nodes details
--get-jobs         Print jobs
--get-jobs-details
                   Print jobs details
--get-jobs-check-params
                   Print jobid, job state, session_id, user, nodes
--get-users        Print users of jobs
```


Cluster usage

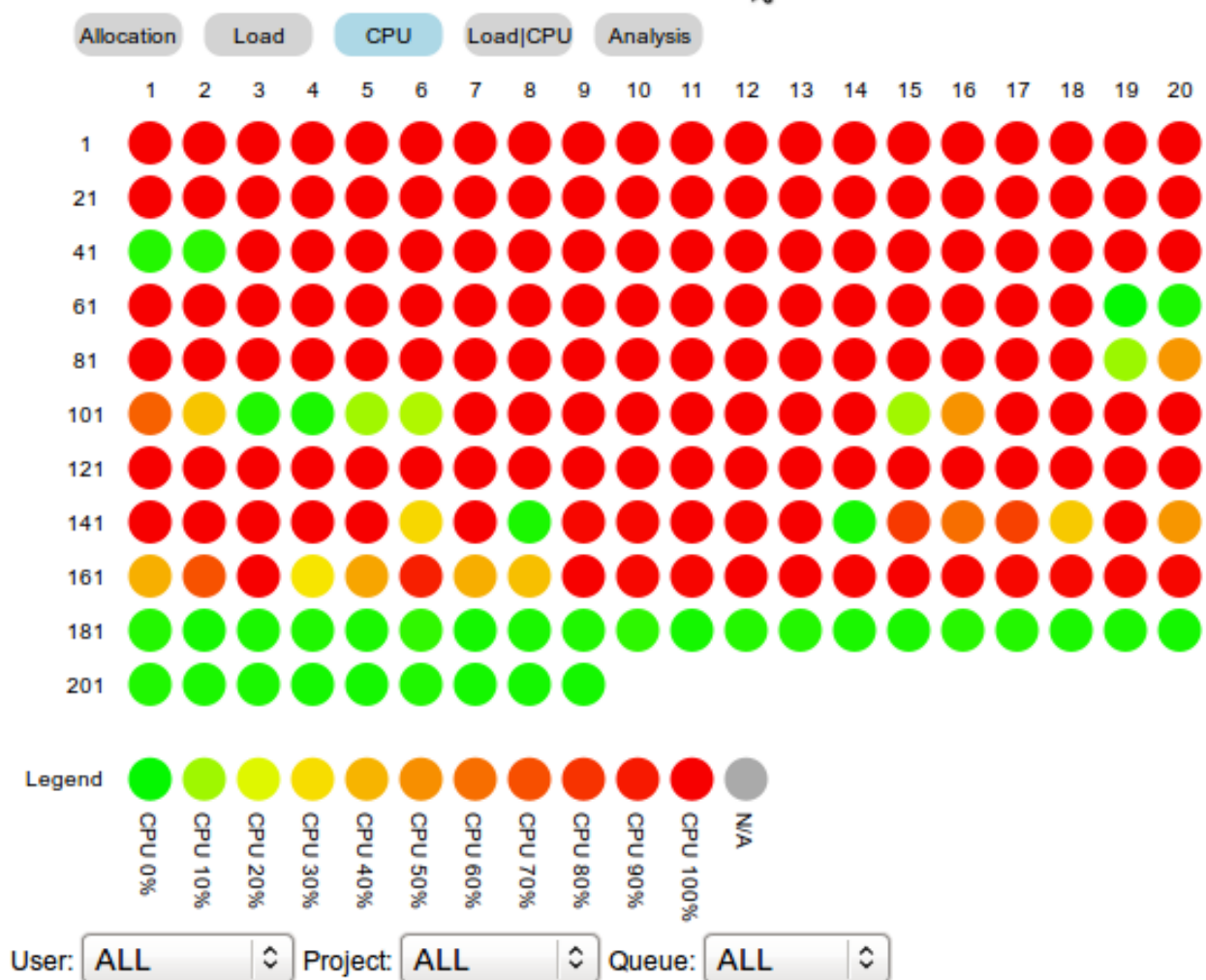


Figure 1: rspbs web interface

```

--get-allocated-nodes          Print allocated nodes of jobs
--get-allocated-nodeset       Print allocated nodeset of jobs
--get-node-users              Print node users
--get-node-jobs               Print node jobs
--get-node-ncpus              Print number of ncpus per node
--get-node-allocated-ncpus    Print number of allocated ncpus per node
--get-node-qlist              Print node qlist
--get-node-ibswitch           Print node ibswitch
--get-user-nodes              Print user nodes
--get-user-nodeset            Print user nodeset
--get-user-jobs               Print user jobs
--get-user-jobc               Print number of jobs per user
--get-user-nodect              Print number of allocated nodes per user
--get-user-ncpus              Print number of allocated ncpus per user
--get-qlist-nodes              Print qlist nodes
--get-qlist-nodeset           Print qlist nodeset
--get-ibswitch-nodes          Print ibswitch nodes
--get-ibswitch-nodeset        Print ibswitch nodeset
--state=STATE                  Only for given job state
--jobid=JOBID                  Only for given job ID
--user=USER                    Only for given user
--node=NODE                    Only for given node
--nodestate=NODESTATE          Only for given node state (affects only --get-node*
                                --get-qlist-* --get-ibswitch-* actions)
--incl-finished                Include finished jobs

```

Resources Accounting Policy

The Core-Hour

The resources that are currently subject to accounting are the core-hours. The core-hours are accounted on the wall clock basis. The accounting runs whenever the computational cores are allocated or blocked via the PBS Pro workload manager (the qsub command), regardless of whether the cores are actually used for any calculation. 1 core-hour is defined as 1 processor core allocated for 1 hour of wall clock time. Allocating a full node (16 cores) for 1 hour accounts to 16 core-hours. See example in the Job submission and execution section.

Check consumed resources

!!! Note “Note” The **it4ifree** command is a part of it4i.portal.clients package, located here: <https://pypi.python.org/pypi/it4i.portal.clients>

User may check at any time, how many core-hours have been consumed by himself/herself and his/her projects. The command is available on clusters’ login nodes.

```

$ it4ifree
Password:

```

PID	Total	Used	...by me	Free
OPEN-0-0	1500000	400644	225265	1099356
DD-13-1	10000	2606	2606	7394

Job submission and execution

Job Submission

When allocating computational resources for the job, please specify

1. suitable queue for your job (default is qprod)
2. number of computational nodes required
3. number of cores per node required
4. maximum wall time allocated to your calculation, note that jobs exceeding maximum wall time will be killed
5. Project ID
6. Jobscript or interactive switch

!!! Note “Note” Use the **qsub** command to submit your job to a queue for allocation of the computational resources.

Submit the job using the qsub command:

```
$ qsub -A Project_ID -q queue -l select=x:ncpus=y,walltime=[[hh:]mm:]ss[.ms] jobscript
```

The qsub submits the job into the queue, in another words the qsub command creates a request to the PBS Job manager for allocation of specified resources. The resources will be allocated when available, subject to above described policies and constraints. **After the resources are allocated the jobscript or interactive shell is executed on first of the allocated nodes.**

Job Submission Examples

```
$ qsub -A OPEN-0-0 -q qprod -l select=64:ncpus=16,walltime=03:00:00 ./myjob
```

In this example, we allocate 64 nodes, 16 cores per node, for 3 hours. We allocate these resources via the qprod queue, consumed resources will be accounted to the Project identified by Project ID OPEN-0-0. Jobscript myjob will be executed on the first node in the allocation.

```
$ qsub -q qexp -l select=4:ncpus=16 -I
```

In this example, we allocate 4 nodes, 16 cores per node, for 1 hour. We allocate these resources via the qexp queue. The resources will be available interactively

```
$ qsub -A OPEN-0-0 -q qnvidia -l select=10:ncpus=16 ./myjob
```

In this example, we allocate 10 nvidia accelerated nodes, 16 cores per node, for 24 hours. We allocate these resources via the qnvidia queue. Jobscript myjob will be executed on the first node in the allocation.

```
$ qsub -A OPEN-0-0 -q qfree -l select=10:ncpus=16 ./myjob
```

In this example, we allocate 10 nodes, 16 cores per node, for 12 hours. We allocate these resources via the qfree queue. It is not required that the project OPEN-0-0 has any available resources left. Consumed resources are still accounted for. Jobscript myjob will be executed on the first node in the allocation.

All qsub options may be saved directly into the jobscript. In such a case, no options to qsub are needed.

```
$ qsub ./myjob
```

By default, the PBS batch system sends an e-mail only when the job is aborted. Disabling mail events completely can be done like this:

```
$ qsub -m n
```

Advanced job placement

Placement by name

Specific nodes may be allocated via the PBS

```
qsub -A OPEN-0-0 -q qprod -l select=1:ncpus=16:host=cn171+1:ncpus=16:host=cn172 -I
```

In this example, we allocate nodes cn171 and cn172, all 16 cores per node, for 24 hours. Consumed resources will be accounted to the Project identified by Project ID OPEN-0-0. The resources will be available interactively.

Placement by CPU type

Nodes equipped with Intel Xeon E5-2665 CPU have base clock frequency 2.4GHz, nodes equipped with Intel Xeon E5-2470 CPU have base frequency 2.3 GHz (see section Compute Nodes for details). Nodes may be selected via the PBS resource attribute `cpu_freq`.

CPU Type	base freq.
Intel Xeon E5-2665	2.4GHz
Intel Xeon E5-2470	2.3GHz

```
$ qsub -A OPEN-0-0 -q qprod -l select=4:ncpus=16:cpu_freq=24 -I
```

In this example, we allocate 4 nodes, 16 cores, selecting only the nodes with Intel Xeon E5-2665 CPU.

Placement by IB switch

Groups of computational nodes are connected to chassis integrated Infiniband switches. These switches form the leaf switch layer of the Infiniband network fat tree topology. Nodes sharing the leaf switch can communicate most efficiently. Sharing the same switch prevents hops in the network and provides for unbiased, most efficient network communication.

Nodes sharing the same switch may be selected via the PBS resource attribute `ibswitch`. Values of this attribute are `iswXX`, where `XX` is the switch number. The node-switch mapping can be seen at Hardware Overview section.

We recommend allocating compute nodes of a single switch when best possible computational network performance is required to run the job efficiently:

```
qsub -A OPEN-0-0 -q qprod -l select=18:ncpus=16:ibswitch=isw11 ./myjob
```

In this example, we request all the 18 nodes sharing the isw11 switch for 24 hours. Full chassis will be allocated.

Advanced job handling

Selecting Turbo Boost off

Intel Turbo Boost Technology is on by default. We strongly recommend keeping the default.

If necessary (such as in case of benchmarking) you can disable the Turbo for all nodes of the job by using the PBS resource attribute `cpu_turbo_boost`

```
$ qsub -A OPEN-0-0 -q qprod -l select=4:ncpus=16 -l cpu_turbo_boost=0 -I
```

More about the Intel Turbo Boost in the TurboBoost section

Advanced examples

In the following example, we select an allocation for benchmarking a very special and demanding MPI program. We request Turbo off, 2 full chassis of compute nodes (nodes sharing the same IB switches) for 30 minutes:

```
$ qsub -A OPEN-0-0 -q qprod
-l select=18:ncpus=16:ibswitch=isw10:mpiprocs=1:ompthreads=16+18:ncpus=16:ibswitch=isw20
-l cpu_turbo_boost=0,walltime=00:30:00
-N Benchmark ./mybenchmark
```

The MPI processes will be distributed differently on the nodes connected to the two switches. On the isw10 nodes, we will run 1 MPI process per node 16 threads per process, on isw20 nodes we will run 16 plain MPI processes.

Although this example is somewhat artificial, it demonstrates the flexibility of the `qsub` command options.

Job Management

!!! Note “Note” Check status of your jobs using the `qstat` and `check-pbs-jobs` commands

```
$ qstat -a
$ qstat -a -u username
$ qstat -an -u username
$ qstat -f 12345.srv11
```

Example:

```
$ qstat -a
```

srv11:

Job ID	Username	Queue	Jobname	SessID	NDS	TSK	Req'd Memory	Req'd Time	Elap S	Time
16287.srv11	user1	qlong	job1	6183	4	64	--	144:0	R	38:25
16468.srv11	user1	qlong	job2	8060	4	64	--	144:0	R	17:44
16547.srv11	user2	qprod	job3x	13516	2	32	--	48:00	R	00:58

In this example user1 and user2 are running jobs named job1, job2 and job3x. The jobs job1 and job2 are using 4 nodes, 16 cores per node each. The job1 already runs for 38 hours and 25 minutes, job2 for 17 hours 44 minutes. The job1 already consumed $64 \times 38.41 = 2458.6$ core hours. The job3x already consumed $0.9632 = 30.93$ core hours. These consumed core hours will be accounted on

the respective project accounts, regardless of whether the allocated cores were actually used for computations.

Check status of your jobs using `check-pbs-jobs` command. Check presence of user's PBS jobs' processes on execution hosts. Display load, processes. Display job standard and error output. Continuously display (`tail -f`) job standard or error output.

```
$ check-pbs-jobs --check-all
$ check-pbs-jobs --print-load --print-processes
$ check-pbs-jobs --print-job-out --print-job-err

$ check-pbs-jobs --jobid JOBID --check-all --print-all

$ check-pbs-jobs --jobid JOBID --tailf-job-out
```

Examples:

```
$ check-pbs-jobs --check-all
JOB 35141.dm2, session_id 71995, user user2, nodes cn164,cn165
Check session id: OK
Check processes
cn164: OK
cn165: No process
```

In this example we see that job 35141.dm2 currently runs no process on allocated node cn165, which may indicate an execution error.

```
$ check-pbs-jobs --print-load --print-processes
JOB 35141.dm2, session_id 71995, user user2, nodes cn164,cn165
Print load
cn164: LOAD: 16.01, 16.01, 16.00
cn165: LOAD: 0.01, 0.00, 0.01
Print processes
      %CPU CMD
cn164: 0.0 -bash
cn164: 0.0 /bin/bash /var/spool/PBS/mom_priv/jobs/35141.dm2.SC
cn164: 99.7 run-task
...
```

In this example we see that job 35141.dm2 currently runs process run-task on node cn164, using one thread only, while node cn165 is empty, which may indicate an execution error.

```
$ check-pbs-jobs --jobid 35141.dm2 --print-job-out
JOB 35141.dm2, session_id 71995, user user2, nodes cn164,cn165
Print job standard output:
===== Job start =====
Started at      : Fri Aug 30 02:47:53 CEST 2013
Script name     : script
Run loop 1
Run loop 2
Run loop 3
```

In this example, we see actual output (some iteration loops) of the job 35141.dm2

!!! Note “Note” Manage your queued or running jobs, using the **qhold**, **qrls**, **qdel**, **qsig** or **qalter** commands

You may release your allocation at any time, using `qdel` command

```
$ qdel 12345.srv11
```

You may kill a running job by force, using qsig command

```
$ qsig -s 9 12345.srv11
```

Learn more by reading the pbs man page

```
$ man pbs_professional
```

Job Execution

Jobscript

!!! Note “Note” Prepare the jobscript to run batch jobs in the PBS queue system

The Jobscript is a user made script, controlling sequence of commands for executing the calculation. It is often written in bash, other scripts may be used as well. The jobscript is supplied to PBS **qsub** command as an argument and executed by the PBS Professional workload manager.

!!! Note “Note” The jobscript or interactive shell is executed on first of the allocated nodes.

```
$ qsub -q qexp -l select=4:ncpus=16 -N Name0 ./myjob
```

```
$ qstat -n -u username
```

```
srv11:
```

Job ID	Username	Queue	Jobname	SessID	NDS	TSK	Req'd Memory	Req'd Time	Elap S	Time
15209.srv11	username	qexp	Name0	5530	4	64	--	01:00	R	00:00
cn17/0*16+cn108/0*16+cn109/0*16+cn110/0*16										

In this example, the nodes cn17, cn108, cn109 and cn110 were allocated for 1 hour via the qexp queue. The jobscript myjob will be executed on the node cn17, while the nodes cn108, cn109 and cn110 are available for use as well.

The jobscript or interactive shell is by default executed in home directory

```
$ qsub -q qexp -l select=4:ncpus=16 -I
qsub: waiting for job 15210.srv11 to start
qsub: job 15210.srv11 ready
```

```
$ pwd
/home/username
```

In this example, 4 nodes were allocated interactively for 1 hour via the qexp queue. The interactive shell is executed in the home directory.

!!! Note “Note” All nodes within the allocation may be accessed via ssh. Unallocated nodes are not accessible to user.

The allocated nodes are accessible via ssh from login nodes. The nodes may access each other via ssh as well.

Calculations on allocated nodes may be executed remotely via the MPI, ssh, pdsh or clush. You may find out which nodes belong to the allocation by reading the \$PBS_NODEFILE file

```
qsub -q qexp -l select=4:ncpus=16 -I
qsub: waiting for job 15210.srv11 to start
```



```
qsub: job 15210.srv11 ready
```

```
$ pwd  
/home/username
```

```
$ sort -u $PBS_NODEFILE  
cn17.bullx  
cn108.bullx  
cn109.bullx  
cn110.bullx
```

```
$ pdsh -w cn17,cn[108-110] hostname  
cn17: cn17  
cn108: cn108  
cn109: cn109  
cn110: cn110
```

In this example, the hostname program is executed via pdsh from the interactive shell. The execution runs on all four allocated nodes. The same result would be achieved if the pdsh is called from any of the allocated nodes or from the login nodes.

Example Jobscript for MPI Calculation

!!! Note “Note” Production jobs must use the /scratch directory for I/O

The recommended way to run production jobs is to change to /scratch directory early in the jobscript, copy all inputs to /scratch, execute the calculations and copy outputs to home directory.

```
#!/bin/bash  
  
# change to scratch directory, exit on failure  
SCRDIR=/scratch/$USER/myjob  
mkdir -p $SCRDIR  
cd $SCRDIR || exit  
  
# copy input file to scratch  
cp $PBS_O_WORKDIR/input .  
cp $PBS_O_WORKDIR/mympiprogram.x .  
  
# load the mpi module  
module load openmpi  
  
# execute the calculation  
mpiexec -pernode ./mympiprogram.x  
  
# copy output file to home  
cp output $PBS_O_WORKDIR/.  
  
#exit  
exit
```

In this example, some directory on the /home holds the input file input and executable mympiprogram.x . We create a directory myjob on the /scratch filesystem, copy input and executable files from the /home directory where the qsub was invoked (\$PBS_O_WORKDIR) to /scratch, execute the MPI

programm mympiprogram.x and copy the output file back to the /home directory. The mympiprogram.x is executed as one process per node, on all allocated nodes.

!!! Note “Note” Consider preloading inputs and executables onto shared scratch before the calculation starts.

In some cases, it may be impractical to copy the inputs to scratch and outputs to home. This is especially true when very large input and output files are expected, or when the files should be reused by a subsequent calculation. In such a case, it is users responsibility to preload the input files on shared /scratch before the job submission and retrieve the outputs manually, after all calculations are finished.

!!! Note “Note” Store the qsub options within the jobscript. Use **mpiprocs** and **ompthreads** qsub options to control the MPI job execution.

Example jobscript for an MPI job with preloaded inputs and executables, options for qsub are stored within the script :

```
#!/bin/bash
#PBS -q qprod
#PBS -N MYJOB
#PBS -l select=100:ncpus=16:mpiprocs=1:ompthreads=16
#PBS -A OPEN-0-0

# change to scratch directory, exit on failure
SCRDIR=/scratch/$USER/myjob
cd $SCRDIR || exit

# load the mpi module
module load openmpi

# execute the calculation
mpiexec ./mympiprogram.x

#exit
exit
```

In this example, input and executable files are assumed preloaded manually in /scratch/\$USER/myjob directory. Note the **mpiprocs** and **ompthreads** qsub options, controlling behavior of the MPI execution. The mympiprogram.x is executed as one process per node, on all 100 allocated nodes. If mympiprogram.x implements OpenMP threads, it will run 16 threads per node.

More information is found in the Running OpenMPI and Running MPICH2 sections.

Example Jobscript for Single Node Calculation

!!! Note “Note” Local scratch directory is often useful for single node jobs. Local scratch will be deleted immediately after the job ends.

Example jobscript for single node calculation, using local scratch on the node:

```
#!/bin/bash

# change to local scratch directory
cd /lscratch/$PBS_JOBID || exit
```

```

# copy input file to scratch
cp $PBS_O_WORKDIR/input .
cp $PBS_O_WORKDIR/myprog.x .

# execute the calculation
./myprog.x

# copy output file to home
cp output $PBS_O_WORKDIR/.

#exit
exit

```

In this example, some directory on the home holds the input file input and executable myprog.x. We copy input and executable files from the home directory where the qsub was invoked ($PBS_{O_w} ORKDIR$) to local scratch/lscratch/PBS_JOBID, execute the myprog.x and copy the output file back to the /home directory. The myprog.x runs on one node only and may use threads.

Other Jobscript Examples

Further jobscript examples may be found in the software section and the Capacity computing section.

Job scheduling

Job execution priority

Scheduler gives each job an execution priority and then uses this job execution priority to select which job(s) to run.

Job execution priority on Anselm is determined by these job properties (in order of importance):

1. queue priority
2. fairshare priority
3. eligible time

Queue priority

Queue priority is priority of queue where job is queued before execution.

Queue priority has the biggest impact on job execution priority. Execution priority of jobs in higher priority queues is always greater than execution priority of jobs in lower priority queues. Other properties of job used for determining job execution priority (fairshare priority, eligible time) cannot compete with queue priority.

Queue priorities can be seen at <https://extranet.it4i.cz/anselm/queues>

Fairshare priority

Fairshare priority is priority calculated on recent usage of resources. Fairshare priority is calculated per project, all members of project share same fairshare priority. Projects with higher recent usage have lower fairshare priority than projects with lower or none recent usage.

Fairshare priority is used for ranking jobs with equal queue priority.

Fairshare priority is calculated as

$$MAX_FAIRSHARE * (1 - \frac{usage_{Project}}{usage_{Total}})$$

Figure 1:

where MAX_FAIRSHARE has value 1E6, usage_{Project} is cumulated usage by all members of selected project, usage_{Total} is total usage by all users, by all projects.

Usage counts allocated corehours (ncpus*walltime). Usage is decayed, or cut in half periodically, at the interval 168 hours (one week). Jobs queued in queue qexp are not calculated to project's usage.

Calculated usage and fairshare priority can be seen at <https://extranet.it4i.cz/anselm/projects>.

Calculated fairshare priority can be also seen as Resource_List.fairshare attribute of a job.

Eligible time

Eligible time is amount (in seconds) of eligible time job accrued while waiting to run. Jobs with higher eligible time gains higher priority.

Eligible time has the least impact on execution priority. Eligible time is used for sorting jobs with equal queue priority and fairshare priority. It is very, very difficult for eligible time to compete with fairshare priority.

Eligible time can be seen as `eligible_time` attribute of job.

Formula

Job execution priority (job sort formula) is calculated as:

$$1000 * \text{queue_priority} + \frac{\text{fairshare_priority}}{1000} + \frac{\text{eligible_time}}{864000}$$

Figure 2:

Job backfilling

Anselm cluster uses job backfilling.

Backfilling means fitting smaller jobs around the higher-priority jobs that the scheduler is going to run next, in such a way that the higher-priority jobs are not delayed. Backfilling allows us to keep resources from becoming idle when the top job (job with the highest execution priority) cannot run.



The scheduler makes a list of jobs to run in order of execution priority. Scheduler looks for smaller jobs that can fit into the usage gaps around the highest-priority jobs in the list. The scheduler looks in the prioritized list of jobs and chooses the highest-priority smaller jobs that fit. Filler jobs are run only if they will not delay the start time of top jobs.

It means, that jobs with lower execution priority can be run before jobs with higher execution priority.


!!! Note “Note” It is **very beneficial to specify the walltime** when submitting jobs.

Specifying more accurate walltime enables better scheduling, better execution times and better resource usage. Jobs with suitable (small) walltime could be backfilled - and overtake job(s) with higher priority.

Network

All compute and login nodes of Anselm are interconnected by Infiniband  QDR network and by Gigabit Ethernet  network. Both networks may be used to transfer user data.

Infiniband Network

All compute and login nodes of Anselm are interconnected by a high-bandwidth, low-latency Infiniband  QDR network (IB 4x QDR, 40 Gbps). The network topology is a fully non-blocking fat-tree.

The compute nodes may be accessed via the Infiniband network using ib0 network interface, in address range 10.2.1.1-209. The MPI may be used to establish native Infiniband connection among the nodes.

!!! Note “Note” The network provides **2170MB/s** transfer rates via the TCP connection (single stream) and up to **3600MB/s** via native Infiniband protocol.

The Fat tree topology ensures that peak transfer rates are achieved between any two nodes, independent of network traffic exchanged among other nodes concurrently.

Ethernet Network

The compute nodes may be accessed via the regular Gigabit Ethernet network interface eth0, in address range 10.1.1.1-209, or by using aliases cn1-cn209. The network provides **114MB/s** transfer rates via the TCP connection.

Example

```
$ qsub -q qexp -l select=4:ncpus=16 -N Name0 ./myjob
$ qstat -n -u username
```

Job ID	Username	Queue	Jobname	SessID	NDS	TSK	Req'd Memory	Req'd Time	Elap S	Time
15209.srv11	username	qexp	Name0	5530	4	64	--	01:00	R	00:00
cn17/0*16+cn108/0*16+cn109/0*16+cn110/0*16										

```
$ ssh 10.2.1.110
$ ssh 10.1.1.108
```

In this example, we access the node cn110 by Infiniband network via the ib0 interface, then from cn110 to cn108 by Ethernet network.

CESNET Data Storage

Introduction

Do not use shared filesystems at IT4Innovations as a backup for large amount of data or long-term archiving purposes.

!!! Note “Note” The IT4Innovations does not provide storage capacity for data archiving. Academic staff and students of research institutions in the Czech Republic can use CESNET Storage service [\[1\]](#).

The CESNET Storage service can be used for research purposes, mainly by academic staff and students of research institutions in the Czech Republic.

User of data storage CESNET (DU) association can become organizations or an individual person who is either in the current employment relationship (employees) or the current study relationship (students) to a legal entity (organization) that meets the “Principles for access to CESNET Large infrastructure (Access Policy)”.

User may only use data storage CESNET for data transfer and storage which are associated with activities in science, research, development, the spread of education, culture and prosperity. In detail see “Acceptable Use Policy CESNET Large Infrastructure (Acceptable Use Policy, AUP)”.

The service is documented at <https://du.cesnet.cz/wiki/doku.php/en/start> [\[2\]](#). For special requirements please contact directly CESNET Storage Department via e-mail du-support@cesnet.cz.

The procedure to obtain the CESNET access is quick and trouble-free.

(source <https://du.cesnet.cz/> [\[3\]](#))

CESNET storage access

Understanding Cesnet storage

!!! Note “Note” It is very important to understand the Cesnet storage before uploading data. Please read <https://du.cesnet.cz/en/navody/home-migrace-plzen/start> [\[4\]](#) first.

Once registered for CESNET Storage, you may access the storage [\[5\]](#) in number of ways. We recommend the SSHFS and RSYNC methods.

SSHFS Access

!!! Note “Note” SSHFS: The storage will be mounted like a local hard drive

The SSHFS provides a very convenient way to access the CESNET Storage. The storage will be mounted onto a local directory, exposing the vast CESNET Storage as if it was a local removable harddrive. Files can be then copied in and out in a usual fashion.

First, create the mountpoint

```
$ mkdir cesnet
```

Mount the storage. Note that you can choose among the [ssh.du1.cesnet.cz](#) (Plzen), [ssh.du2.cesnet.cz](#) (Jihlava), [ssh.du3.cesnet.cz](#) (Brno) Mount tier1_home (**only 5120M !**):

```
$ sshfs username@ssh.du1.cesnet.cz:. cesnet/
```

For easy future access from Anselm, install your public key

```
$ cp .ssh/id_rsa.pub cesnet/.ssh/authorized_keys
```

Mount tier1_cache_tape for the Storage VO:

```
$ sshfs username@ssh.du1.cesnet.cz:/cache_tape/VO_storage/home/username cesnet/
```

View the archive, copy the files and directories in and out

```
$ ls cesnet/  
$ cp -a mydir cesnet/.  
$ cp cesnet/myfile .
```

Once done, please remember to unmount the storage

```
$ fusermount -u cesnet
```

Rsync access

!!! Note “Note” Rsync provides delta transfer for best performance, can resume interrupted transfers

Rsync is a fast and extraordinarily versatile file copying tool. It is famous for its delta-transfer algorithm, which reduces the amount of data sent over the network by sending only the differences between the source files and the existing files in the destination. Rsync is widely used for backups and mirroring and as an improved copy command for everyday use.

Rsync finds files that need to be transferred using a “quick check” algorithm (by default) that looks for files that have changed in size or in last-modified time. Any changes in the other preserved attributes (as requested by options) are made on the destination file directly when the quick check indicates that the file’s data does not need to be updated.

More about Rsync at https://du.cesnet.cz/en/navody/rsync/start#pro_bezne_uzivatele 

Transfer large files to/from Cesnet storage, assuming membership in the Storage VO

```
$ rsync --progress datafile username@ssh.du1.cesnet.cz:VO_storage-cache_tape/.  
$ rsync --progress username@ssh.du1.cesnet.cz:VO_storage-cache_tape/datafile .
```

Transfer large directories to/from Cesnet storage, assuming membership in the Storage VO

```
$ rsync --progress -av datafolder username@ssh.du1.cesnet.cz:VO_storage-cache_tape/.  
$ rsync --progress -av username@ssh.du1.cesnet.cz:VO_storage-cache_tape/datafolder .
```

Transfer rates of about 28MB/s can be expected.

Storage

There are two main shared file systems on Anselm cluster, the HOME and SCRATCH. All login and compute nodes may access same data on shared filesystems. Compute nodes are also equipped with local (non-shared) scratch, ramdisk and tmp filesystems.

Archiving

Please don't use shared filesystems as a backup for large amount of data or long-term archiving mean. The academic staff and students of research institutions in the Czech Republic can use CESNET storage service, which is available via SSHFS.

Shared Filesystems

Anselm computer provides two main shared filesystems, the HOME filesystem and the SCRATCH filesystem. Both HOME and SCRATCH filesystems are realized as a parallel Lustre filesystem. Both shared file systems are accessible via the Infiniband network. Extended ACLs are provided on both Lustre filesystems for the purpose of sharing data with other users using fine-grained control.

Understanding the Lustre Filesystems

(source <http://www.nas.nasa.gov>)

A user file on the Lustre filesystem can be divided into multiple chunks (stripes) and stored across a subset of the object storage targets (OSTs) (disks). The stripes are distributed among the OSTs in a round-robin fashion to ensure load balancing.

When a client (a compute node from your job) needs to create or access a file, the client queries the metadata server (MDS) and the metadata target (MDT) for the layout and location of the file's stripes. Once the file is opened and the client obtains the striping information, the MDS is no longer involved in the file I/O process. The client interacts directly with the object storage servers (OSSes) and OSTs to perform I/O operations such as locking, disk allocation, storage, and retrieval.

If multiple clients try to read and write the same part of a file at the same time, the Lustre distributed lock manager enforces coherency so that all clients see consistent results.

There is default stripe configuration for Anselm Lustre filesystems. However, users can set the following stripe parameters for their own directories or files to get optimum I/O performance:

1. `stripe_size`: the size of the chunk in bytes; specify with k, m, or g to use units of KB, MB, or GB, respectively; the size must be an even multiple of 65,536 bytes; default is 1MB for all Anselm Lustre filesystems
2. `stripe_count` the number of OSTs to stripe across; default is 1 for Anselm Lustre filesystems one can specify -1 to use all OSTs in the filesystem.
3. `stripe_offset` The index of the OST where the first stripe is to be placed; default is -1 which results in random selection; using a non-default value is NOT recommended.

!!! Note "Note" Setting stripe size and stripe count correctly for your needs may significantly impact the I/O performance you experience.

Use the `lfs getstripe` for getting the stripe parameters. Use the `lfs setstripe` command for setting the stripe parameters to get optimal I/O performance. The correct stripe setting depends on your needs and file access patterns.

```
$ lfs getstripe dir|filename
$ lfs setstripe -s stripe_size -c stripe_count -o stripe_offset dir|filename
```

Example:

```
$ lfs getstripe /scratch/username/
/scratch/username/
stripe_count: 1 stripe_size: 1048576 stripe_offset: -1

$ lfs setstripe -c -1 /scratch/username/
$ lfs getstripe /scratch/username/
/scratch/username/
stripe_count: 10 stripe_size: 1048576 stripe_offset: -1
```

In this example, we view current stripe setting of the `/scratch/username/` directory. The stripe count is changed to all OSTs, and verified. All files written to this directory will be striped over 10 OSTs.

Use `lfs check OSTs` to see the number and status of active OSTs for each filesystem on Anselm. Learn more by reading the man page.

```
$ lfs check osts
$ man lfs
```

Hints on Lustre Stripping

!!! Note “Note” Increase the `stripe_count` for parallel I/O to the same file.

When multiple processes are writing blocks of data to the same file in parallel, the I/O performance for large files will improve when the `stripe_count` is set to a larger value. The stripe count sets the number of OSTs the file will be written to. By default, the stripe count is set to 1. While this default setting provides for efficient access of metadata (for example to support the `ls -l` command), large files should use stripe counts of greater than 1. This will increase the aggregate I/O bandwidth by using multiple OSTs in parallel instead of just one. A rule of thumb is to use a stripe count approximately equal to the number of gigabytes in the file.

Another good practice is to make the stripe count be an integral factor of the number of processes performing the write in parallel, so that you achieve load balance among the OSTs. For example, set the stripe count to 16 instead of 15 when you have 64 processes performing the writes.

!!! Note “Note” Using a large stripe size can improve performance when accessing very large files.

Large stripe size allows each client to have exclusive access to its own part of a file. However, it can be counterproductive in some cases if it does not match your I/O pattern. The choice of stripe size has no effect on a single-stripe file.

Read more on http://wiki.lustre.org/manual/LustreManual20_HTML/ManagingStripingFreeSpace.html 

Lustre on Anselm

The architecture of Lustre on Anselm is composed of two metadata servers (MDS) and four data/object storage servers (OSS). Two object storage servers are used for file system HOME and

another two object storage servers are used for file system SCRATCH.

Configuration of the storages

- HOME Lustre object storage
 - One disk array NetApp E5400
 - 22 OSTs
 - 227 2TB NL-SAS 7.2krpm disks
 - 22 groups of 10 disks in RAID6 (8+2)
 - 7 hot-spare disks
- SCRATCH Lustre object storage
 - Two disk arrays NetApp E5400
 - 10 OSTs
 - 106 2TB NL-SAS 7.2krpm disks
 - 10 groups of 10 disks in RAID6 (8+2)
 - 6 hot-spare disks
- Lustre metadata storage
 - One disk array NetApp E2600
 - 12 300GB SAS 15krpm disks
 - 2 groups of 5 disks in RAID5
 - 2 hot-spare disks

HOME

The HOME filesystem is mounted in directory /home. Users home directories /home/username reside on this filesystem. Accessible capacity is 320TB, shared among all users. Individual users are restricted by filesystem usage quotas, set to 250GB per user. If 250GB should prove as insufficient for particular user, please contact support [\[4\]](#), the quota may be lifted upon request.

!!! Note “Note” The HOME filesystem is intended for preparation, evaluation, processing and storage of data generated by active Projects.

The HOME filesystem should not be used to archive data of past Projects or other unrelated data.

The files on HOME filesystem will not be deleted until end of the users lifecycle.

The filesystem is backed up, such that it can be restored in case of catastrophic failure resulting in significant data loss. This backup however is not intended to restore old versions of user data or to restore (accidentaly) deleted files.

The HOME filesystem is realized as Lustre parallel filesystem and is available on all login and computational nodes. Default stripe size is 1MB, stripe count is 1. There are 22 OSTs dedicated for the HOME filesystem.

!!! Note “Note” Setting stripe size and stripe count correctly for your needs may significantly impact the I/O performance you experience.

HOME filesystem	
Mountpoint	/home
Capacity	320TB
Throughput	2GB/s
User quota	250GB
Default stripe size	1MB
Default stripe count	1
Number of OSTs	22

SCRATCH

The SCRATCH filesystem is mounted in directory /scratch. Users may freely create subdirectories and files on the filesystem. Accessible capacity is 146TB, shared among all users. Individual users are restricted by filesystem usage quotas, set to 100TB per user. The purpose of this quota is to prevent runaway programs from filling the entire filesystem and deny service to other users. If 100TB should prove as insufficient for particular user, please contact support [\[4\]](#), the quota may be lifted upon request.

!!! Note “Note” The Scratch filesystem is intended for temporary scratch data generated during the calculation as well as for high performance access to input and output files. All I/O intensive jobs must use the SCRATCH filesystem as their working directory.

>Users are advised to save the necessary data from the SCRATCH filesystem to HOME filesystem as

Files on the SCRATCH filesystem that are ****not accessed for more than 90 days**** will be automati

The SCRATCH filesystem is realized as Lustre parallel filesystem and is available from all login and computational nodes. Default stripe size is 1MB, stripe count is 1. There are 10 OSTs dedicated for the SCRATCH filesystem.

!!! Note “Note” Setting stripe size and stripe count correctly for your needs may significantly impact the I/O performance you experience.

SCRATCH filesystem	
Mountpoint	/scratch
Capacity	146TB
Throughput	6GB/s
User quota	100TB
Default stripe size	1MB
Default stripe count	1
Number of OSTs	10

Disk usage and quota commands

User quotas on the file systems can be checked and reviewed using following command:

```
$ lfs quota dir
```

Example for Lustre HOME directory:

```
$ lfs quota /home
```

```
Disk quotas for user user001 (uid 1234):
```

Filesystem	kbytes	quota	limit	grace	files	quota	limit	grace
/home	300096	0	250000000	-	2102	0	500000	-

```
Disk quotas for group user001 (gid 1234):
```

Filesystem	kbytes	quota	limit	grace	files	quota	limit	grace
/home	300096	0	0	-	2102	0	0	-

In this example, we view current quota size limit of 250GB and 300MB currently used by user001.

Example for Lustre SCRATCH directory:

```
$ lfs quota /scratch
```

```
Disk quotas for user user001 (uid 1234):
```

Filesystem	kbytes	quota	limit	grace	files	quota	limit	grace
------------	--------	-------	-------	-------	-------	-------	-------	-------

```

      /scratch      8      0 1000000000000      -      3      0      0      -
Disk quotas for group user001 (gid 1234):
Filesystem kbytes quota limit grace files quota limit grace
/scratch   8      0      0      -      3      0      0      -

```

In this example, we view current quota size limit of 100TB and 8KB currently used by user001.

To have a better understanding of where the space is exactly used, you can use following command to find out.

```
$ du -hs dir
```

Example for your HOME directory:

```

$ cd /home
$ du -hs * .[a-zA-z0-9]* | grep -E "[0-9]*G|[0-9]*M" | sort -hr
258M      cuda-samples
15M       .cache
13M       .mozilla
5,5M      .eclipse
2,7M      .idb_13.0_linux_intel64_app

```

This will list all directories which are having MegaBytes or GigaBytes of consumed space in your actual (in this example HOME) directory. List is sorted in descending order from largest to smallest files/directories.

To have a better understanding of previous commands, you can read manpages.

```
$ man lfs
```

```
$ man du
```

Extended ACLs

Extended ACLs provide another security mechanism beside the standard POSIX ACLs which are defined by three entries (for owner/group/others). Extended ACLs have more than the three basic entries. In addition, they also contain a mask entry and may contain any number of named user and named group entries.

ACLs on a Lustre file system work exactly like ACLs on any Linux file system. They are manipulated with the standard tools in the standard manner. Below, we create a directory and allow a specific user access.

```

[vop999@login1.anselm ~]$ umask 027
[vop999@login1.anselm ~]$ mkdir test
[vop999@login1.anselm ~]$ ls -ld test
drwxr-x--- 2 vop999 vop999 4096 Nov  5 14:17 test
[vop999@login1.anselm ~]$ getfacl test
# file: test
# owner: vop999
# group: vop999
user::rwx
group::r-x
other::---

[vop999@login1.anselm ~]$ setfacl -m user:johnsm:rwx test
[vop999@login1.anselm ~]$ ls -ld test

```

```
drwxrwx---+ 2 vop999 vop999 4096 Nov  5 14:17 test
[vop999@login1.anselm ~]$ getfacl test
# file: test
# owner: vop999
# group: vop999
user::rwx
user:johnsm:rwx
group::r-x
mask::rwx
other::---
```

Default ACL mechanism can be used to replace setuid/setgid permissions on directories. Setting a default ACL on a directory (-d flag to setfacl) will cause the ACL permissions to be inherited by any newly created file or subdirectory within the directory. Refer to this page for more information on Linux ACL:

http://www.vanemery.com/Linux/ACL/POSIX_ACL_on_Linux.html 

Local Filesystems

Local Scratch

!!! Note “Note” Every computational node is equipped with 330GB local scratch disk.

Use local scratch in case you need to access large amount of small files during your calculation.

The local scratch disk is mounted as /lscratch and is accessible to user at /lscratch/\$PBS_JOBID directory.

The local scratch filesystem is intended for temporary scratch data generated during the calculation as well as for high performance access to input and output files. All I/O intensive jobs that access large number of small files within the calculation must use the local scratch filesystem as their working directory. This is required for performance reasons, as frequent access to number of small files may overload the metadata servers (MDS) of the Lustre filesystem.

!!! Note “Note” The local scratch directory /lscratch/\$PBS_JOBID will be deleted immediately after the calculation end. Users should take care to save the output data from within the jobscript.

local SCRATCH filesystem	
Mountpoint	/lscratch
Accesspoint	/lscratch/\$PBS_JOBID
Capacity	330GB
Throughput	100MB/s
User quota	none

RAM disk

Every computational node is equipped with filesystem realized in memory, so called RAM disk.

!!! Note “Note” Use RAM disk in case you need really fast access to your data of limited size during your calculation. Be very careful, use of RAM disk filesystem is at the expense of operational memory.

The local RAM disk is mounted as /ramdisk and is accessible to user at /ramdisk/\$PBS_JOBID

directory.

The local RAM disk filesystem is intended for temporary scratch data generated during the calculation as well as for high performance access to input and output files. Size of RAM disk filesystem is limited. Be very careful, use of RAM disk filesystem is at the expense of operational memory. It is not recommended to allocate large amount of memory and use large amount of data in RAM disk filesystem at the same time.

!!! Note “Note” The local RAM disk directory /ramdisk/\$PBS_JOBID will be deleted immediately after the calculation end. Users should take care to save the output data from within the jobscript.

RAM
disk

Mountpoint: /ramdisk

Accesspoint: /ramdisk/\$PBS_JOBID

Capacity: 60GB

at
com-
pute
nodes
with-
out
ac-
cel-
era-
tor,
90GB
at
com-
pute
nodes
with
ac-
cel-
era-
tor,
500GB
at
fat
nodes

RAM	
disk	
Throughput	1.5 GB/s write, over 5 GB/s read, single thread, over 10 GB/s write, over 50 GB/s read, 16 threads
User	none
quota	

tmp

Each node is equipped with local /tmp directory of few GB capacity. The /tmp directory should be used to work with small temporary files. Old files in /tmp directory are automatically purged.

Summary

Mount point	/home
Usage	home directory
Mount point	/scratch
Usage	cluster shared jobs' data

Mount Usage	
/lscratch	node local jobs' data
/ramdisk	node local jobs' data
/tmp	local tem- po- rary files

PRACE User Support

Intro

PRACE users coming to Anselm as to TIER-1 system offered through the DECI calls are in general treated as standard users and so most of the general documentation applies to them as well. This section shows the main differences for quicker orientation, but often uses references to the original documentation. PRACE users who don't undergo the full procedure (including signing the IT4I AuP on top of the PRACE AuP) will not have a password and thus access to some services intended for regular users. This can lower their comfort, but otherwise they should be able to use the TIER-1 system as intended. Please see the Obtaining Login Credentials section, if the same level of access is required.

All general PRACE User Documentation [\[4\]](#) should be read before continuing reading the local documentation here.

Help and Support

If you have any troubles, need information, request support or want to install additional software, please use PRACE Helpdesk [\[4\]](#).

Information about the local services are provided in the introduction of general user documentation. Please keep in mind, that standard PRACE accounts don't have a password to access the web interface of the local (IT4Innovations) request tracker and thus a new ticket should be created by sending an e-mail to support[at]it4i.cz.

Obtaining Login Credentials

In general PRACE users already have a PRACE account setup through their HOMESITE (institution from their country) as a result of rewarded PRACE project proposal. This includes signed PRACE AuP, generated and registered certificates, etc.

If there's a special need a PRACE user can get a standard (local) account at IT4Innovations. To get an account on the Anselm cluster, the user needs to obtain the login credentials. The procedure is the same as for general users of the cluster, so please see the corresponding section of the general documentation here.

Accessing the cluster




Access with GSI-SSH

For all PRACE users the method for interactive access (login) and data transfer based on grid services from Globus Toolkit (GSI SSH and GridFTP) is supported.

The user will need a valid certificate and to be present in the PRACE LDAP (please contact your HOME SITE or the primary investigator of your project for LDAP account creation).

Most of the information needed by PRACE users accessing the Anselm TIER-1 system can be found here:

- General user's FAQ [\[4\]](#)
- Certificates FAQ [\[4\]](#)

- Interactive access using GSISSH 
- Data transfer with GridFTP 
- Data transfer with gtransfer 

Before you start to use any of the services don't forget to create a proxy certificate from your certificate:

```
$ grid-proxy-init
```

To check whether your proxy certificate is still valid (by default it's valid 12 hours), use:

```
$ grid-proxy-info
```

To access Anselm cluster, two login nodes running GSI SSH service are available. The service is available from public Internet as well as from the internal PRACE network (accessible only from other PRACE partners).

Access from PRACE network:

It is recommended to use the single DNS name `anselm-prace.it4i.cz` which is distributed between the two login nodes. If needed, user can login directly to one of the login nodes. The addresses are:

Login address	Port
<code>anselm-prace.it4i.cz</code>	2222
<code>login1-prace.anselm.it4i.cz</code>	2222
<code>login2-prace.anselm.it4i.cz</code>	2222

```
$ gsissh -p 2222 anselm-prace.it4i.cz
```

When logging from other PRACE system, the `prace_service` script can be used:

```
$ gsissh `prace_service` -i -s anselm`
```

Access from public Internet:

It is recommended to use the single DNS name `anselm.it4i.cz` which is distributed between the two login nodes. If needed, user can login directly to one of the login nodes. The addresses are:

Login address	Port
<code>anselm.it4i.cz</code>	2222
<code>login1.anselm.it4i.cz</code>	2222
<code>login2.anselm.it4i.cz</code>	2222

```
$ gsissh -p 2222 anselm.it4i.cz
```

When logging from other PRACE system, the `prace_service` script can be used:

```
$ gsissh `prace_service` -e -s anselm`
```

Although the preferred and recommended file transfer mechanism is using GridFTP, the GSI SSH implementation on Anselm supports also SCP, so for small files transfer `gsiscp` can be used:

```
$ gsiscp -P 2222 _LOCAL_PATH_TO_YOUR_FILE_ anselm.it4i.cz:_ANSELM_PATH_TO_YOUR_FILE_
```

```
$ gsiscp -P 2222 anselm.it4i.cz:_ANSELM_PATH_TO_YOUR_FILE_ _LOCAL_PATH_TO_YOUR_FILE_
```

```
$ gsiscp -P 2222 _LOCAL_PATH_TO_YOUR_FILE_ anselm-prace.it4i.cz:_ANSELM_PATH_TO_YOUR_FILE_
```

```
$ gsiscp -P 2222 anselm-prace.it4i.cz:_ANSELM_PATH_TO_YOUR_FILE_ _LOCAL_PATH_TO_YOUR_FILE_
```

Access to X11 applications (VNC)

If the user needs to run X11 based graphical application and does not have a X11 server, the applications can be run using VNC service. If the user is using regular SSH based access, please see the section in general documentation.

If the user uses GSI SSH based access, then the procedure is similar to the SSH based access, only the port forwarding must be done using GSI SSH:

```
$ gsissh -p 2222 anselm.it4i.cz -L 5961:localhost:5961
```

Access with SSH

After successful obtainment of login credentials for the local IT4Innovations account, the PRACE users can access the cluster as regular users using SSH. For more information please see the section in general documentation.

File transfers

PRACE users can use the same transfer mechanisms as regular users (if they've undergone the full registration procedure). For information about this, please see the section in the general documentation.

Apart from the standard mechanisms, for PRACE users to transfer data to/from Anselm cluster, a GridFTP server running Globus Toolkit GridFTP service is available. The service is available from public Internet as well as from the internal PRACE network (accessible only from other PRACE partners).

There's one control server and three backend servers for striping and/or backup in case one of them would fail.

Access from PRACE network:

Login address	Port
gridftp-prace.anselm.it4i.cz	2812
login1-prace.anselm.it4i.cz	2813
login2-prace.anselm.it4i.cz	2813
dm1-prace.anselm.it4i.cz	2813

Copy files **to** Anselm by running the following commands on your local machine:

```
$ globus-url-copy file://_LOCAL_PATH_TO_YOUR_FILE_ gsiftp://gridftp-prace.anselm.it4i.cz
```

Or by using `prace_service` script:

```
$ globus-url-copy file://_LOCAL_PATH_TO_YOUR_FILE_ gsiftp://`prace_service` -i -f anse
```

Copy files **from** Anselm:

```
$ globus-url-copy gsiftp://gridftp-prace.anselm.it4i.cz:2812/home/prace/_YOUR_ACCOUNT
```

Or by using `prace_service` script:

```
$ globus-url-copy gsiftp://`prace_service` -i -f anselm`/home/prace/_YOUR_ACCOUNT_ON_A
```

Access from public Internet:

Login address	Port
gridftp.anselm.it4i.cz	2812
login1.anselm.it4i.cz	2813
login2.anselm.it4i.cz	2813
dm1.anselm.it4i.cz	2813

Copy files **to** Anselm by running the following commands on your local machine:

```
$ globus-url-copy file://_LOCAL_PATH_TO_YOUR_FILE_ gsiftp://gridftp.anselm.it4i.cz:2812
```

Or by using prace_service script:

```
$ globus-url-copy file://_LOCAL_PATH_TO_YOUR_FILE_ gsiftp://`prace_service -e -f anse
```

Copy files **from** Anselm:

```
$ globus-url-copy gsiftp://gridftp.anselm.it4i.cz:2812/home/prace/_YOUR_ACCOUNT_ON_AN
```

Or by using prace_service script:

```
$ globus-url-copy gsiftp://`prace_service -e -f anselm`/home/prace/_YOUR_ACCOUNT_ON_A
```

Generally both shared file systems are available through GridFTP:

File	
sys-	
tem	
mount	
point	Filesystem
/home	Lustre
/scratch	Lustre

More information about the shared file systems is available [here](#).

Usage of the cluster

There are some limitations for PRACE user when using the cluster. By default PRACE users aren't allowed to access special queues in the PBS Pro to have high priority or exclusive access to some special equipment like accelerated nodes and high memory (fat) nodes. There may be also restrictions obtaining a working license for the commercial software installed on the cluster, mostly because of the license agreement or because of insufficient amount of licenses.

For production runs always use scratch file systems, either the global shared or the local ones. The available file systems are described [here](#).

Software, Modules and PRACE Common Production Environment

All system wide installed software on the cluster is made available to the users via the modules. The information about the environment and modules usage is in this section of general documentation.

PRACE users can use the “prace” module to use the PRACE Common Production Environment [\[4\]](#).

```
$ module load prace
```

Resource Allocation and Job Execution

General information about the resource allocation, job queuing and job execution is in this section of general documentation.

For PRACE users, the default production run queue is “qprace”. PRACE users can also use two other queues “qexp” and “qfree”.

Project		
Active re-queue project sources		
qexp	no	none
Ex-press queue		re-quired
qprace	yes	> 0
Pro-duc-tion queue		
qfree	yes	none
Free re-source queue		re-quired

qprace, **the PRACE ***: This queue is intended for normal production runs. It is required that active project with nonzero remaining resources is specified to enter the qprace. The queue runs with medium priority and no special authorization is required to use it. The maximum runtime in qprace is 12 hours. If the job needs longer time, it must use checkpoint/restart functionality.

Accounting & Quota

The resources that are currently subject to accounting are the core hours. The core hours are accounted on the wall clock basis. The accounting runs whenever the computational cores are allocated or blocked via the PBS Pro workload manager (the qsub command), regardless of whether the cores are actually used for any calculation. See example in the general documentation [\[4\]](#).

PRACE users should check their project accounting using the PRACE Accounting Tool (DART) [\[4\]](#).

Users who have undergone the full local registration procedure (including signing the IT4Innovations Acceptable Use Policy) and who have received local password may check at any time, how many core-hours have been consumed by themselves and their projects using the command “it4ifree”.

Please note that you need to know your user password to use the command and that the displayed core hours are “system core hours” which differ from PRACE “standardized core hours”.

!!! Note “Note” The **it4ifree** command is a part of it4i.portal.clients package, located here: <https://pypi.python.org/pypi/it4i.portal.clients>

```
$ it4ifree
Password:
  PID      Total    Used    ...by me Free
-----
OPEN-0-0 1500000 400644    225265 1099356
DD-13-1   10000    2606      2606    7394
```


By default file system quota is applied. To check the current status of the quota use

```
$ lfs quota -u USER_LOGIN /home
$ lfs quota -u USER_LOGIN /scratch
```

If the quota is insufficient, please contact the support and request an increase.

Hardware Overview

The Anselm cluster consists of 209 computational nodes named cn[1-209] of which 180 are regular compute nodes, 23 GPU Kepler K20 accelerated nodes, 4 MIC Xeon Phi 5110 accelerated nodes and 2 fat nodes. Each node is a powerful x86-64 computer, equipped with 16 cores (two eight-core Intel Sandy Bridge processors), at least 64GB RAM, and local hard drive. The user access to the Anselm cluster is provided by two login nodes login[1,2]. The nodes are interlinked by high speed InfiniBand and Ethernet networks. All nodes share 320TB /home disk storage to store the user files. The 146TB shared /scratch storage is available for the scratch data.

The Fat nodes are equipped with large amount (512GB) of memory. Virtualization infrastructure provides resources to run long term servers and services in virtual mode. Fat nodes and virtual servers may access 45 TB of dedicated block storage. Accelerated nodes, fat nodes, and virtualization infrastructure are available upon request  made by a PI.

Schematic representation of the Anselm cluster. Each box represents a node (computer) or storage capacity:

User-oriented infrastructure

Storage

Management infrastructure

login1

login2

dm1

Rack 01, Switch isw5

cn186

cn187

cn188

cn189

cn181

cn182

cn183

cn184

cn185

Rack 01, Switch isw4

cn29

cn30

cn31

cn32

cn33

cn34

cn35

cn36

cn19

cn20

cn21

cn22

cn23

cn24

cn25

cn26

cn27

cn28

Lustre FS

/home320TB

Lustre FS

/scratch146TB

Managementnodes

Block storage45 TB

Virtualizationinfrastructureservers

...

Srv node

Srv node

Srv node

...

Rack 01, Switch isw0

cn11

cn12

cn13

cn14

cn15

cn16

cn17

cn18

cn1

cn2

cn3

cn4

cn5

cn6

cn7

cn8

cn9

cn10

Rack 02, Switch isw10

cn73

cn74

cn75

cn76

cn77

cn78

cn79

cn80

cn190

cn191

cn192

cn205

cn206

Rack 02, Switch isw9

cn65

cn66

cn67

cn68

cn69

cn70

cn71

cn72

cn55

cn56

cn57

cn58

cn59

cn60

cn61

cn62

cn63

cn64

Rack 02, Switch isw6

cn47

cn48

cn49

cn50

cn51

cn52

cn53

cn54

cn37

cn38

cn39

cn40

cn41

cn42

cn43

cn44

cn45

cn46

Rack 03, Switch isw15

cn193

cn194

cn195

cn207

cn117

cn118

cn119

cn120

cn121

cn122

cn123

cn124

cn125

cn126

Rack 03, Switch isw14

cn109

cn110

cn111

cn112

cn113

cn114

cn115

cn116

cn99

cn100

cn101

cn102

cn103

cn104

cn105

cn106

cn107

cn108

Rack 03, Switch isw11

cn91

cn92

cn93

cn94

cn95

cn96

cn97

cn98

cn81

cn82

cn83

cn84

cn85

cn86

cn87

cn88

cn89

cn90

Rack 04, Switch isw20

cn173

cn174

cn175

cn176

cn177

cn178

cn179

cn180

cn163

cn164

cn165

cn166

cn167

cn168

cn169

cn170

cn171

cn172

Rack 04, Switch isw19

cn155

cn156

cn157

cn158

cn159

cn160

cn161

cn162

cn145

cn146

cn147

cn148

cn149

cn150

cn151

cn152

cn153

cn154

Rack 04, Switch isw16

cn137

cn138

cn139

cn140

cn141

cn142

cn143

cn144

cn127

cn128

cn129

cn130

cn131

cn132

cn133

cn134

cn135

cn136

Rack 05, Switch isw21

cn201

cn202

cn203

cn204

cn196

cn197

cn198

cn199

cn200

Fat node cn208

Fat node cn209

...

The cluster compute nodes cn[1-207] are organized within 13 chassis.

There are four types of compute nodes:

- 180 compute nodes without the accelerator
- 23 compute nodes with GPU accelerator - equipped with NVIDIA Tesla Kepler K20
- 4 compute nodes with MIC accelerator - equipped with Intel Xeon Phi 5110P
- 2 fat nodes - equipped with 512GB RAM and two 100GB SSD drives

More about Compute nodes.

GPU and accelerated nodes are available upon request, see the Resources Allocation Policy.

All these nodes are interconnected by fast InfiniBand network and Ethernet network. More about the Network. Every chassis provides Infiniband switch, marked **isw**, connecting all nodes in the chassis, as well as connecting the chassis to the upper level switches.

All nodes share 360TB /home disk storage to store user files. The 146TB shared /scratch storage is available for the scratch data. These file systems are provided by Lustre parallel file system. There is also local disk storage available on all compute nodes /lscratch. More about Storage.

The user access to the Anselm cluster is provided by two login nodes login1, login2, and data mover node dm1. More about accessing cluster.

The parameters are summarized in the following tables:

In general

Primary purpose	High Performance Computing
Architecture of compute nodes	x86-64
Operating system	Linux
Compute nodes	
Totally	209
Processor cores	16 (2x8 cores)
RAM	min. 64 GB, min. 4 GB per core
Local disk drive	yes - usually 500 GB
Compute network	InfiniBand QDR, fully non-blocking, fat-tree
w/o accelerator	180, cn[1-180]
GPU accelerated	23, cn[181-203]
MIC accelerated	4, cn[204-207]
Fat compute nodes	2, cn[208-209]

In general			
In total			
Total theoretical peak performance	(Rpeak)	94 Tflop/s	
Total max. LINPACK performance	(Rmax)	73 Tflop/s	
Total amount of RAM		15.136 TB	

Node	Processors	Memory	Accelerator
w/o accelerator	2x Intel Sandy Bridge E5-2665, 2.4GHz	64GB	-
GPU accelerated	2x Intel Sandy Bridge E5-2470, 2.3GHz	96GB	NVIDIA Kepler K20
MIC accelerated	2x Intel Sandy Bridge E5-2470, 2.3GHz	96GB	Intel Xeon Phi P5110
Fat compute node	2x Intel Sandy Bridge E5-2665, 2.4GHz	512GB	-

For more details please refer to the Compute nodes, Storage, and Network.

Compute Nodes

Nodes Configuration

Salomon is cluster of x86-64 Intel based nodes. The cluster contains two types of compute nodes of the same processor type and memory size. Compute nodes with MIC accelerator **contains two Intel Xeon Phi 7120P accelerators**.

More about schematic representation of the Salomon cluster compute nodes IB topology.

Compute Nodes Without Accelerator

- codename “grifton”
- 576 nodes
- 13 824 cores in total
- two Intel Xeon E5-2680v3, 12-core, 2.5GHz processors per node
- 128 GB of physical memory per node

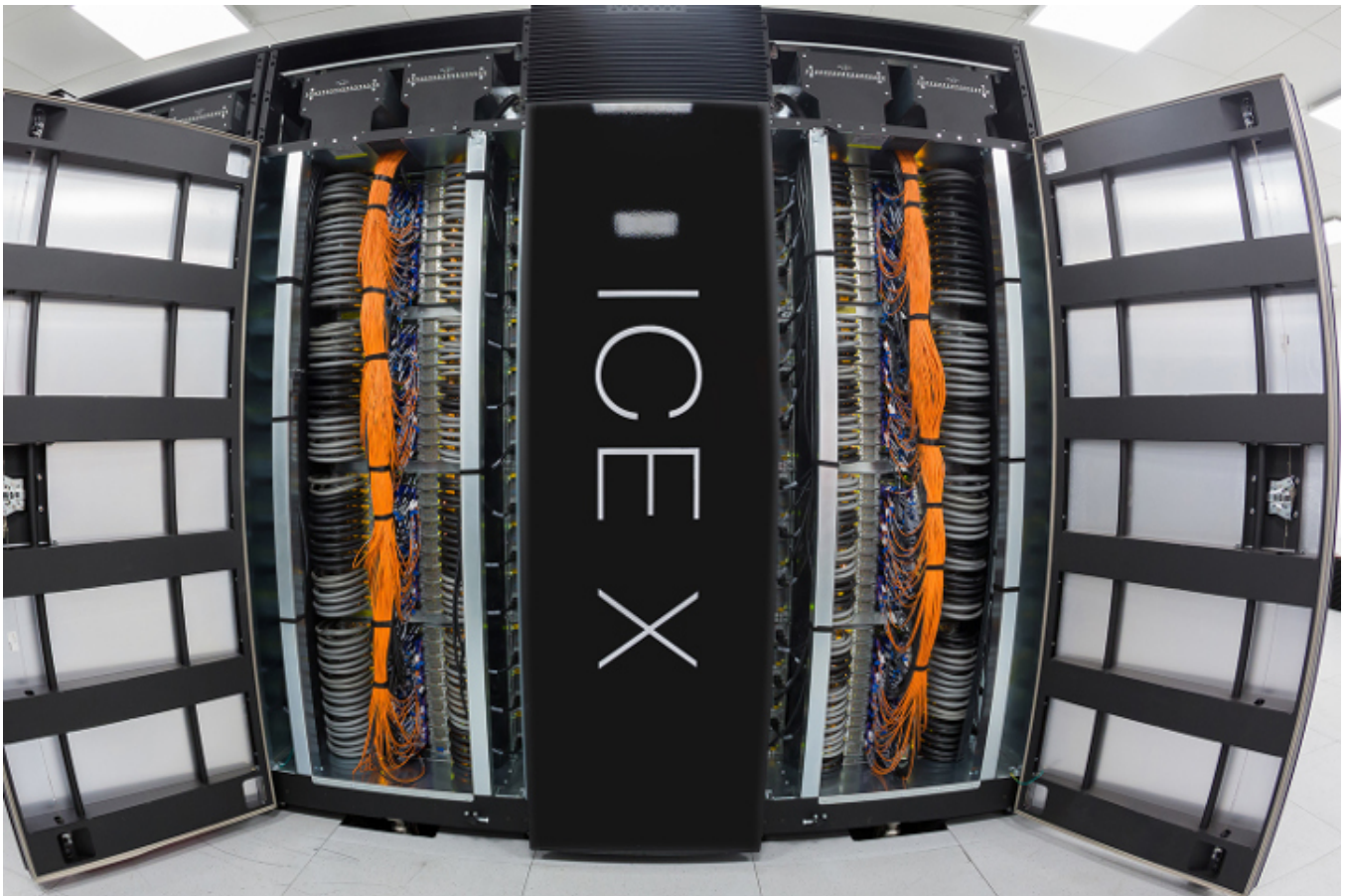


Figure 1: cn_m_cell

Compute Nodes With MIC Accelerator

- codename “perrin”
- 432 nodes
- 10 368 cores in total
- two Intel Xeon E5-2680v3, 12-core, 2.5GHz processors per node
- 128 GB of physical memory per node

- MIC accelerator 2x Intel Xeon Phi 7120P per node, 61-cores, 16GB per accelerator

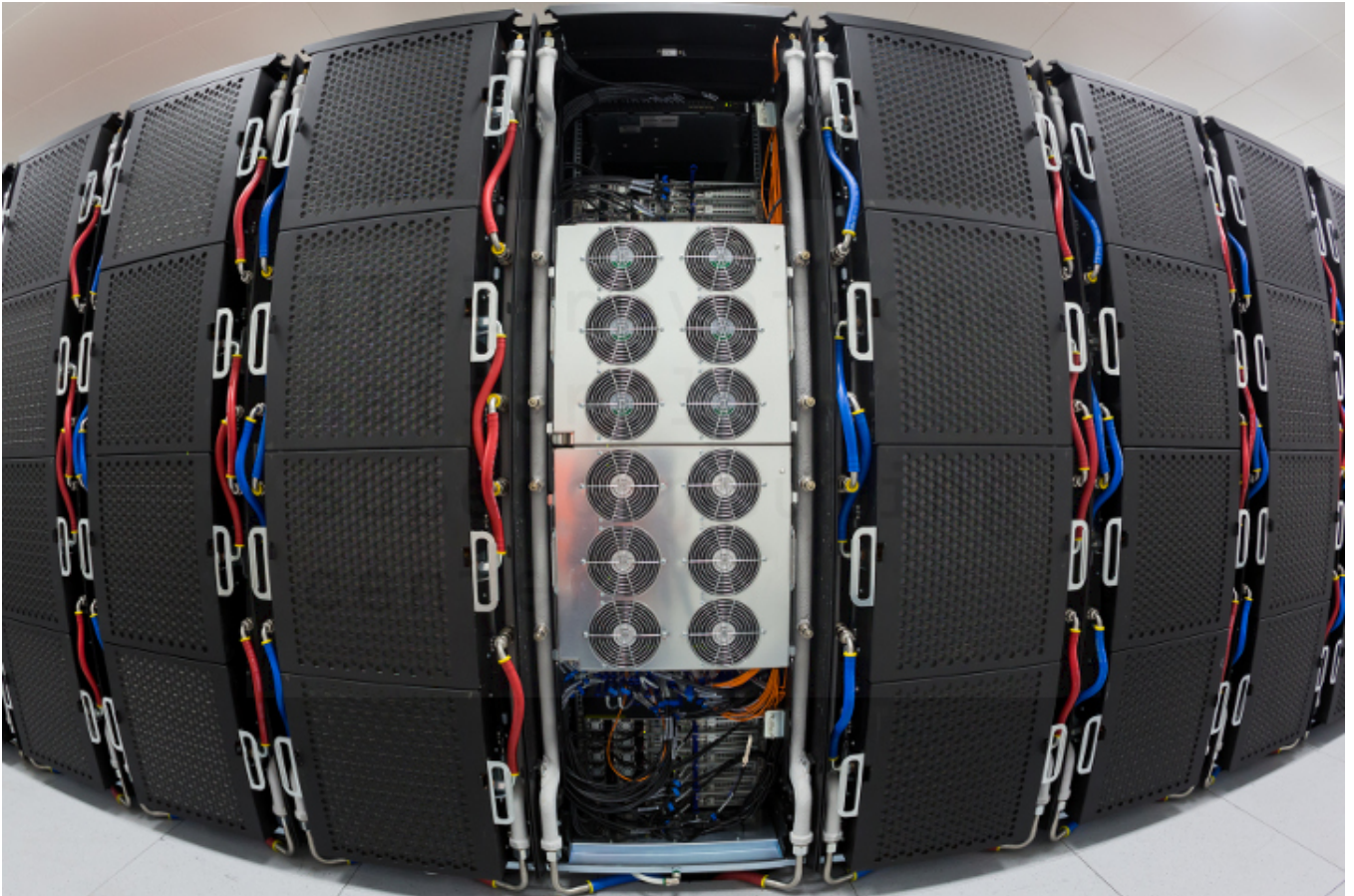


Figure 2: cn_mic



Figure 3: (source Silicon Graphics International Corp.)

UV 2000

- codename “UV2000”
- 1 node
- 112 cores in total
- 14x Intel Xeon E5-4627v2, 8-core, 3.3GHz processors, in 14 NUMA nodes
- 3328 GB of physical memory per node
- 1x NVIDIA GM200 (GeForce GTX TITAN X), 12GB RAM



Figure 4: cn_mic

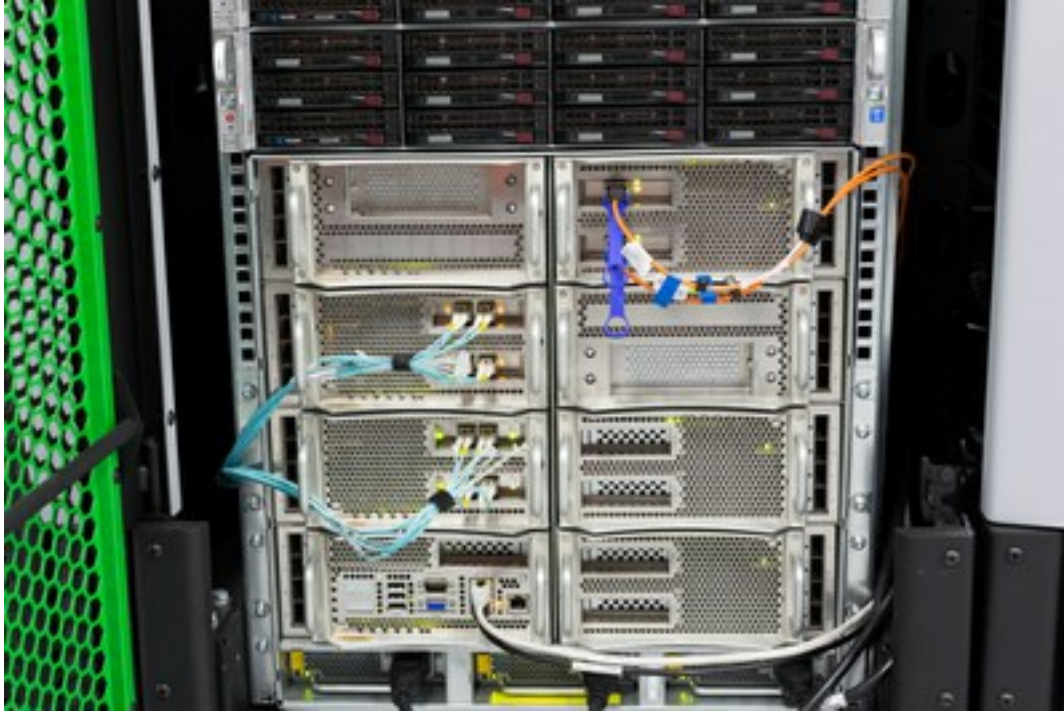


Figure 5:

Compute Nodes Summary

Node type	Count	Memory	Cores	—	—	Nodes without accelerator
	576	128GB	24 @ 2.5Ghz			
Nodes with MIC accelerator	432	128GB	32GB			
			24 @ 2.5Ghz			
	61	@ 1.238GHz	UV2000 SMP node	1	3328GB	
	112	@ 3.3GHz				

Processor Architecture

Salomon is equipped with Intel Xeon processors Intel Xeon E5-2680v3. Processors support Advanced Vector Extensions 2.0 (AVX2) 256-bit instruction set.

Intel Xeon E5-2680v3 Processor

- 12-core
- speed: 2.5 GHz, up to 3.3 GHz using Turbo Boost Technology
- peak performance: 19.2 Gflop/s per core
- caches:
 - Intel® Smart Cache: 30 MB
- memory bandwidth at the level of the processor: 68 GB/s

MIC Accelerator Intel Xeon Phi 7120P Processor

- 61-core
- speed: 1.238 GHz, up to 1.333 GHz using Turbo Boost Technology

- peak performance: 18.4 Gflop/s per core
- caches:
 - L2: 30.5 MB
- memory bandwidth at the level of the processor: 352 GB/s

Memory Architecture

Memory is equally distributed across all CPUs and cores for optimal performance. Memory is composed of memory modules of the same size and evenly distributed across all memory controllers and memory channels.

Compute Node Without Accelerator

- 2 sockets
- Memory Controllers are integrated into processors.
 - 8 DDR4 DIMMS per node
 - 4 DDR4 DIMMS per CPU
 - 1 DDR4 DIMMS per channel
- Populated memory: 8x 16GB DDR4 DIMM >2133MHz

Compute Node With MIC Accelerator

2 sockets Memory Controllers are integrated into processors.

- 8 DDR4 DIMMS per node
- 4 DDR4 DIMMS per CPU
- 1 DDR4 DIMMS per channel

Populated memory: 8x 16GB DDR4 DIMM 2133MHz MIC Accelerator Intel Xeon Phi 7120P Processor

- 2 sockets
- Memory Controllers are are connected via an Interprocessor Network (IPN) ring.
 - 16 GDDR5 DIMMS per node
 - 8 GDDR5 DIMMS per CPU
 - 2 GDDR5 DIMMS per channel

Shell access and data transfer

Interactive Login

The Salomon cluster is accessed by SSH protocol via login nodes login1, login2, login3 and login4 at address salomon.it4i.cz. The login nodes may be addressed specifically, by prepending the login node name to the address.

!!! Note “Note” The alias >salomon.it4i.cz is currently not available through VPN connection. Please use loginX.salomon.it4i.cz when connected to VPN.

Login address	Port	Protocol	Login node
salomon.it4i.cz	22	ssh	round-robin DNS record for login[1-4]
login1.salomon.it4i.cz	22	ssh	login1
login1.salomon.it4i.cz	22	ssh	login1
login1.salomon.it4i.cz	22	ssh	login1
login1.salomon.it4i.cz	22	ssh	login1

The authentication is by the private key

!!! Note “Note” Please verify SSH fingerprints during the first logon. They are identical on all login nodes:

f6:28:98:e4:f9:b2:a6:8f:f2:f4:2d:0a:09:67:69:80 (DSA)

70:01:c9:9a:5d:88:91:c7:1b:c0:84:d1:fa:4e:83:5c (RSA)

Private key authentication:

On **Linux** or **Mac**, use

```
local $ ssh -i /path/to/id_rsa username@salomon.it4i.cz
```

If you see warning message “UNPROTECTED PRIVATE KEY FILE!”, use this command to set lower permissions to private key file.

```
local $ chmod 600 /path/to/id rsa
```

On **Windows**, use PuTTY ssh client.

After logging in, you will see the command prompt:

<http://www.it4i.cz/?lang=en>

Last login: Tue Jul 9 15:57:38 2013 from your-host.example.com

```
[username@login2.salomon ~]$
```

!!! Note “Note” The environment is **not** shared between login nodes, except for shared filesystems.

Data Transfer

Data in and out of the system may be transferred by the scp  and sftp protocols.

In case large volumes of data are transferred, use dedicated data mover nodes cedge[1-3].salomon.it4i.cz for increased performance.

HTML commented section #1 (removed cedge servers from the table)

Address	Port
salomon.it4i.cz	22
login1.salomon.it4i.cz	22
login2.salomon.it4i.cz	22
login3.salomon.it4i.cz	22
login4.salomon.it4i.cz	22

The authentication is by the private key

HTML commented section #2 (ssh transfer performance data need to be verified)


On linux or Mac, use scp or sftp client to transfer the data to Salomon:

```
local $ scp -i /path/to/id_rsa my-local-file username@salomon.it4i.cz:directory/file
```

```
local $ scp -i /path/to/id_rsa -r my-local-dir username@salomon.it4i.cz:directory
```

or

```
local $ sftp -o IdentityFile=/path/to/id_rsa username@salomon.it4i.cz
```

Very convenient way to transfer files in and out of the Salomon computer is via the fuse filesystem sshfs 

```
local $ sshfs -o IdentityFile=/path/to/id_rsa username@salomon.it4i.cz:. mountpoint
```



Using sshfs, the users Salomon home directory will be mounted on your local computer, just like an external disk.

Learn more on ssh, scp and sshfs by reading the manpages

```
$ man ssh
```

```
$ man scp
```

```
$ man sshfs
```

On Windows, use WinSCP client  to transfer the data. The win-sshfs client  provides a way to mount the Salomon filesystems directly as an external disc.

More information about the shared file systems is available here.

Outgoing connections

Connection restrictions

Outgoing connections, from Salomon Cluster login nodes to the outside world, are restricted to following ports:

Port	Protocol
22	ssh
80	http
443	https
9418	git

!!! Note “Note” Please use **ssh port forwarding** and proxy servers to connect from Salomon to all other remote ports.

Outgoing connections, from Salomon Cluster compute nodes are restricted to the internal network. Direct connections from compute nodes to outside world are cut.

Port forwarding

Port forwarding from login nodes

!!! Note “Note” Port forwarding allows an application running on Salomon to connect to arbitrary remote host and port.

It works by tunneling the connection from Salomon back to users workstation and forwarding from the workstation to the remote host.

Pick some unused port on Salomon login node (for example 6000) and establish the port forwarding:

```
local $ ssh -R 6000:remote.host.com:1234 salomon.it4i.cz
```

In this example, we establish port forwarding between port 6000 on Salomon and port 1234 on the remote.host.com. By accessing localhost:6000 on Salomon, an application will see response of remote.host.com:1234. The traffic will run via users local workstation.

Port forwarding may be done **using PuTTY** as well. On the PuTTY Configuration screen, load your Salomon configuration first. Then go to Connection->SSH->Tunnels to set up the port forwarding. Click Remote radio button. Insert 6000 to Source port textbox. Insert remote.host.com:1234. Click Add button, then Open.

Port forwarding may be established directly to the remote host. However, this requires that user has ssh access to remote.host.com

```
$ ssh -L 6000:localhost:1234 remote.host.com
```

Note: Port number 6000 is chosen as an example only. Pick any free port.

Port forwarding from compute nodes

Remote port forwarding from compute nodes allows applications running on the compute nodes to access hosts outside Salomon Cluster.

First, establish the remote port forwarding from the login node, as described above.

Second, invoke port forwarding from the compute node to the login node. Insert following line into your jobscript or interactive shell

```
$ ssh -TN -f -L 6000:localhost:6000 login1
```

In this example, we assume that port forwarding from login1:6000 to remote.host.com:1234 has been established beforehand. By accessing localhost:6000, an application running on a compute node will see response of remote.host.com:1234

Using proxy servers

Port forwarding is static, each single port is mapped to a particular port on remote host. Connection to other remote host, requires new forward.

!!! Note “Note” Applications with inbuilt proxy support, experience unlimited access to remote hosts, via single proxy server.

To establish local proxy server on your workstation, install and run SOCKS proxy server software. On Linux, sshd demon provides the functionality. To establish SOCKS proxy server listening on port 1080 run:

```
local $ ssh -D 1080 localhost
```

On Windows, install and run the free, open source Sock Puppet  server.

Once the proxy server is running, establish ssh port forwarding from Salomon to the proxy server, port 1080, exactly as described above.

```
local $ ssh -R 6000:localhost:1080 salomon.it4i.cz
```

Now, configure the applications proxy settings to **localhost:6000**. Use port forwarding to access the proxy server from compute nodes as well .

VPN Access

Accessing IT4Innovations internal resources via VPN

For using resources and licenses which are located at IT4Innovations local network, it is necessary to VPN connect to this network. We use Cisco AnyConnect Secure Mobility Client, which is supported on the following operating systems:

- Windows XP
- Windows Vista
- Windows 7
- Windows 8
- Linux
- MacOS

It is impossible to connect to VPN from other operating systems.

VPN client installation

You can install VPN client from web interface after successful login with LDAP credentials on address <https://vpn.it4i.cz/user>

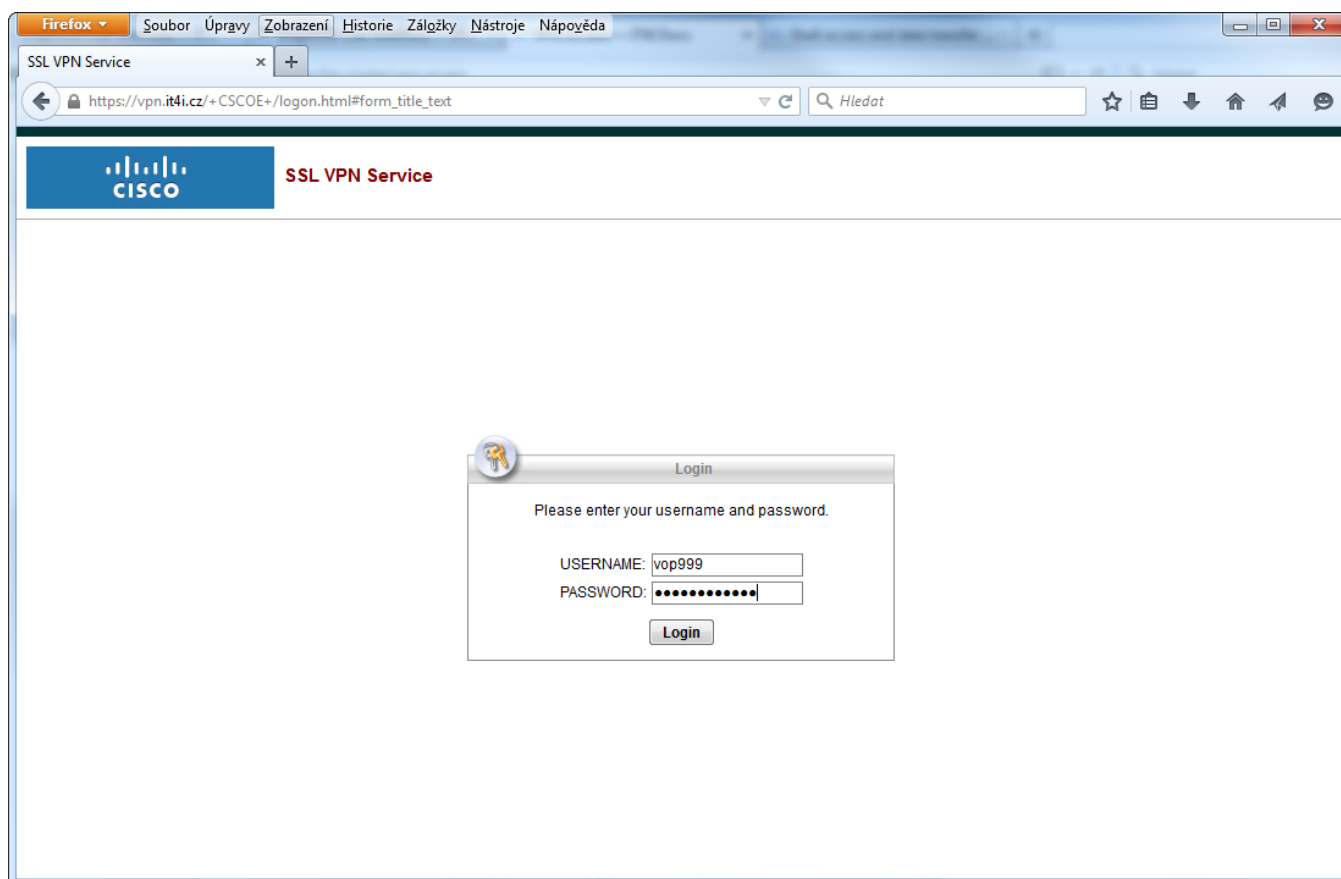
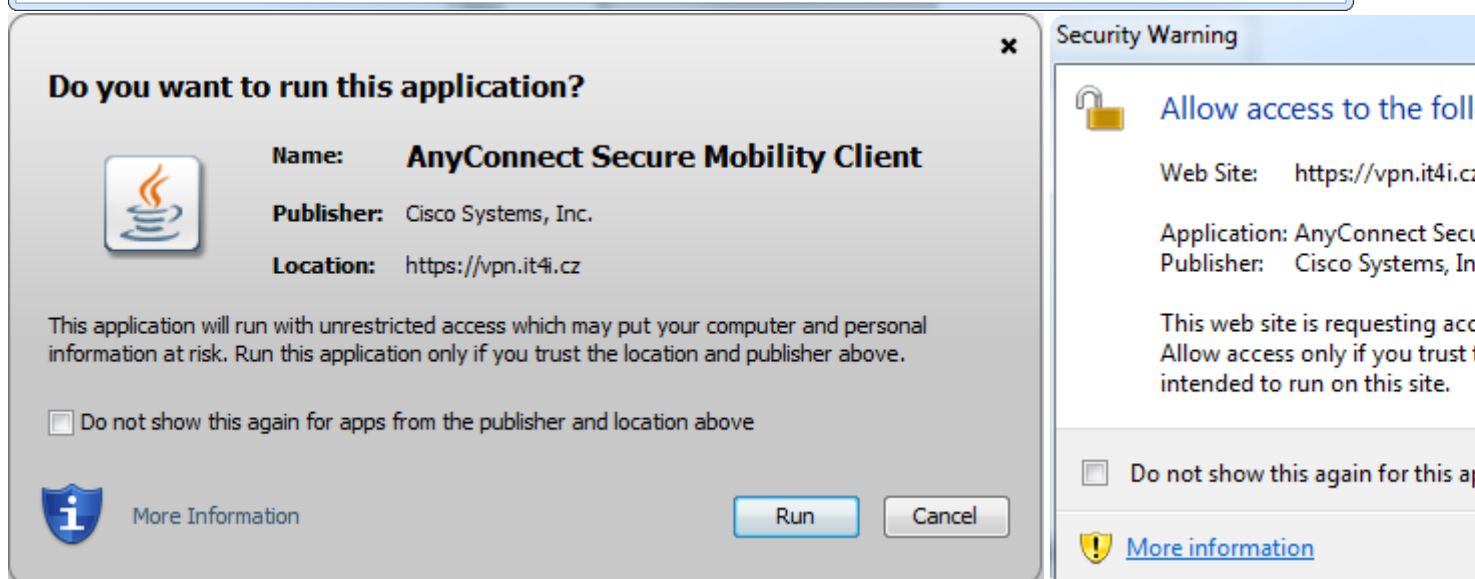
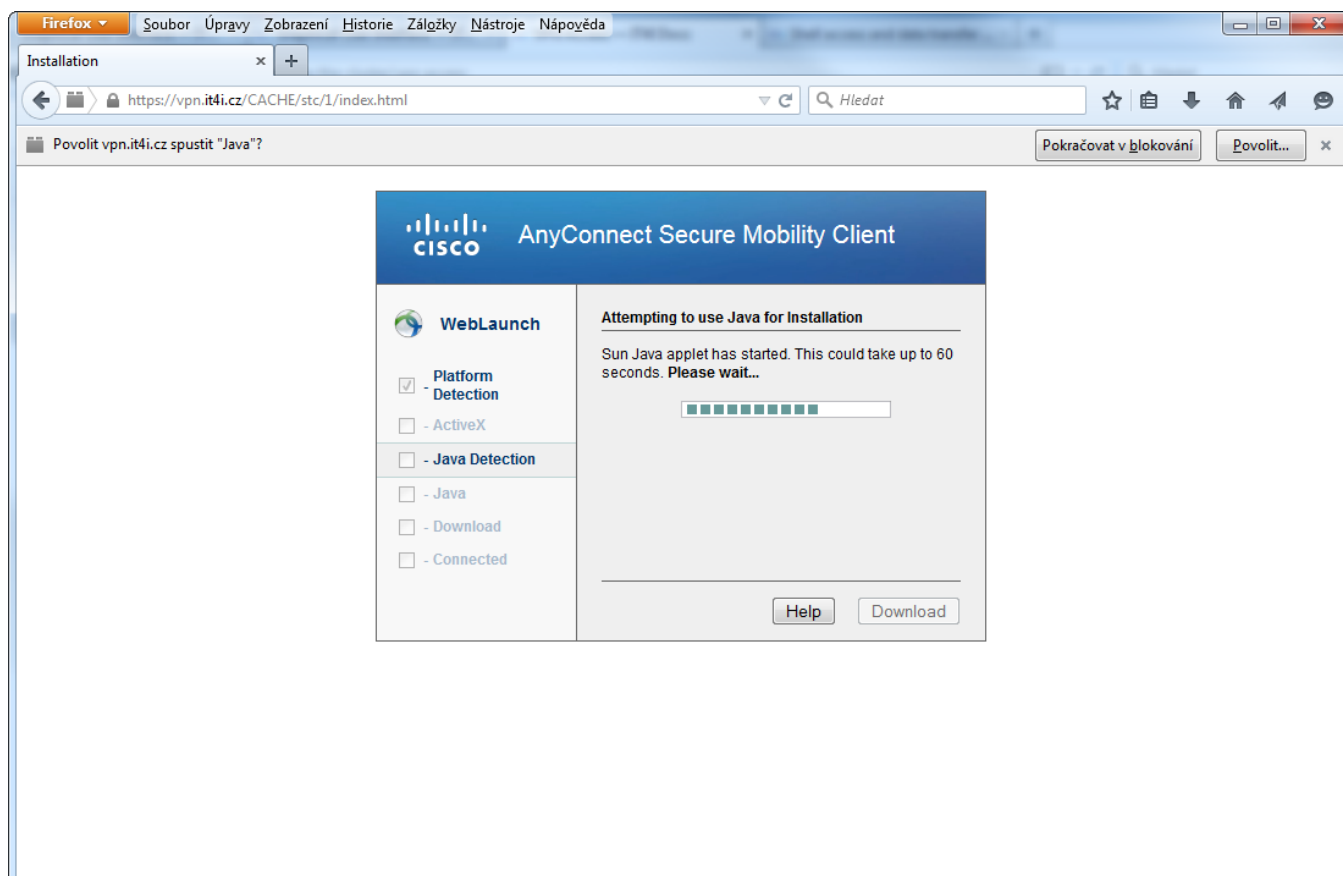


Figure 1:

According to the Java settings after login, the client either automatically installs, or downloads installation file for your operating system. It is necessary to allow start of installation tool for automatic installation.



After successful installation, VPN connection will be established and you can use available resources from IT4I network.

If your Java setting doesn't allow automatic installation, you can download installation file and install VPN client manually.

After you click on the link, download of installation file will start.

After successful download of installation file, you have to execute this tool with administrator's rights and install VPN client manually.

Working with VPN client

You can use graphical user interface or command line interface to run VPN client on all supported operating systems. We suggest using GUI.

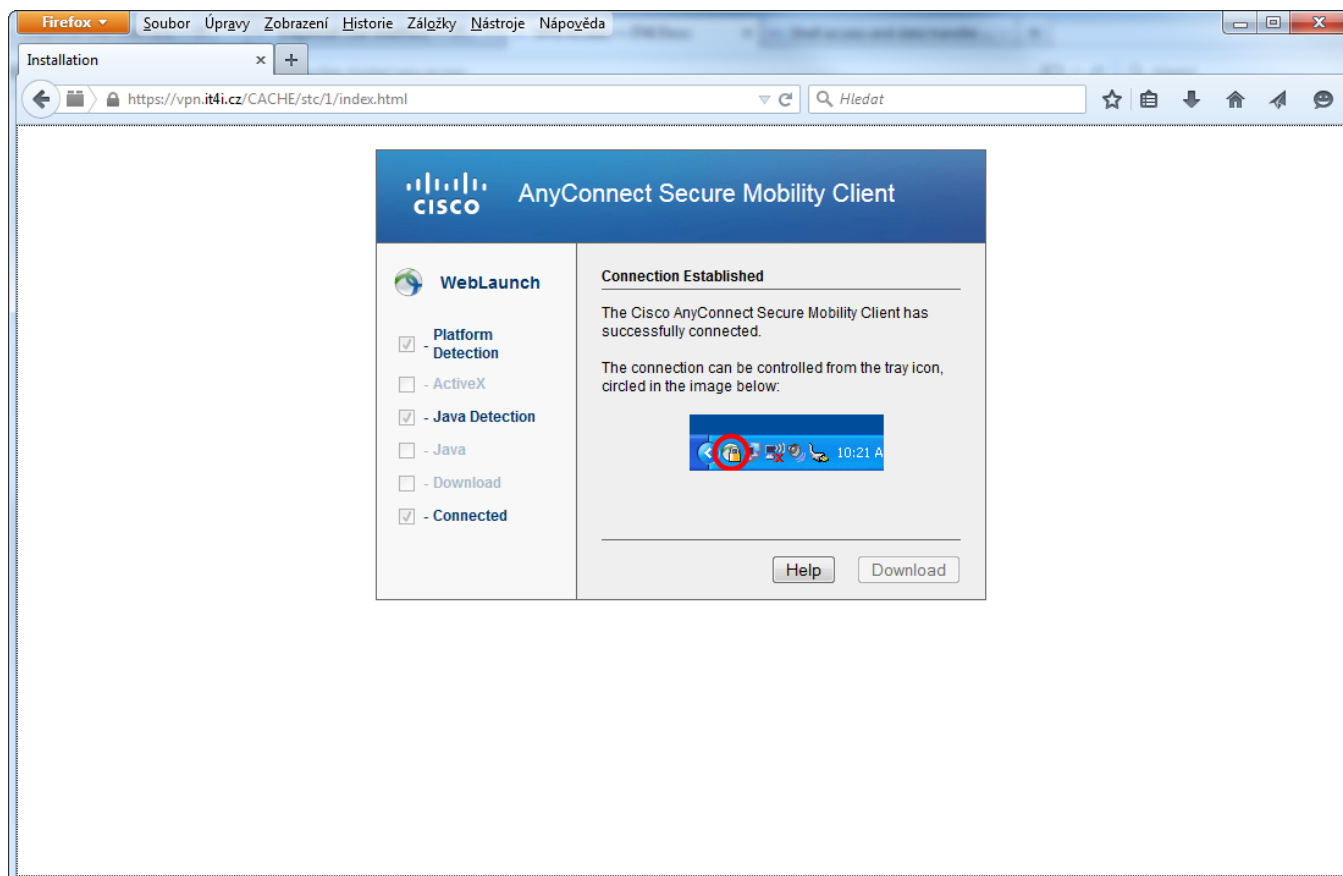


Figure 2:

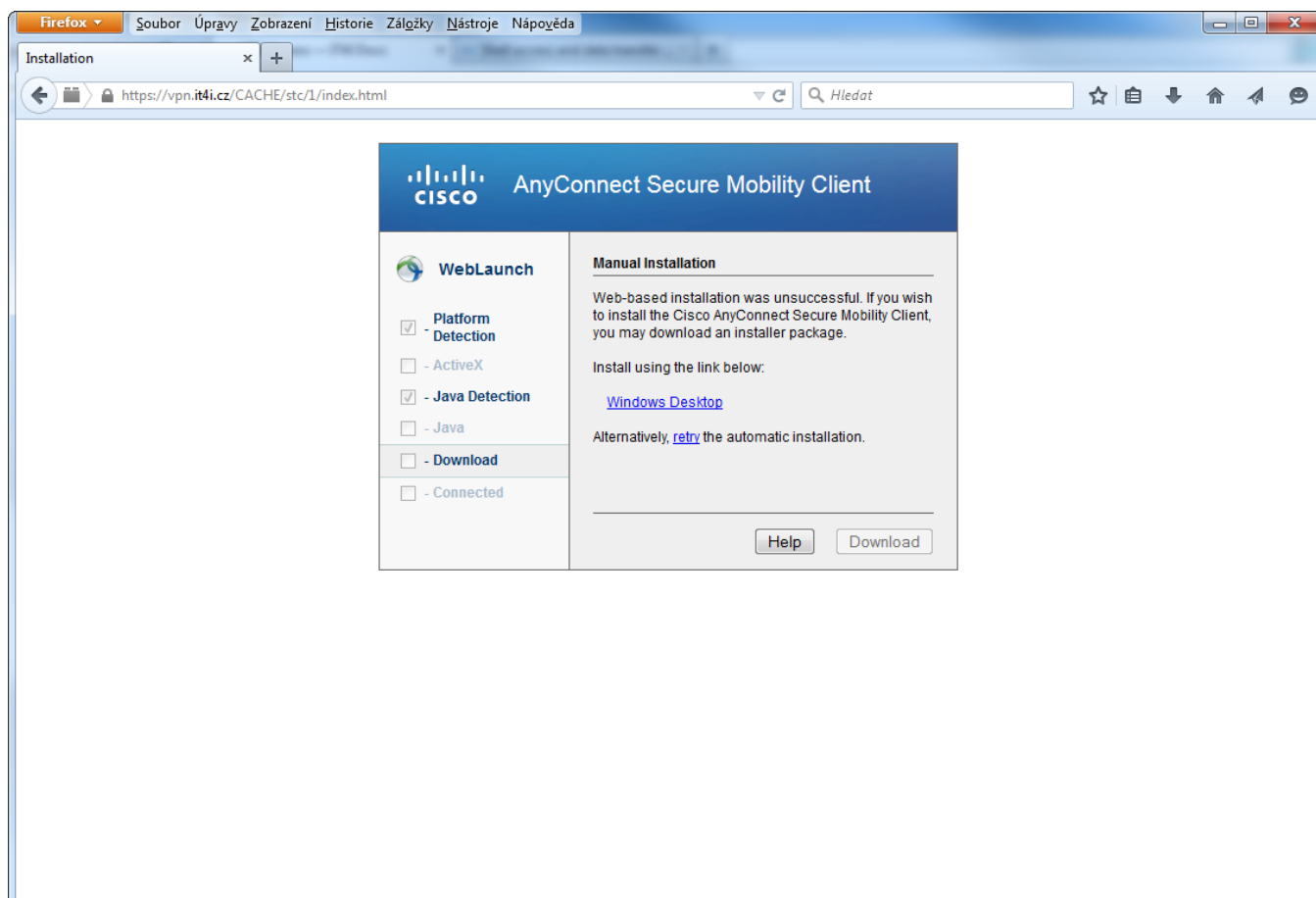


Figure 3:

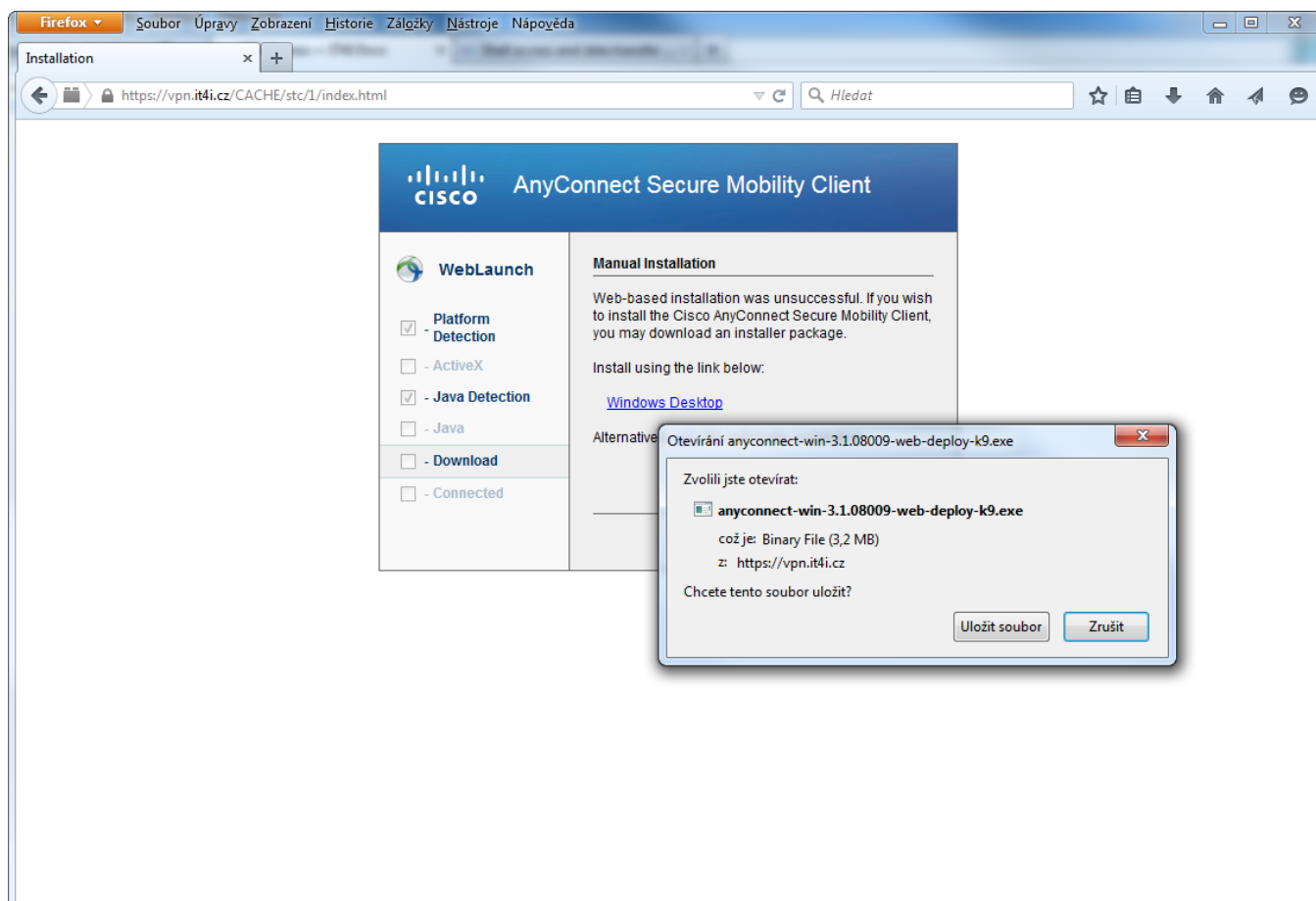


Figure 4:

Before the first login to VPN, you have to fill URL **https://vpn.it4i.cz/user** into the text field.

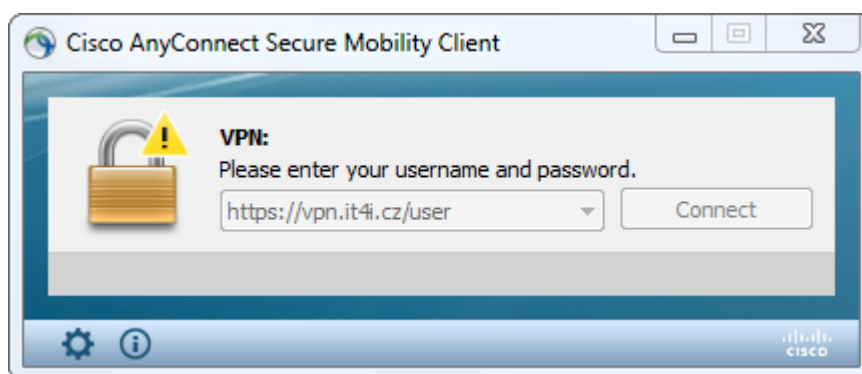


Figure 5:

After you click on the Connect button, you must fill your login credentials.

After a successful login, the client will minimize to the system tray. If everything works, you can see a lock in the Cisco tray icon.

If you right-click on this icon, you will see a context menu in which you can control the VPN connection.

When you connect to the VPN for the first time, the client downloads the profile and creates a new item “IT4I cluster” in the connection list. For subsequent connections, it is not necessary to re-enter the URL address, but just select the corresponding item.

Then AnyConnect automatically proceeds like in the case of first logon.

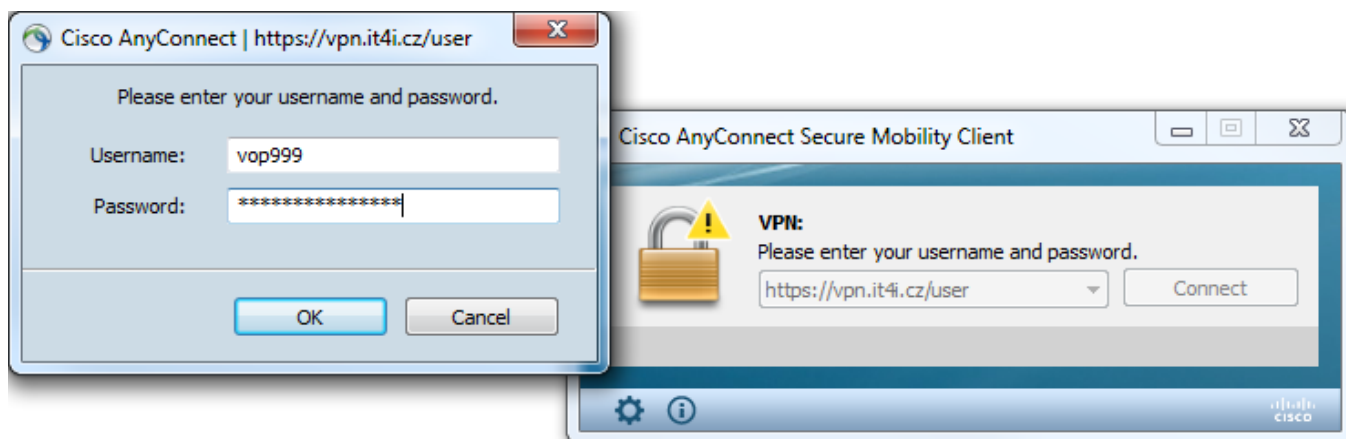


Figure 6:



Figure 7:

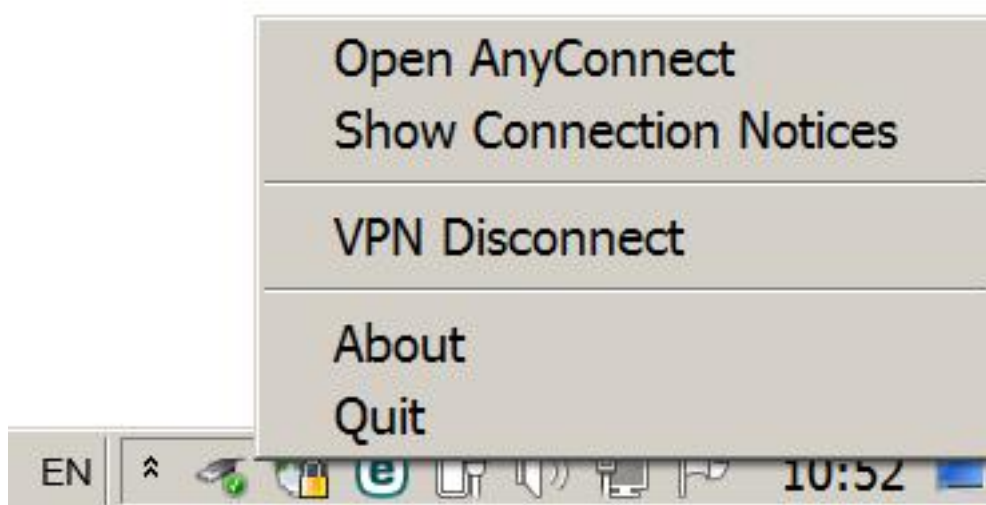


Figure 8:

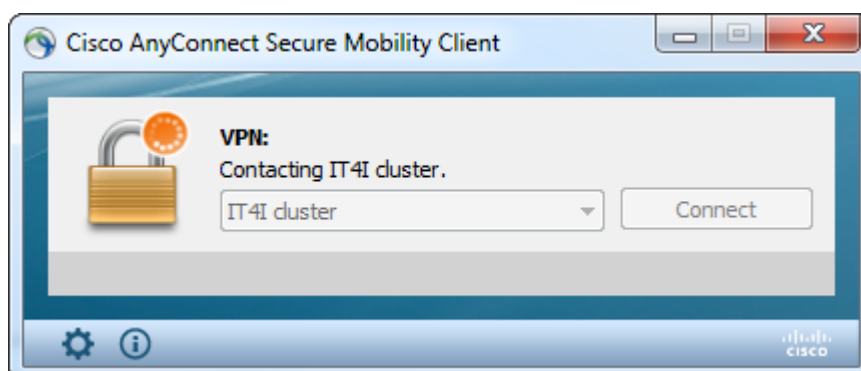


Figure 9:

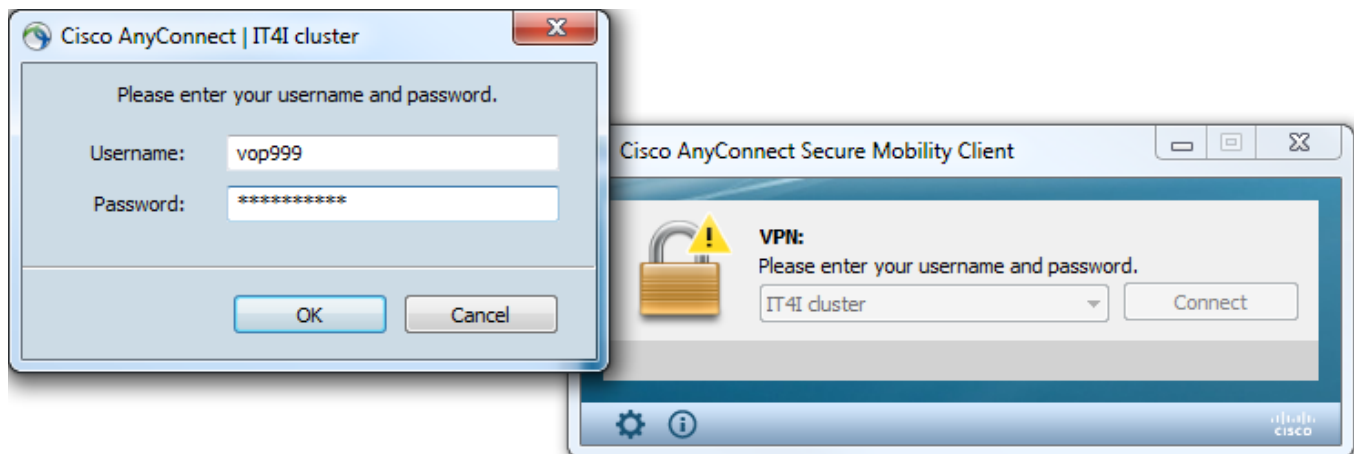


Figure 10:

After a successful login, you can see a green circle with a tick mark on the lock icon.

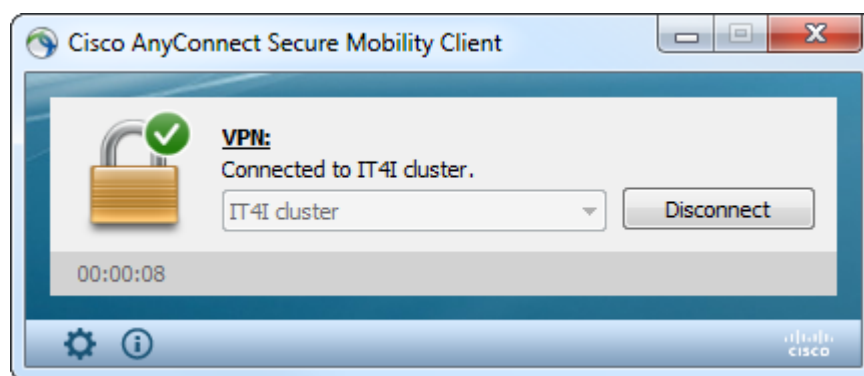




Figure 11:

For disconnecting, right-click on the AnyConnect client icon in the system tray and select **VPN Disconnect**.

Introduction

Welcome to Salomon supercomputer cluster. The Salomon cluster consists of 1008 compute nodes, totaling 24192 compute cores with 129TB RAM and giving over 2 Pflop/s theoretical peak performance. Each node is a powerful x86-64 computer, equipped with 24 cores, at least 128GB RAM. Nodes are interconnected by 7D Enhanced hypercube Infiniband network and equipped with Intel Xeon E5-2680v3 processors. The Salomon cluster consists of 576 nodes without accelerators and 432 nodes equipped with Intel Xeon Phi MIC accelerators. Read more in Hardware Overview.

The cluster runs CentOS Linux  operating system, which is compatible with the RedHat Linux family. 

Water-cooled Compute Nodes With MIC Accelerator



Figure 1:

Tape Library T950B



Figure 2:



Figure 3:



Figure 4:

Compilers

Available compilers, including GNU, INTEL and UPC compilers

There are several compilers for different programming languages available on the cluster:

- C/C++
- Fortran 77/90/95/HPF
- Unified Parallel C
- Java

The C/C++ and Fortran compilers are provided by:

Opensource:

- GNU GCC
- Clang/LLVM

Commercial licenses:

- Intel
- PGI

Intel Compilers

For information about the usage of Intel Compilers and other Intel products, please read the Intel Parallel studio page.

PGI Compilers

The Portland Group Cluster Development Kit (PGI CDK) is available.

```
$ module load PGI
$ pgcc -v
$ pgc++ -v
$ pgf77 -v
$ pgf90 -v
$ pgf95 -v
$ pghpf -v
```

The PGI CDK also includes tools for debugging and profiling.

PGDBG OpenMP/MPI debugger and PGPROF OpenMP/MPI profiler are available

```
$ module load PGI
$ module load Java
$ pgdbg &
$ pgprof &
```

For more information, see the PGI page [PGI](#).

GNU

For compatibility reasons there are still available the original (old 4.4.7-11) versions of GNU compilers as part of the OS. These are accessible in the search path by default.

It is strongly recommended to use the up to date version which comes with the module GCC:

```
$ module load GCC
$ gcc -v
$ g++ -v
$ gfortran -v
```

With the module loaded two environment variables are predefined. One for maximum optimizations on the cluster's architecture, and the other for debugging purposes:

```
$ echo $OPTFLAGS
-O3 -march=native

$ echo $DEBUGFLAGS
-O0 -g
```

For more information about the possibilities of the compilers, please see the man pages.

Unified Parallel C

UPC is supported by two compiler/runtime implementations:

- GNU - SMP/multi-threading support only
- Berkley - multi-node support as well as SMP/multi-threading support

GNU UPC Compiler

To use the GNU UPC compiler and run the compiled binaries use the module gupc

```
$ module add gupc
$ gupc -v
$ g++ -v
```

Simple program to test the compiler

```
$ cat count.upc

/* hello.upc - a simple UPC example */
#include <upc.h>
#include <stdio.h>

int main() {
    if (MYTHREAD == 0) {
        printf("Welcome to GNU UPC!!!\n");
    }
    upc_barrier;
    printf(" - Hello from thread %in", MYTHREAD);
    return 0;
}
```

To compile the example use

```
$ gupc -o count.upc.x count.upc
```

To run the example with 5 threads issue

```
$ ./count.upc.x -fupc-threads-5
```

For more informations see the man pages.

Berkley UPC Compiler

To use the Berkley UPC compiler and runtime environment to run the binaries use the module `bupc`

```
$ module add BerkeleyUPC/2.16.2-gompi-2015b
$ upcc -version
```

As default UPC network the “smp” is used. This is very quick and easy way for testing/debugging, but limited to one node only.

For production runs, it is recommended to use the native Infiniband implementation of UPC network “ibv”. For testing/debugging using multiple nodes, the “mpi” UPC network is recommended. Please note, that the selection of the network is done at the compile time and not at runtime (as expected)!

Example UPC code:

```
$ cat hello.upc

/* hello.upc - a simple UPC example */
#include <upc.h>
#include <stdio.h>

int main() {
    if (MYTHREAD == 0) {
        printf("Welcome to Berkeley UPC!!!\n");
    }
    upc_barrier;
    printf(" - Hello from thread %in", MYTHREAD);
    return 0;
}
```

To compile the example with the “ibv” UPC network use

```
$ upcc -network=ibv -o hello.upc.x hello.upc
```

To run the example with 5 threads issue

```
$ upcrun -n 5 ./hello.upc.x
```

To run the example on two compute nodes using all 48 cores, with 48 threads, issue

```
$ qsub -I -q qprod -A PROJECT_ID -l select=2:ncpus=24
$ module add bupc
$ upcrun -n 48 ./hello.upc.x
```

For more informations see the man pages.

Java

For information how to use Java (runtime and/or compiler), please read the Java page.

nVidia CUDA

For information how to work with nVidia CUDA, please read the nVidia CUDA page.

Operating System

The operating system on Salomon is Linux - **CentOS 6.X**

The CentOS Linux distribution is a stable, predictable, manageable and reproducible platform derived from the sources of Red Hat Enterprise Linux (RHEL).

Intel Advisor

is tool aiming to assist you in vectorization and threading of your code. You can use it to profile your application and identify loops, that could benefit from vectorization and/or threading parallelism.

Installed versions

The following versions are currently available on Salomon as modules:

2016 Update 2 - Advisor/2016__update2

Usage

Your program should be compiled with -g switch to include symbol names. You should compile with -O2 or higher to see code that is already vectorized by the compiler.

Profiling is possible either directly from the GUI, or from command line.

To profile from GUI, launch Advisor:

```
$ advixe-gui
```

Then select menu File -> New -> Project. Choose a directory to save project data to. After clicking OK, Project properties window will appear, where you can configure path to your binary, launch arguments, working directory etc. After clicking OK, the project is ready.

In the left pane, you can switch between Vectorization and Threading workflows. Each has several possible steps which you can execute by clicking Collect button. Alternatively, you can click on Command Line, to see the command line required to run the analysis directly from command line.

References

1. Intel® Advisor 2015 Tutorial: Find Where to Add Parallelism - C++ Sample [!\[\]\(93b46f02aeb0dec7325ae721eddb1f5c_img.jpg\)](#)
2. Product page [!\[\]\(df95500177ee1448bca3fb6d8db555eb_img.jpg\)](#)
3. Documentation [!\[\]\(bb046f17e347213a23b77fd6f78ea23d_img.jpg\)](#)

Intel MKL

Intel Math Kernel Library

Intel Math Kernel Library (Intel MKL) is a library of math kernel subroutines, extensively threaded and optimized for maximum performance. Intel MKL provides these basic math kernels:

- BLAS (level 1, 2, and 3) and LAPACK linear algebra routines, offering vector, vector-matrix, and matrix-matrix operations.
- The PARDISO direct sparse solver, an iterative sparse solver, and supporting sparse BLAS (level 1, 2, and 3) routines for solving sparse systems of equations.
- ScaLAPACK distributed processing linear algebra routines for Linux* and Windows* operating systems, as well as the Basic Linear Algebra Communications Subprograms (BLACS) and the Parallel Basic Linear Algebra Subprograms (PBLAS).
- Fast Fourier transform (FFT) functions in one, two, or three dimensions with support for mixed radices (not limited to sizes that are powers of 2), as well as distributed versions of these functions.
- Vector Math Library (VML) routines for optimized mathematical operations on vectors.
- Vector Statistical Library (VSL) routines, which offer high-performance vectorized random number generators (RNG) for several probability distributions, convolution and correlation routines, and summary statistics functions.
- Data Fitting Library, which provides capabilities for spline-based approximation of functions, derivatives and integrals of functions, and search.
- Extended Eigensolver, a shared memory version of an eigensolver based on the Feast Eigenvalue Solver.

For details see the Intel MKL Reference Manual [\[1\]](#).

Intel MKL version 11.2.3.187 is available on the cluster

```
$ module load imkl
```

The module sets up environment variables, required for linking and running mkl enabled applications. The most important variables are the \$MKLROOT, \$CPATH, \$LD_LIBRARY_PATH and \$MKL_EXAMPLES

Intel MKL library may be linked using any compiler. With intel compiler use -mkl option to link default threaded MKL.

Interfaces

Intel MKL library provides number of interfaces. The fundamental ones are the LP64 and ILP64. The Intel MKL ILP64 libraries use the 64-bit integer type (necessary for indexing large arrays, with more than $2^{31}-1$ elements), whereas the LP64 libraries index arrays with the 32-bit integer type.

Interface	Integer type
LP64	32-bit, int, integer(kind=4), MPI_INT
ILP64	64-bit, long int, integer(kind=8), MPI_INT64

Linking

Linking Intel MKL libraries may be complex. Intel mkl link line advisor [\[2\]](#) helps. See also examples below.

You will need the mkl module loaded to run the mkl enabled executable. This may be avoided, by compiling library search paths into the executable. Include `rpath` on the compile line:

```
$ icc .... -Wl,-rpath=$LIBRARY_PATH ...
```

Threading

Advantage in using Intel MKL library is that it brings threaded parallelization to applications that are otherwise not parallel.

For this to work, the application must link the threaded MKL library (default). Number and behaviour of MKL threads may be controlled via the OpenMP environment variables, such as `OMP_NUM_THREADS` and `KMP_AFFINITY`. `MKL_NUM_THREADS` takes precedence over `OMP_NUM_THREADS`

```
$ export OMP_NUM_THREADS=24
$ export KMP_AFFINITY=granularity=fine,compact,1,0
```

The application will run with 24 threads with affinity optimized for fine grain parallelization.

Examples

Number of examples, demonstrating use of the Intel MKL library and its linking is available on clusters, in the `$MKL_EXAMPLES` directory. In the examples below, we demonstrate linking Intel MKL to Intel and GNU compiled program for multi-threaded matrix multiplication.

Working with examples

```
$ module load intel
$ module load imkl
$ cp -a $MKL_EXAMPLES/cblas /tmp/
$ cd /tmp/cblas

$ make sointel64 function=cblas_dgemm
```

In this example, we compile, link and run the `cblas_dgemm` example, demonstrating use of MKL example suite installed on clusters.

Example: MKL and Intel compiler

```
$ module load intel
$ module load imkl
$ cp -a $MKL_EXAMPLES/cblas /tmp/
$ cd /tmp/cblas
$
$ icc -w source/cblas_dgemmx.c source/common_func.c -mkl -o cblas_dgemmx.x
$ ./cblas_dgemmx.x data/cblas_dgemmx.d
```

In this example, we compile, link and run the `cblas_dgemm` example, demonstrating use of MKL with `icc -mkl` option. Using the `-mkl` option is equivalent to:

```
$ icc -w source/cblas_dgemmx.c source/common_func.c -o cblas_dgemmx.x
-I$MKL_INC_DIR -L$MKL_LIB_DIR -lmkl_intel_lp64 -lmkl_intel_thread -lmkl_core -liomp5
```

In this example, we compile and link the `cblas_dgemm` example, using LP64 interface to threaded MKL and Intel OMP threads implementation.

Example: Intel MKL and GNU compiler

```
$ module load GCC
$ module load imkl
$ cp -a $MKL_EXAMPLES/cblas /tmp/
$ cd /tmp/cblas

$ gcc -w source/cblas_dgemmx.c source/common_func.c -o cblas_dgemmx.x
-lmkl_intel_lp64 -lmkl_gnu_thread -lmkl_core -lgomp -lm

$ ./cblas_dgemmx.x data/cblas_dgemmx.d
```

In this example, we compile, link and run the `cblas_dgemm` example, using LP64 interface to threaded MKL and gnu OMP threads implementation.

MKL and MIC accelerators

The Intel MKL is capable to automatically offload the computations to the MIC accelerator. See section Intel Xeon Phi for details.

LAPACKE C Interface

MKL includes LAPACKE C Interface to LAPACK. For some reason, although Intel is the author of LAPACKE, the LAPACKE header files are not present in MKL. For this reason, we have prepared LAPACKE module, which includes Intel's LAPACKE headers from official LAPACK, which you can use to compile code using LAPACKE interface against MKL.

Further reading

Read more on Intel website [🔗](#), in particular the MKL users guide [🔗](#).

Intel Debugger

IDB is no longer available since Intel Parallel Studio 2015

Debugging serial applications

The intel debugger version 13.0 is available, via module intel. The debugger works for applications compiled with C and C++ compiler and the ifort fortran 77/90/95 compiler. The debugger provides java GUI environment. Use X display for running the GUI.

```
$ module load intel/2014.06
$ module load Java
$ idb
```

The debugger may run in text mode. To debug in text mode, use

```
$ idbc
```

To debug on the compute nodes, module intel must be loaded. The GUI on compute nodes may be accessed using the same way as in the GUI section

Example:

```
$ qsub -q qexp -l select=1:ncpus=24 -X -I
qsub: waiting for job 19654.srv11 to start
qsub: job 19654.srv11 ready

$ module load intel
$ module load Java
$ icc -O0 -g myprog.c -o myprog.x
$ idb ./myprog.x
```

In this example, we allocate 1 full compute node, compile program myprog.c with debugging options -O0 -g and run the idb debugger interactively on the myprog.x executable. The GUI access is via X11 port forwarding provided by the PBS workload manager.

Debugging parallel applications

Intel debugger is capable of debugging multithreaded and MPI parallel programs as well.

Small number of MPI ranks

For debugging small number of MPI ranks, you may execute and debug each rank in separate xterm terminal (do not forget the X display). Using Intel MPI, this may be done in following way:

```
$ qsub -q qexp -l select=2:ncpus=24 -X -I
qsub: waiting for job 19654.srv11 to start
qsub: job 19655.srv11 ready

$ module load intel impi
$ mpirun -ppn 1 -hostfile $PBS_NODEFILE --enable-x xterm -e idbc ./mympprog.x
```

In this example, we allocate 2 full compute node, run xterm on each node and start idb debugger in command line mode, debugging two ranks of mympiprogram application. The xterm will pop up for each rank, with idb prompt ready. The example is not limited to use of Intel MPI

Large number of MPI ranks

Run the idb debugger from within the MPI debug option. This will cause the debugger to bind to all ranks and provide aggregated outputs across the ranks, pausing execution automatically just after startup. You may then set break points and step the execution manually. Using Intel MPI:


```
$ qsub -q qexp -l select=2:ncpus=24 -X -I
qsub: waiting for job 19654.srv11 to start
qsub: job 19655.srv11 ready

$ module load intel impi
$ mpirun -n 48 -idb ./mympiprogram
```

Debugging multithreaded application

Run the idb debugger in GUI mode. The menu Parallel contains number of tools for debugging multiple threads. One of the most useful tools is the **Serialize Execution** tool, which serializes execution of concurrent threads for easy orientation and identification of concurrency related bugs.

Further information

Exhaustive manual on idb features and usage is published at Intel website, <https://software.intel.com/sites/products/documentation/doclib/iss/2013/compiler/cpp-lin/> 

Intel IPP

Intel Integrated Performance Primitives

Intel Integrated Performance Primitives, version 9.0.1, compiled for AVX2 vector instructions is available, via module ipp. The IPP is a very rich library of highly optimized algorithmic building blocks for media and data applications. This includes signal, image and frame processing algorithms, such as FFT, FIR, Convolution, Optical Flow, Hough transform, Sum, MinMax, as well as cryptographic functions, linear algebra functions and many more.

Check out IPP before implementing own math functions for data processing, it is likely already there.

```
$ module load ipp
```

The module sets up environment variables, required for linking and running ipp enabled applications.

IPP example

```
#include "ipp.h"
#include <stdio.h>
int main(int argc, char* argv[])
{
    const IppLibraryVersion *lib;
    Ipp64u fm;
    IppStatus status;

    status= ippInit();           //IPP initialization with the best optimization
    if( status != ippStsNoErr ) {
        printf("IppInit() Error:n");
        printf("%sn", ippGetStatusString(status) );
        return -1;
    }

    //Get version info
    lib = ippiGetLibVersion();
    printf("%s %sn", lib->Name, lib->Version);

    //Get CPU features enabled with selected library level
    fm=ippGetEnabledCpuFeatures();
    printf("SSE      :%cn", (fm>1)&1?'Y':'N');
    printf("SSE2     :%cn", (fm>2)&1?'Y':'N');
    printf("SSE3      :%cn", (fm>3)&1?'Y':'N');
    printf("SSSE3     :%cn", (fm>4)&1?'Y':'N');
    printf("SSE41     :%cn", (fm>6)&1?'Y':'N');
    printf("SSE42     :%cn", (fm>7)&1?'Y':'N');
    printf("AVX       :%cn", (fm>8)&1?'Y':'N');
    printf("AVX2      :%cn", (fm>15)&1?'Y':'N' );
    printf("-----n");
    printf("OS Enabled AVX :%cn", (fm>9)&1?'Y':'N');
    printf("AES           :%cn", (fm>10)&1?'Y':'N');
    printf("CLMUL        :%cn", (fm>11)&1?'Y':'N');
```

```

printf("RDRAND          :%cn", (fm>13)&1?'Y':'N');
printf("F16C           :%cn", (fm>14)&1?'Y':'N');

return 0;
}

```

Compile above example, using any compiler and the ipp module.

```

$ module load intel
$ module load ipp

$ icc testipp.c -o testipp.x -lippi -lipps -lippcore

```

You will need the ipp module loaded to run the ipp enabled executable. This may be avoided, by compiling library search paths into the executable

```

$ module load intel
$ module load ipp

$ icc testipp.c -o testipp.x -Wl,-rpath=$LIBRARY_PATH -lippi -lipps -lippcore

```

Code samples and documentation

Intel provides number of Code Samples for IPP [🔗](#), illustrating use of IPP.

Read full documentation on IPP on Intel website, [🔗](#) in particular the IPP Reference manual. [🔗](#)

Intel TBB

Intel Threading Building Blocks

Intel Threading Building Blocks (Intel TBB) is a library that supports scalable parallel programming using standard ISO C++ code. It does not require special languages or compilers. To use the library, you specify tasks, not threads, and let the library map tasks onto threads in an efficient manner. The tasks are executed by a runtime scheduler and may be offloaded to MIC accelerator.

Intel TBB version 4.3.5.187 is available on the cluster.

```
$ module load tbb
```

The module sets up environment variables, required for linking and running tbb enabled applications.

Link the tbb library, using -ltbb

Examples

Number of examples, demonstrating use of TBB and its built-in scheduler is available on Anselm, in the \$TBB_EXAMPLES directory.

```
$ module load intel
$ module load tbb
$ cp -a $TBB_EXAMPLES/common $TBB_EXAMPLES/parallel_reduce /tmp/
$ cd /tmp/parallel_reduce/primes
$ icc -O2 -DNDEBUG -o primes.x main.cpp primes.cpp -ltbb
$ ./primes.x
```

In this example, we compile, link and run the primes example, demonstrating use of parallel task-based reduce in computation of prime numbers.

You will need the tbb module loaded to run the tbb enabled executable. This may be avoided, by compiling library search paths into the executable.

```
$ icc -O2 -o primes.x main.cpp primes.cpp -Wl,-rpath=$LIBRARY_PATH -ltbb
```

Further reading

Read more on Intel website, http://software.intel.com/sites/products/documentation/doclib/tbb_sa/help/index.htm 

Intel Inspector

Intel Inspector is a dynamic memory and threading error checking tool for C/C++/Fortran applications. It can detect issues such as memory leaks, invalid memory references, uninitialized variables, race conditions, deadlocks etc.

Installed versions

The following versions are currently available on Salomon as modules:

2016 Update 1 - Inspector/2016_update1

Usage

Your program should be compiled with -g switch to include symbol names. Optimizations can be turned on.

Debugging is possible either directly from the GUI, or from command line.

GUI mode

To debug from GUI, launch Inspector:

```
$ inspxe-gui &
```

Then select menu File -> New -> Project. Choose a directory to save project data to. After clicking OK, Project properties window will appear, where you can configure path to your binary, launch arguments, working directory etc. After clicking OK, the project is ready.

In the main pane, you can start a predefined analysis type or define your own. Click Start to start the analysis. Alternatively, you can click on Command Line, to see the command line required to run the analysis directly from command line.

Batch mode

Analysis can be also run from command line in batch mode. Batch mode analysis is run with command inspxe-cl. To obtain the required parameters, either consult the documentation or you can configure the analysis in the GUI and then click “Command Line” button in the lower right corner to the respective command line.

Results obtained from batch mode can be then viewed in the GUI by selecting File -> Open -> Result...

References

1. Product page [!\[\]\(f9e62ae797645c5367e33d9390832789_img.jpg\)](#)
2. Documentation and Release Notes [!\[\]\(3ae06528cbf191565604ae076c36537e_img.jpg\)](#)
3. Tutorials [!\[\]\(1c1752aff31fb3ae93f0f9295ffb3f4c_img.jpg\)](#)

Intel Compilers

The Intel compilers in multiple versions are available, via module intel. The compilers include the icc C and C++ compiler and the ifort fortran 77/90/95 compiler.

```
$ module load intel
$ icc -v
$ ifort -v
```

The intel compilers provide for vectorization of the code, via the AVX2 instructions and support threading parallelization via OpenMP

For maximum performance on the Salomon cluster compute nodes, compile your programs using the AVX2 instructions, with reporting where the vectorization was used. We recommend following compilation options for high performance

```
$ icc -ipo -O3 -xCORE-AVX2 -qopt-report1 -qopt-report-phase=vec myprog.c mysubrouti
$ ifort -ipo -O3 -xCORE-AVX2 -qopt-report1 -qopt-report-phase=vec myprog.f mysubrouti
```

In this example, we compile the program enabling interprocedural optimizations between source files (-ipo), aggressive loop optimizations (-O3) and vectorization (-xCORE-AVX2)

The compiler recognizes the omp, simd, vector and ivdep pragmas for OpenMP parallelization and AVX2 vectorization. Enable the OpenMP parallelization by the **-openmp** compiler switch.

```
$ icc -ipo -O3 -xCORE-AVX2 -qopt-report1 -qopt-report-phase=vec -openmp myprog.c mysu
$ ifort -ipo -O3 -xCORE-AVX2 -qopt-report1 -qopt-report-phase=vec -openmp myprog.f my
```

Read more at <https://software.intel.com/en-us/intel-cplusplus-compiler-16.0-user-and-reference-guide>

Sandy Bridge/Ivy Bridge/Haswell binary compatibility

Anselm nodes are currently equipped with Sandy Bridge CPUs, while Salomon compute nodes are equipped with Haswell based architecture. The UV1 SMP compute server has Ivy Bridge CPUs, which are equivalent to Sandy Bridge (only smaller manufacturing technology). The new processors are backward compatible with the Sandy Bridge nodes, so all programs that ran on the Sandy Bridge processors, should also run on the new Haswell nodes. To get optimal performance out of the Haswell processors a program should make use of the special AVX2 instructions for this processor. One can do this by recompiling codes with the compiler flags designated to invoke these instructions. For the Intel compiler suite, there are two ways of doing this:

- Using compiler flag (both for Fortran and C): -xCORE-AVX2. This will create a binary with AVX2 instructions, specifically for the Haswell processors. Note that the executable will not run on Sandy Bridge/Ivy Bridge nodes.
- Using compiler flags (both for Fortran and C): -xAVX -xCORE-AVX2. This will generate multiple, feature specific auto-dispatch code paths for Intel® processors, if there is a performance benefit. So this binary will run both on Sandy Bridge/Ivy Bridge and Haswell processors. During runtime it will be decided which path to follow, dependent on which processor you are running on. In general this will result in larger binaries.

Intel Parallel Studio

The Salomon cluster provides following elements of the Intel Parallel Studio XE

Intel Parallel Studio XE
Intel Compilers
Intel Debugger
Intel MKL Library
Intel Integrated Performance Primitives Library
Intel Threading Building Blocks Library
Intel Trace Analyzer and Collector
Intel Advisor
Intel Inspector

Intel compilers

The Intel compilers version 131.3 are available, via module iccifort/2013.5.192-GCC-4.8.3. The compilers include the icc C and C++ compiler and the ifort fortran 77/90/95 compiler.

```
$ module load intel
$ icc -v
$ ifort -v
```

Read more at the Intel Compilers page.

Intel debugger

IDB is no longer available since Parallel Studio 2015.

The intel debugger version 13.0 is available, via module intel. The debugger works for applications compiled with C and C++ compiler and the ifort fortran 77/90/95 compiler. The debugger provides java GUI environment.

```
$ module load intel
$ idb
```

Read more at the Intel Debugger page.

Intel Math Kernel Library

Intel Math Kernel Library (Intel MKL) is a library of math kernel subroutines, extensively threaded and optimized for maximum performance. Intel MKL unites and provides these basic components: BLAS, LAPACK, ScaLapack, PARDISO, FFT, VML, VSL, Data fitting, Feast Eigensolver and many more.

```
$ module load imkl
```

Read more at the Intel MKL page.

Intel Integrated Performance Primitives

Intel Integrated Performance Primitives, version 7.1.1, compiled for AVX is available, via module `ipp`. The IPP is a library of highly optimized algorithmic building blocks for media and data applications. This includes signal, image and frame processing algorithms, such as FFT, FIR, Convolution, Optical Flow, Hough transform, Sum, MinMax and many more.

```
$ module load ipp
```

Read more at the Intel IPP page.

Intel Threading Building Blocks

Intel Threading Building Blocks (Intel TBB) is a library that supports scalable parallel programming using standard ISO C++ code. It does not require special languages or compilers. It is designed to promote scalable data parallel programming. Additionally, it fully supports nested parallelism, so you can build larger parallel components from smaller parallel components. To use the library, you specify tasks, not threads, and let the library map tasks onto threads in an efficient manner.

```
$ module load tbb
```

Read more at the Intel TBB page.

Intel Trace Analyzer and Collector

Intel Trace Analyzer and Collector (ITAC) is a tool to collect and graphically analyze behaviour of MPI applications. It helps you to analyze communication patterns of your application, identify hotspots, perform correctness checking (identify deadlocks, data corruption etc), simulate how your application would run on a different interconnect.

ITAC is an offline analysis tool - first you run your application to collect a trace file, then you can open the trace in a GUI analyzer to view it.

Installed version

Currently on Salomon is version 9.1.2.024 available as module `itac/9.1.2.024`

Collecting traces

ITAC can collect traces from applications that are using Intel MPI. To generate a trace, simply add `-trace` option to your `mpirun` command :

```
$ module load itac/9.1.2.024
$ mpirun -trace myapp
```

The trace will be saved in file `myapp.stf` in the current directory.

Viewing traces

To view and analyze the trace, open ITAC GUI in a graphical environment:

```
$ module load itac/9.1.2.024
$ traceanalyzer
```

The GUI will launch and you can open the produced `*.stf` file.

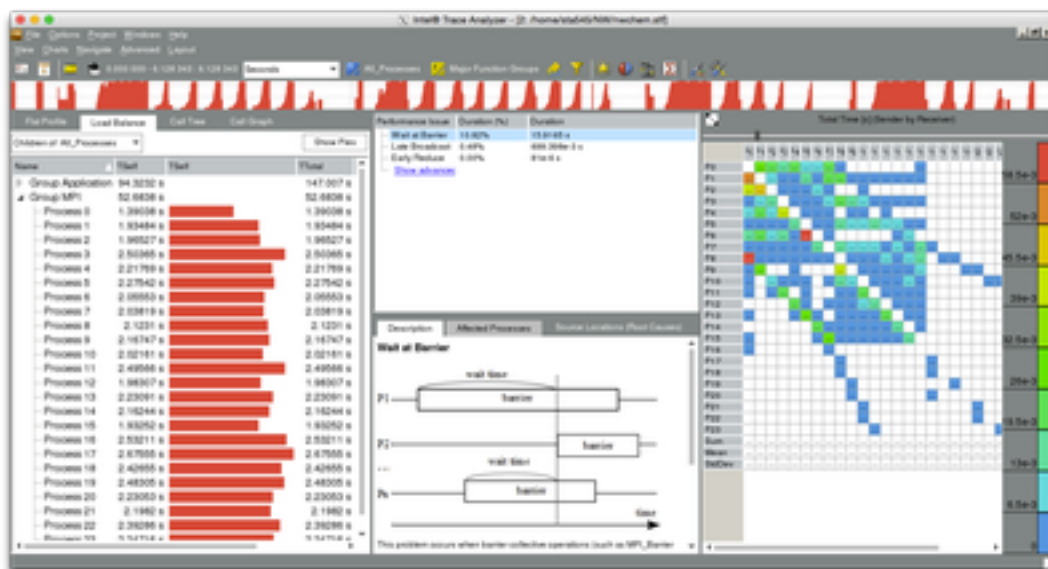


Figure 1:

Please refer to Intel documentation about usage of the GUI tool.

References

1. Getting Started with Intel® Trace Analyzer and Collector 
2. Intel® Trace Analyzer and Collector - Documentation 

MPI

Setting up MPI Environment

The Salomon cluster provides several implementations of the MPI library:

MPI Library	Thread support
Intel MPI 4.1	Full thread support up to, MPI_THREAD_MULTIPLE
Intel MPI 5.0	Full thread support up to, MPI_THREAD_MULTIPLE
OpenMPI 1.8.6	Full thread support up to, MPI_THREAD_MULTIPLE, MPI- 3.0, support
SGI MPT 2.12	

MPI libraries are activated via the environment modules.

Look up section modulefiles/mpi in module avail

```
$ module avail
----- /apps/modules/mpi -----
impi/4.1.1.036-iccifort-2013.5.192
impi/4.1.1.036-iccifort-2013.5.192-GCC-4.8.3
impi/5.0.3.048-iccifort-2015.3.187
impi/5.0.3.048-iccifort-2015.3.187-GNU-5.1.0-2.25
```

MPT/2.12

OpenMPI/1.8.6-GNU-5.1.0-2.25

There are default compilers associated with any particular MPI implementation. The defaults may be changed, the MPI libraries may be used in conjunction with any compiler. The defaults are selected via the modules in following way

Module	MPI
impi-5.0.3.048-iccifort- Intel MPI 5.0.3	2015.3.187
OpenMP-1.8.6-GNU-5.1.0-2 OpenMPI 1.8.6	.25

Examples:

```
$ module load gomp/2015b
```

In this example, we activate the latest OpenMPI with latest GNU compilers (OpenMPI 1.8.6 and GCC 5.1). Please see more information about toolchains in section Environment and Modules .

To use OpenMPI with the intel compiler suite, use

```
$ module load iompi/2015.03
```

In this example, the openmpi 1.8.6 using intel compilers is activated. It's used "iompi" toolchain.

Compiling MPI Programs

After setting up your MPI environment, compile your program using one of the mpi wrappers

```
$ mpicc -v
$ mpif77 -v
$ mpif90 -v
```

When using Intel MPI, use the following MPI wrappers:

```
$ mpicc
$ mpiifort
```

Wrappers mpif90, mpif77 that are provided by Intel MPI are designed for gcc and gfortran. You might be able to compile MPI code by them even with Intel compilers, but you might run into problems (for example, native MIC compilation with -mmic does not work with mpif90).

Example program:

```
// helloworld_mpi.c
#include <stdio.h>

#include<mpi.h>

int main(int argc, char **argv) {

    int len;
    int rank, size;
    char node[MPI_MAX_PROCESSOR_NAME];

    // Initiate MPI
    MPI_Init(&argc, &argv);
```



```

MPI_Comm_rank(MPI_COMM_WORLD,&rank);
MPI_Comm_size(MPI_COMM_WORLD,&size);

// Get hostame and print
MPI_Get_processor_name(node,&len);
printf("Hello world! from rank %d of %d on host %sn",rank,size,node);

// Finalize and exit
MPI_Finalize();

return 0;
}

```

Compile the above example with

```
$ mpicc helloworld_mpi.c -o helloworld_mpi.x
```

Running MPI Programs

The MPI program executable must be compatible with the loaded MPI module. Always compile and execute using the very same MPI module.

It is strongly discouraged to mix mpi implementations. Linking an application with one MPI implementation and running mpirun/mpiexec from other implementation may result in unexpected errors.

The MPI program executable must be available within the same path on all nodes. This is automatically fulfilled on the /home and /scratch filesystem. You need to preload the executable, if running on the local scratch /lscratch filesystem.

Ways to run MPI programs

Optimal way to run an MPI program depends on its memory requirements, memory access pattern and communication pattern.

Consider these ways to run an MPI program: 1. One MPI process per node, 24 threads per process 2. Two MPI processes per node, 12 threads per process 3. 24 MPI processes per node, 1 thread per process.

One MPI process per node, using 24 threads, is most useful for memory demanding applications, that make good use of processor cache memory and are not memory bound. This is also a preferred way for communication intensive applications as one process per node enjoys full bandwidth access to the network interface.


Two MPI processes per node, using 12 threads each, bound to processor socket is most useful for memory bandwidth bound applications such as BLAS1 or FFT, with scalable memory demand. However, note that the two processes will share access to the network interface. The 12 threads and socket binding should ensure maximum memory access bandwidth and minimize communication, migration and numa effect overheads.

!!! Note “Note” Important! Bind every OpenMP thread to a core!

In the previous two cases with one or two MPI processes per node, the operating system might still migrate OpenMP threads between cores. You want to avoid this by setting the KMP_AFFINITY or GOMP_CPU_AFFINITY environment variables.

24 MPI processes per node, using 1 thread each bound to processor core is most suitable for highly scalable applications with low communication demand.

Running OpenMPI

The **OpenMPI 1.8.6** is based on OpenMPI. Read more on how to run OpenMPI based MPI.

The Intel MPI may run on the Intel Xeon Phi accelerators as well. Read more on how to run Intel MPI on accelerators.

Running OpenMPI

OpenMPI program execution

The OpenMPI programs may be executed only via the PBS Workload manager, by entering an appropriate queue. On the cluster, the **OpenMPI 1.8.6** is OpenMPI based MPI implementation.

Basic usage

Use the `mpiexec` to run the OpenMPI code.

Example:

```
$ qsub -q qexp -l select=4:ncpus=24 -I
qsub: waiting for job 15210.isrv5 to start
qsub: job 15210.isrv5 ready

$ pwd
/home/username

$ module load OpenMPI
$ mpiexec -pernode ./helloworld_mpi.x
Hello world! from rank 0 of 4 on host r1i0n17
Hello world! from rank 1 of 4 on host r1i0n5
Hello world! from rank 2 of 4 on host r1i0n6
Hello world! from rank 3 of 4 on host r1i0n7
```

Please be aware, that in this example, the directive **-pernode** is used to run only **one task per node**, which is normally an unwanted behaviour (unless you want to run hybrid code with just one MPI and 24 OpenMP tasks per node). In normal MPI programs **omit the -pernode directive** to run up to 24 MPI tasks per each node.

In this example, we allocate 4 nodes via the express queue interactively. We set up the openmpi environment and interactively run the `helloworld_mpi.x` program. Note that the executable `helloworld_mpi.x` must be available within the same path on all nodes. This is automatically fulfilled on the `/home` and `/scratch` filesystem.

You need to preload the executable, if running on the local ramdisk `/tmp` filesystem

```
$ pwd
/tmp/pbs.15210.isrv5

$ mpiexec -pernode --preload-binary ./helloworld_mpi.x
Hello world! from rank 0 of 4 on host r1i0n17
Hello world! from rank 1 of 4 on host r1i0n5
Hello world! from rank 2 of 4 on host r1i0n6
Hello world! from rank 3 of 4 on host r1i0n7
```

In this example, we assume the executable `helloworld_mpi.x` is present on compute node `r1i0n17` on ramdisk. We call the `mpiexec` with the **--preload-binary** argument (valid for `openmpi`). The `mpiexec` will copy the executable from `r1i0n17` to the `/tmp/pbs.15210.isrv5` directory on `r1i0n5`, `r1i0n6` and `r1i0n7` and execute the program.

MPI process mapping may be controlled by PBS parameters.

The `mpiprocs` and `ompthreads` parameters allow for selection of number of running MPI processes per node as well as number of OpenMP threads per MPI process.

One MPI process per node

Follow this example to run one MPI process per node, 24 threads per process.

```
$ qsub -q qexp -l select=4:ncpus=24:mpiprocs=1:ompthreads=24 -I

$ module load OpenMPI

$ mpiexec --bind-to-none ./helloworld_mpi.x
```

In this example, we demonstrate recommended way to run an MPI application, using 1 MPI processes per node and 24 threads per socket, on 4 nodes.

Two MPI processes per node

Follow this example to run two MPI processes per node, 8 threads per process. Note the options to `mpiexec`.

```
$ qsub -q qexp -l select=4:ncpus=24:mpiprocs=2:ompthreads=12 -I

$ module load OpenMPI

$ mpiexec -bysocket -bind-to-socket ./helloworld_mpi.x
```

In this example, we demonstrate recommended way to run an MPI application, using 2 MPI processes per node and 12 threads per socket, each process and its threads bound to a separate processor socket of the node, on 4 nodes

24 MPI processes per node

Follow this example to run 24 MPI processes per node, 1 thread per process. Note the options to `mpiexec`.

```
$ qsub -q qexp -l select=4:ncpus=24:mpiprocs=24:ompthreads=1 -I

$ module load OpenMPI

$ mpiexec -bycore -bind-to-core ./helloworld_mpi.x
```

In this example, we demonstrate recommended way to run an MPI application, using 24 MPI processes per node, single threaded. Each process is bound to separate processor core, on 4 nodes.

OpenMP thread affinity

!!! Note “Note” Important! Bind every OpenMP thread to a core!

In the previous two examples with one or two MPI processes per node, the operating system might still migrate OpenMP threads between cores. You might want to avoid this by setting these environment variable for GCC OpenMP:

```
$ export GOMP_CPU_AFFINITY="0-23"
```

or this one for Intel OpenMP:

```
$ export KMP_AFFINITY=granularity=fine,compact,1,0
```

As of OpenMP 4.0 (supported by GCC 4.9 and later and Intel 14.0 and later) the following variables may be used for Intel or GCC:

```
$ export OMP_PROC_BIND=true
$ export OMP_PLACES=cores
```

OpenMPI Process Mapping and Binding

The `mpiexec` allows for precise selection of how the MPI processes will be mapped to the computational nodes and how these processes will bind to particular processor sockets and cores.

MPI process mapping may be specified by a hostfile or rankfile input to the `mpiexec` program. Although all implementations of MPI provide means for process mapping and binding, following examples are valid for the `openmpi` only.

Hostfile

Example hostfile

```
r1i0n17.smc.salomon.it4i.cz
r1i0n5.smc.salomon.it4i.cz
r1i0n6.smc.salomon.it4i.cz
r1i0n7.smc.salomon.it4i.cz
```

Use the hostfile to control process placement

```
$ mpiexec -hostfile hostfile ./helloworld_mpi.x
Hello world! from rank 0 of 4 on host r1i0n17
Hello world! from rank 1 of 4 on host r1i0n5
Hello world! from rank 2 of 4 on host r1i0n6
Hello world! from rank 3 of 4 on host r1i0n7
```

In this example, we see that ranks have been mapped on nodes according to the order in which nodes show in the hostfile

Rankfile

Exact control of MPI process placement and resource binding is provided by specifying a rankfile

Appropriate binding may boost performance of your application.

Example rankfile

```
rank 0=r1i0n7.smc.salomon.it4i.cz slot=1:0,1
rank 1=r1i0n6.smc.salomon.it4i.cz slot=0:*
rank 2=r1i0n5.smc.salomon.it4i.cz slot=1:1-2
rank 3=r1i0n17.smc.salomon slot=0:1,1:0-2
rank 4=r1i0n6.smc.salomon.it4i.cz slot=0:*,1:*
```

This rankfile assumes 5 ranks will be running on 4 nodes and provides exact mapping and binding of the processes to the processor sockets and cores

MPI4Py (MPI for Python)

OpenMPI interface to Python

Introduction

MPI for Python provides bindings of the Message Passing Interface (MPI) standard for the Python programming language, allowing any Python program to exploit multiple processors.

This package is constructed on top of the MPI-1/2 specifications and provides an object oriented interface which closely follows MPI-2 C++ bindings. It supports point-to-point (sends, receives) and collective (broadcasts, scatters, gathers) communications of any picklable Python object, as well as optimized communications of Python object exposing the single-segment buffer interface (NumPy arrays, builtin bytes/string/array objects).

On Anselm MPI4Py is available in standard Python modules.

Modules

MPI4Py is build for OpenMPI. Before you start with MPI4Py you need to load Python and OpenMPI modules. You can use toolchain, that loads Python and OpenMPI at once.

```
$ module load Python/2.7.9-foss-2015g
```

Execution

You need to import MPI to your python program. Include the following line to the python script:

```
from mpi4py import MPI
```

The MPI4Py enabled python programs execute as any other OpenMPI code. The simplest way is to run

```
$ mpiexec python <script>.py
```

For example

```
$ mpiexec python hello_world.py
```

Examples

Hello world!

```
from mpi4py import MPI

comm = MPI.COMM_WORLD

print "Hello! I'm rank %d from %d running in total..." % (comm.rank, comm.size)

comm.Barrier()  # wait for everybody to synchronize
```

Collective Communication with NumPy arrays

```
from __future__ import division
from mpi4py import MPI
import numpy as np

comm = MPI.COMM_WORLD

print("-"*78)
print(" Running on %d cores" % comm.size)
print("-"*78)

comm.Barrier()

# Prepare a vector of N=5 elements to be broadcasted...
N = 5
if comm.rank == 0:
    A = np.arange(N, dtype=np.float64)    # rank 0 has proper data
else:
    A = np.empty(N, dtype=np.float64)    # all other just an empty array

# Broadcast A from rank 0 to everybody
comm.Bcast( [A, MPI.DOUBLE] )

# Everybody should now have the same...
print "[%02d] %s" % (comm.rank, A)
```

Execute the above code as:

```
$ qsub -q qexp -l select=4:ncpus=24:mpiprocs=24:omphreads=1 -I

$ module load Python/2.7.9-foss-2015g

$ mpiexec --map-by core --bind-to core python hello_world.py
```

In this example, we run MPI4Py enabled code on 4 nodes, 24 cores per node (total of 96 processes), each python process is bound to a different core. More examples and documentation can be found on MPI for Python webpage [here](#).

Package ‘parallel’

R-core

May 16, 2013

1 Introduction

Package **parallel** was first included in R 2.14.0. It builds on the work done for CRAN packages **multicore** (Urbanek, 2009–present) and **snow** (Tierney *et al.*, 2003–present) and provides drop-in replacements for most of the functionality of those packages, with integrated handling of random-number generation.

Parallelism can be done in computation at many different levels: this package is principally concerned with ‘coarse-grained parallelization’. At the lowest level, modern CPUs can do several basic operations simultaneously (e.g. integer and floating-point arithmetic), and several implementations of external BLAS libraries use multiple threads to do parts of basic vector/matrix operations in parallel. Several contributed R packages use multiple threads at C level *via* OpenMP or pthreads.

This package handles running much larger chunks of computations in parallel. A typical example is to evaluate the same R function on many different sets of data: often simulated data as in bootstrap computations (or with ‘data’ being the random-number stream). The crucial point is that these chunks of computation are unrelated and do not need to communicate in any way. It is often the case that the chunks take approximately the same length of time. The basic computational model is

- (a) Start up M ‘worker’ processes, and do any initialization needed on the workers.
- (b) Send any data required for each task to the workers.
- (c) Split the task into M roughly equally-sized chunks, and send the chunks (including the R code needed) to the workers.
- (d) Wait for all the workers to complete their tasks, and ask them for their results.
- (e) Repeat steps (b–d) for any further tasks.
- (f) Shut down the worker processes.

Amongst the initializations which may be needed are to load packages and initialize the random-number stream.

There are implementations of this model in the functions `mclapply` and `parLapply` as near-drop-in replacements for `lapply`.

A slightly different model is to split the task into $M_1 > M$ chunks, send the first M chunks to the workers, then repeatedly wait for any worker to complete and send it the next remaining task: see the section on ‘load balancing’.

In principle the workers could be implemented by threads¹ or lightweight processes, but in the

¹only ‘in principle’ since the R interpreter is not thread-safe.

current implementation they are full processes. They can be created in one of three ways:

1. *Via* `system("Rscript")` or similar to launch a new process on the current machine or a similar machine with an identical R installation. This then needs a way to communicate between master and worker processes, which is usually done *via* sockets.

This should be available on all R platforms, although it is conceivable that zealous security measures could block the inter-process communication *via* sockets. Users of Windows and OS X may expect pop-up dialog boxes from the firewall asking if an R process should accept incoming connections.

Following **snow**, a pool of worker processes listening *via* sockets for commands from the master is called a ‘cluster’ of nodes.

2. *Via* forking. *Fork* is a concept² from POSIX operating systems, and should be available on all R platforms except Windows. This creates a new R process by taking a complete copy of the master process, including the workspace and state of the random-number stream. However, the copy will (in any reasonable OS) share memory pages with the master until modified so forking is very fast.

The use of forking was pioneered by package **multicore**.

Note that as it does share the complete process, it also shares any GUI elements, for example an R console and on-screen devices. This can cause havoc.³

There needs to be a way to communicate between master and worker. Once again there are several possibilities since master and workers share memory. In **multicore** the initial fork sends an R expression to be evaluated to the worker, and the master process opens a pipe for reading that is used by the worker to return the results. Both that and creating a cluster of nodes communicating *via* sockets are supported in package **parallel**.

3. Using OS-level facilities to set up a means to send tasks to other members of a group of machines. There are several ways to do that, and for example package **snow** can make use of MPI (‘message passing interface’) using R package **Rmpi**. Communication overheads can dominate computation times in this approach, so it is most often used on tightly-coupled networks of computers with high-speed interconnects.

CRAN packages following this approach include **GridR** (using Condor or Globus) and **Rsgs** (using SGE, currently called ‘Oracle Grid Engine’).

It will not be considered further in this vignette, but those parts of **parallel** which provide **snow**-like functions will accept **snow** clusters including MPI clusters.

The landscape of parallel computing has changed with the advent of shared-memory computers with multiple (and often many) CPU cores. Until the late 2000’s parallel computing was mainly done on clusters of large numbers of single- or dual-CPU computers: nowadays even laptops have two or four cores, and servers with 8 or more cores are commonplace. It is such hardware that package **parallel** is designed to exploit. It can also be used with several computers running the same version of R connected by (reasonable-speed) ethernet: the computers need not be running the same OS.

Note that all these methods of communication use `serialize/unserialize` to send R objects between processes. This has limits (typically hundreds of millions of elements) which a well-designed parallelized algorithm should not approach.

²[http://en.wikipedia.org/wiki/Fork_\(operating_system\)](http://en.wikipedia.org/wiki/Fork_(operating_system))

³Some precautions are taken on Mac OS X: for example the event loops for **R.app** and the **quartz** device are inhibited in the child. This information is available at C level in the **Rboolean** variable `R_isForkedChild`.

2 Numbers of CPUs/cores

In setting up parallel computations it can be helpful to have some idea of the number of CPUs or cores available, but this is a rather slippery concept. Nowadays almost all physical CPUs contain two or more cores that run more-or-less independently (they may share parts of the cache memory, and they do share access to RAM). However, on some processors these cores may themselves be able to run multiple tasks simultaneously, and some OSes (e.g. Windows) have the concept of *logical* CPUs which may exceed the number of cores.

Note that all a program can possibly determine is the total number of CPUs and/or cores available. This is not necessarily the same as the number of CPUs available *to the current user* which may well be restricted by system policies on multi-user systems. Nor does it give much idea of a reasonable number of CPUs to use for the current task: the user may be running many R processes simultaneously, and those processes may themselves be using multiple threads through a multi-threaded BLAS, compiled code using OpenMP or other low-level forms of parallelism. We have even seen instances of **multicore**'s `mclapply` being called recursively,⁴ generating $2n+n^2$ processes on a machine estimated to have $n = 16$ cores.

But in so far as it is a useful guideline, function `detectCores()` tries to determine the number of CPU cores in the machine on which R is running: it has ways to do so on all known current R platforms. What exactly it measures is OS-specific: we try where possible to report the number of physical cores available.

On Windows the default is to report the number of logical CPUs. On modern hardware (e.g. Intel *Core i7*) the latter may not be unreasonable as hyper-threading does give a significant extra throughput. What `detectCores(logical = FALSE)` reports is OS-version-dependent: on recent versions of Windows it reports the number of physical cores but on older versions it might report the number of physical CPU packages.

3 Analogues of apply functions

By far the most common direct applications of packages **multicore** and **snow** have been to provide parallelized replacements of `lapply`, `sapply`, `apply` and related functions.

As analogues of `lapply` there are

```
parLapply(cl, x, FUN, ...)  
mclapply(X, FUN, ..., mc.cores)
```

where `mclapply` is not available⁵ on Windows and has further arguments discussed on its help page. They differ slightly in philosophy: `mclapply` sets up a pool of `mc.cores` workers just for this computation, whereas `parLapply` uses a less ephemeral pool specified by the object `cl` created by a call to `makeCluster` (which *inter alia* specifies the size of the pool). So the workflow is

```
cl <- makeCluster(<size of pool>)  
# one or more parLapply calls  
stopCluster(cl)
```

For matrices there are the rarely used `parApply` and `parCapply` functions, and the more commonly used `parRapply`, a parallel row `apply` for a matrix.

⁴`parallel`: `mclapply` detects this and runs nested calls serially.

⁵except as a stub which simply calls `lapply`.

4 SNOW Clusters

The package contains a slightly revised copy of much of **snow**, and the functions it contains can also be used with clusters created by **snow** (provided the package is on the search path).

Two functions are provided to create SNOW clusters, **makePSOCKcluster** (a streamlined version of **snow::makeSOCKcluster**) and (except on Windows) **makeForkCluster**. They differ only in the way they spawn worker processes: **makePSOCKcluster** uses **Rscript** to launch further copies of R (on the same host or optionally elsewhere) whereas **makeForkCluster** forks the workers on the current host (which thus inherit the environment of the current session).

These functions would normally be called *via* **makeCluster**.

Both **stdout()** and **stderr()** of the workers are redirected, by default being discarded but they can be logged using the **outfile** option. Note that the previous sentence refers to the *connections* of those names, not the C-level file handles. Thus properly written R packages using **Rprintf** will have their output redirected, but not direct C-level output.

A default cluster can be registered by a call to **setDefaultCluster()**: this is then used whenever one of the higher-level functions such as **parApply** is called without an explicit cluster. A little care is needed when repeatedly re-using a pool of workers, as their workspaces will accumulate objects from past usage, and packages may get added to the search path.

If clusters are to be created on a host other than the current machine ('localhost'), **makeCluster** may need to be given more information in the shape of extra arguments.

- If the worker machines are not set up in exactly the same way as the master (for example if they are of a different architecture), use **homogeneous = FALSE** and perhaps set **rscript** to the full path to **Rscript** on the workers.
- The worker machines need to know how to communicate with the master: normally this can be done using the hostname found by **Sys.info()**, but on private networks this need not be the case and **master** may need to be supplied as a name or IP address, e.g. **master = "192.168.1.111"**.
- By default **ssh** is used to launch R on the workers. If it is known by some other name, use e.g. **rshcmd = "plink.exe"** for a Windows box using PUTTY. SSH should be set up to use silent authentication: setups which require a password to be supplied may or may not work.
- Socket communication is done over port a randomly chosen port in the range 11000:11999: site policies might require some other port to be used, in which case set argument **port** or environment variable **R_PARALLEL_PORT**.

5 Forking

Except on Windows, the package contains a copy of **multicore**: there a few names with the added prefix **mc**, e.g. **mccollect** and **mcparallel**. (Package **multicore** used these names, but also the versions without the prefix which are too easily masked: e.g. package **lattice** has a function **parallel**.)

The low-level functions from **multicore** are provided but not exported from the namespace.

There are high-level functions **mclapply** and **pvec**: unlike the versions in **multicore** these default to 2 cores, but this can be controlled by setting **options("mc.cores")**, and that takes its default from environment variable **MC_CORES** when the package is loaded. (Setting this to 1

inhibits parallel operation: there are stub versions of these functions on Windows which force `mc.cores = 1.`)

As from R 2.15.0, `mcmapply` and `mcMap` provide analogues of `mapply` and `Map`.

Note the earlier comments about using forking in a GUI environment.

The parent and forked R processes share the per-session temporary directory `tempdir()`, which can be a problem as a lot of code has assumed it is private to the R process. Further, prior to R 2.14.1 it was possible for `tempfile` in two processes to select the same filename in that temporary directory, and do it sufficiently simultaneously that neither saw it as being in use.

The forked workers share file handles with the master: this means that any output from the worker should go to the same place as `'stdout'` and `'stderr'` of the master process. (This does not work reliably on all OSes: problems have also been noted when forking a session that is processing batch input from `'stdin'`.) Setting argument `mc.silent = TRUE` silences `'stdout'` for the child: `'stderr'` is not affected.

Sharing file handles also impacts graphics devices as forked workers inherit all open graphics devices of the parent: they should not attempt to make use of them.

6 Random-number generation

Some care is needed with parallel computation using (pseudo-)random numbers: the processes/threads which run separate parts of the computation need to run independent (and preferably reproducible) random-number streams. One way to avoid any difficulties is (where possible) to do all the randomization in the master process: this is done where possible in package **boot** (version 1.3-1 and later).

When an R process is started up it takes the random-number seed from the object `.Random.seed` in a saved workspace or constructs one from the clock time and process ID when random-number generation is first used (see the help on `RNG`). Thus worker processes might get the same seed because a workspace containing `.Random.seed` was restored or the random number generator has been used before forking: otherwise these get a non-reproducible seed (but with very high probability a different seed for each worker).

The alternative is to set separate seeds for each worker process in some reproducible way from the seed in the master process. This is generally plenty safe enough, but there have been worries that the random-number streams in the workers might somehow get into step. One approach is to take the seeds a long way apart in the random-number stream: note that random numbers taken a long (fixed) distance apart in a single stream are not necessarily (and often are not) as independent as those taken a short distance apart. Yet another idea (as used by e.g. **JAGS**) is to use different random-number generators for each separate run/process.

Package **parallel** contains an implementation of the ideas of L'Ecuyer *et al.* (2002): this uses a single RNG and make *streams* with seeds 2^{127} steps apart in the random number stream (which has period approximately 2^{191}). This is based on the generator of L'Ecuyer (1999); the reason for choosing that generator⁶ is that it has a fairly long period with a small seed (6 integers), and unlike R's default "Mersenne-Twister" RNG, it is simple to advance the seed by a fixed

⁶apart from the commonality of authors!

number of steps. The generator is the combination of two:

$$\begin{aligned}x_n &= 1403580 \times x_{n-2} - 810728 \times x_{n-3} \mod (2^{32} - 209) \\y_n &= 527612 \times y_{n-1} - 1370589 \times y_{n-3} \mod (2^{32} - 22853) \\z_n &= (x_n - y_n) \mod 4294967087 \\u_n &= z_n / 4294967088 \text{ unless } z_n = 0\end{aligned}$$

The ‘seed’ then consists of $(x_n, x_{n-1}, x_{n-2}, y_n, y_{n-1}, y_{n-2})$, and the recursion for each of x_n and y_n can have pre-computed coefficients for k steps ahead. For $k = 2^{127}$, the seed is advanced by k steps by R call `.Random.seed <- nextRNGStream(.Random.seed)`.

The L’Ecuyer (1999) generator is available in R as from version 2.14.0 *via* `RNGkind("L'Ecuyer-CMRG")`. Thus using the ideas of L’Ecuyer *et al.* (2002) is as simple as

```
RNGkind("L'Ecuyer-CMRG")
set.seed(<something>)
## start M workers
s <- .Random.seed
for (i in 1:M) {
  s <- nextRNGStream(s)
  # send s to worker i as .Random.seed
}
```

and this is implemented for SNOW clusters in function `clusterSetRNGStream`, and as part of `mclapply` and `mclapply` (by default).

Apart from *streams* (2^{127} steps apart), there is the concept of *sub-streams* starting from seeds 2^{76} steps apart. Function `nextRNGSubStream` advances to the next substream.

A direct R interface to the (clunkier) original C implementation is available in CRAN package **rlecuyer** (Sevcikova and Rossini, 2004–present). That works with named streams, each of which have three 6-element seeds associated with them. This can easily be emulated in R by storing `.Random.seed` at suitable times. There is another interface using S4 classes in package **rstream** (Leydold, 2005–present).

7 Load balancing

The introduction mentioned a different strategy which dynamically allocates tasks to workers: this is sometimes known as ‘load balancing’ and is implemented in `mclapply(mc.preschedule = FALSE)`, `clusterApplyLB` and wrappers such as `parLapplyLB` and `clusterMap(.scheduling = "dynamic")`.

Load balancing is potentially advantageous when the tasks take quite dissimilar amounts of computation time, or where the nodes are of disparate capabilities. But some *caveats* are in order:

- (a) Random number streams are allocated to nodes, so if the tasks involve random numbers they are likely to be non-repeatable (as the allocation of tasks to nodes depends on the workloads of the nodes). It would however take only slightly more work to allocate a stream to each task.
- (b) More care is needed in allocating the tasks. If 1000 tasks need to be allocated to 10 nodes, the standard approach send chunks of 100 tasks to each of the nodes. The load-balancing approach sends tasks one at a time to a node, and the communication overhead may be high. So it makes sense to have substantially more tasks than nodes, but not by a factor of 100 (and maybe not by 10).

8 Portability considerations

People wanting to provide parallel facilities in their code need to decide how hard they want to try to be portable and efficient: no approach works optimally on all platforms.

Using `mclapply` is usually the simplest approach, but will run serial versions of the code on Windows. This may suffice where parallel computation is only required for use on a single multi-core Unix-alike server—for `mclapply` can only run on a single shared-memory system. There is fallback to serial use when needed, by setting `mc.cores = 1`.

Using `parLapply` will work everywhere that socket communication works, and can be used, for example, to harness all the CPU cores in a lab of machines that are not otherwise in use. But socket communication may be blocked even when using a single machine and is quite likely to be blocked between machines in a lab. There is not currently any fallback to serial use, nor could there easily be (as the workers start with a different R environment from the one current on the master).

An example of providing access to both approaches as well as serial code is package **boot**, version 1.3-3 or later.

9 Extended examples

```
> library(parallel)
```

Probably the most common use of coarse-grained parallelization in statistics is to do multiple simulation runs, for example to do large numbers of bootstrap replicates or several runs of an MCMC simulation. We show an example of each.

Note that some of the examples will only work serially on Windows and some actually are computer-intensive.

9.1 Bootstrapping

Package **boot** (Canty and Ripley, 1999–present) is support software for the monograph by Davison and Hinkley (1997). Bootstrapping is often used as an example of easy parallelization, and some methods of producing confidence intervals require many thousands of bootstrap samples. As from version 1.3-1 the package itself has parallel support within its main functions, but we illustrate how to use the original (serial) functions in parallel computations.

We consider two examples using the `cd4` dataset from package **boot** where the interest is in the correlation between before and after measurements. The first is a straight simulation, often called a *parametric bootstrap*. The non-parallel form is

```
> library(boot)
> cd4.rg <- function(data, mle) MASS::mvrnorm(nrow(data), mle$m, mle$v)
> cd4.mle <- list(m = colMeans(cd4), v = var(cd4))
> cd4.boot <- boot(cd4, corr, R = 999, sim = "parametric",
+               ran.gen = cd4.rg, mle = cd4.mle)
> boot.ci(cd4.boot, type = c("norm", "basic", "perc"),
+       conf = 0.9, h = atanh, hinv = tanh)
```

To do this with `mclapply` we need to break this into separate runs, and we will illustrate two runs of 500 simulations each:

```

> cd4.rg <- function(data, mle) MASS::mvrnorm(nrow(data), mle$m, mle$v)
> cd4.mle <- list(m = colMeans(cd4), v = var(cd4))
> run1 <- function(...) boot(cd4, corr, R = 500, sim = "parametric",
+                             ran.gen = cd4.rg, mle = cd4.mle)
> mc <- 2 # set as appropriate for your hardware
> ## To make this reproducible:
> set.seed(123, "L'Ecuyer")
> cd4.boot <- do.call(c, mclapply(seq_len(mc), run1) )
> boot.ci(cd4.boot, type = c("norm", "basic", "perc"),
+         conf = 0.9, h = atanh, hinv = tanh)

```

There are many ways to program things like this: often the neatest is to encapsulate the computation in a function, so this is the parallel form of

```

> do.call(c, lapply(seq_len(mc), run1))

```

To run this with `parLapply` we could take a similar approach by

```

> run1 <- function(...) {
+   library(boot)
+   cd4.rg <- function(data, mle) MASS::mvrnorm(nrow(data), mle$m, mle$v)
+   cd4.mle <- list(m = colMeans(cd4), v = var(cd4))
+   boot(cd4, corr, R = 500, sim = "parametric",
+         ran.gen = cd4.rg, mle = cd4.mle)
+ }
> cl <- makeCluster(mc)
> ## make this reproducible
> clusterSetRNGStream(cl, 123)
> library(boot) # needed for c() method on master
> cd4.boot <- do.call(c, parLapply(cl, seq_len(mc), run1) )
> boot.ci(cd4.boot, type = c("norm", "basic", "perc"),
+         conf = 0.9, h = atanh, hinv = tanh)
> stopCluster(cl)

```

Note that whereas with `mclapply` all the packages and objects we use are automatically available on the workers, this is not in general⁷ the case with the `parLapply` approach. There is often a delicate choice of where to do the computations: for example we could compute `cd4.mle` on the workers (as above) or on the master and send the value to the workers. We illustrate the latter by the following code

```

> cl <- makeCluster(mc)
> cd4.rg <- function(data, mle) MASS::mvrnorm(nrow(data), mle$m, mle$v)
> cd4.mle <- list(m = colMeans(cd4), v = var(cd4))
> clusterExport(cl, c("cd4.rg", "cd4.mle"))
> junk <- clusterEvalQ(cl, library(boot)) # discard result
> clusterSetRNGStream(cl, 123)
> res <- clusterEvalQ(cl, boot(cd4, corr, R = 500,
+                             sim = "parametric", ran.gen = cd4.rg, mle = cd4.mle))
> library(boot) # needed for c() method on master
> cd4.boot <- do.call(c, res)
> boot.ci(cd4.boot, type = c("norm", "basic", "perc"),
+         conf = 0.9, h = atanh, hinv = tanh)
> stopCluster(cl)

```

⁷it is with clusters set up with `makeForkCluster`.

Running the double bootstrap on the same problem is far more computer-intensive. The standard version is

```
> R <- 999; M <- 999 ## we would like at least 999 each
> cd4.nest <- boot(cd4, nested.corr, R=R, stype="w", t0=corr(cd4), M=M)
> ## nested.corr is a function in package boot
> op <- par(pty = "s", xaxs = "i", yaxs = "i")
> qqplot((1:R)/(R+1), cd4.nest$t[, 2], pch = ".", asp = 1,
+         xlab = "nominal", ylab = "estimated")
> abline(a = 0, b = 1, col = "grey")
> abline(h = 0.05, col = "grey")
> abline(h = 0.95, col = "grey")
> par(op)
> nominal <- (1:R)/(R+1)
> actual <- cd4.nest$t[, 2]
> 100*nominal[c(sum(actual <= 0.05), sum(actual < 0.95))]
```

which took about 55 secs on one core of an 8-core Linux server.

Using `mclapply` we could use

```
> mc <- 9
> R <- 999; M <- 999; RR <- floor(R/mc)
> run2 <- function(...)
+   cd4.nest <- boot(cd4, nested.corr, R=RR, stype="w", t0=corr(cd4), M=M)
> cd4.nest <- do.call(c, mclapply(seq_len(mc), run2, mc.cores = mc) )
> nominal <- (1:R)/(R+1)
> actual <- cd4.nest$t[, 2]
> 100*nominal[c(sum(actual <= 0.05), sum(actual < 0.95))]
```

which ran in 11 secs (elapsed) using all of that server.

9.2 MCMC runs

Ripley (1988) discusses the maximum-likelihood estimation of the Strauss process, which is done by solving a moment equation

$$E_c T = t$$

where T is the number of R -close pairs and t is the observed value, 30 in the following example. A serial approach to the initial exploration might be

```
> library(spatial)
> towns <- ppinit("towns.dat")
> tget <- function(x, r=3.5) sum(dist(cbind(x$x, x$y)) < r)
> t0 <- tget(towns)
> R <- 1000
> c <- seq(0, 1, 0.1)
> ## res[1] = 0
> res <- c(0, sapply(c[-1], function(c)
+   mean(replicate(R, tget(Strauss(69, c=c, r=3.5))))))
> plot(c, res, type="l", ylab="E t")
> abline(h=t0, col="grey")
```

which takes about 20 seconds today, but many hours when first done in 1985. A parallel version might be

```

> run3 <- function(c) {
+   library(spatial)
+   towns <- ppinit("towns.dat") # has side effects
+   mean(replicate(R, tget(Strauss(69, c=c, r=3.5))))
+ }
> cl <- makeCluster(10, methods = FALSE)
> clusterExport(cl, c("R", "towns", "tget"))
> res <- c(0, parSapply(cl, c[-1], run3)) # 10 tasks
> stopCluster(cl)

```

which took about 4.5 secs, plus 2 secs to set up the cluster. Using a fork cluster (not on Windows) makes the startup much faster and setup easier:

```

> cl <- makeForkCluster(10) # fork after the variables have been set up
> run4 <- function(c) mean(replicate(R, tget(Strauss(69, c=c, r=3.5))))
> res <- c(0, parSapply(cl, c[-1], run4))
> stopCluster(cl)

```

As one might expect, the `mclapply` version is slightly simpler:

```

> run4 <- function(c) mean(replicate(R, tget(Strauss(69, c=c, r=3.5))))
> res <- c(0, unlist(mclapply(c[-1], run4, mc.cores = 10)))

```

If you do not have as many as 10 cores, you might want to consider load-balancing in a task like this as the time taken per simulation does vary with `c`. This can be done using `mclapply(mc.preschedule = FALSE)` or `parSapplyLB`. The disadvantage is that the results would not be reproducible (which does not matter here).

9.3 Package installation

With over 4000 R packages available, it is often helpful to do a comprehensive install in parallel. We provide facilities to do so in `install.packages` *via* a parallel `make`, but that approach may not be suitable.⁸ Some of the tasks take many times longer than others, and we do not know in advance which these are.

We illustrate an approach using package **parallel** which is used on part of the CRAN check farm. Suppose that there is a function `do_one(pkg)` which installs a single package and then returns. Then the task is to run `do_one` on as many of the `M` workers as possible whilst ensuring that all of the direct and indirect dependencies of `pkg` are installed before `pkg` itself. As the installation of a single package can block several others, we do need to allow the number of installs running simultaneously to vary: the following code achieves that, but needs to use low-level functions to do so.

```

> pkgs <- "<names of packages to be installed>"
> M <- 20 # number of parallel installs
> M <- min(M, length(pkgs))
> library(parallel)
> unlink("install_log")
> cl <- makeCluster(M, outfile = "install_log")
> clusterExport(cl, c("tars", "fakes", "gcc")) # variables needed by do_one
> ## set up available via a call to available.packages() for
> ## repositories containing all the packages involved and all their
> ## dependencies.

```

⁸A parallel `make` might not be available, and we have seen a couple of instances of package installation not working correctly when run from `make`.

```

> DL <- utils:::make_dependency_list(pkgs, available, recursive = TRUE)
> DL <- lapply(DL, function(x) x[x %in% pkgs])
> lens <- sapply(DL, length)
> ready <- names(DL[lens == 0L])
> done <- character() # packages already installed
> n <- length(ready)
> submit <- function(node, pkg)
+   parallel::sendCall(cl[[node]], do_one, list(pkg), tag = pkg)
> for (i in 1:min(n, M)) submit(i, ready[i])
> DL <- DL[!names(DL) %in% ready[1:min(n, M)]]
> av <- if(n < M) (n+1L):M else integer() # available workers
> while(length(done) < length(pkgs)) {
+   d <- parallel::recvOneResult(cl)
+   av <- c(av, d$node)
+   done <- c(done, d$tag)
+   OK <- unlist(lapply(DL, function(x) all(x %in% done) ))
+   if (!any(OK)) next
+   p <- names(DL)[OK]
+   m <- min(length(p), length(av)) # >= 1
+   for (i in 1:m) submit(av[i], p[i])
+   av <- av[-(1:m)]
+   DL <- DL[!names(DL) %in% p[1:m]]
+ }

```

9.4 Passing ...

The semantics of ... do not fit well with parallel operation, since lazy evaluation may be delayed until the tasks have been sent to the workers. This is no problem in the forking approach, as the information needed for lazy evaluation will be present in the forked workers.

For **snow**-like clusters the trick is to ensure that ... has any promises forced whilst the information is still available. This is how **boot** does it:

```

>   fn <- function(r) statistic(data, i[r, ], ...)
>   RR <- sum(R)
>   res <- if (ncpus > 1L && (have_mc || have_snow)) {
+     if (have_mc) {
+       parallel::mclapply(seq_len(RR), fn, mc.cores = ncpus)
+     } else if (have_snow) {
+       list(...) # evaluate any promises
+       if (is.null(cl)) {
+         cl <- parallel::makePSOCKcluster(rep("localhost", ncpus))
+         if(RNGkind()[1L] == "L'Ecuyer-CMRG")
+           parallel::clusterSetRNGStream(cl)
+         res <- parallel::parLapply(cl, seq_len(RR), fn)
+         parallel::stopCluster(cl)
+         res
+       } else parallel::parLapply(cl, seq_len(RR), fn)
+     }
+   } else lapply(seq_len(RR), fn)

```

Note that ... is an argument to boot, and so after

```

>   list(...) # evaluate any promises

```

it refers to objects within the evaluation frame of `boot` and hence the environment of `fn` which will therefore be sent to the workers along with `fn`.

10 Differences from earlier versions

The support of parallel RNG differs from `snow`: `multicore` had no support.

10.1 Differences from `multicore`

`multicore` had quite elaborate code to inhibit the Aqua event loop for `R.app` and the event loop for the `quartz` graphics device. This has been replaced by recording a flag in the R executable for a child process.

The version of `detectCores` here is biased towards the number of physical CPUs. This was occasioned by serious problems on Sparc Solaris where `multicore` defaulted to a silly number of processes.

Functions `fork` and `kill` have `mc` prefixes and are not exported. This avoids confusion with other packages (such as package `fork`), and note that `mckill` is not as general as `tools::pskill`.

Aliased functions `collect` and `parallel` are no longer provided.

10.2 Differences from `snow`

`snow` set a timeout that exceeded the maximum value required by POSIX, and did not provide a means to set the timeout on the workers. This resulted in process interlocks on Solaris.

`makeCluster` creates MPI or NWS clusters by calling `snow`.

`makePSOCKcluster` has been streamlined, as package `parallel` is in a known location on all systems, and `Rscript` is these days always available. Logging of workers is set up to append to the file, so multiple processes can be logged.

`parSapply` has been brought into line with `sapply`.

`clusterMap()` gains `SIMPLIFY` and `USE.NAMES` arguments to make it a parallel version of `mapply` and `Map`.


The timing interface has not been copied.

References

- Canty A, Ripley BD (1999–present). “boot: Bootstrap Functions.” URL <http://cran.r-project.org/package=boot>.
- Davison AC, Hinkley DV (1997). *Bootstrap Methods and Their Application*. Cambridge University Press, Cambridge.
- L’Ecuyer P (1999). “Good parameters and implementations for combined multiple recursive random number generators.” *Operations Research*, **47**, 195–164. URL <http://www.imo.umontreal.ca/~lecuyer/myftp/papers/combmrng2.ps>.

- L'Ecuyer P, Simard R, Chen EJ, Kelton WD (2002). "An object-oriented random-number package with many long streams and substreams." *Operations Research*, **50**, 1073–5. URL <http://www.iro.umontreal.ca/~lecuyer/myftp/papers/streams00.pdf>.
- Leydold J (2005–present). "rstream: Streams of random numbers." URL <http://cran.r-project.org/package=rstream>.
- Ripley BD (1988). *Statistical Inference for Spatial Processes*. Cambridge University Press, Cambridge.
- Sevcikova H, Rossini T (2004–present). "rlecuyer: R interface to RNG with multiple streams." URL <http://cran.r-project.org/package=rlecuyer>.
- Tierney L, Rossini AJ, Li N, Sevcikova H (2003–present). "Simple Network of WorkStations for R." URL <http://www.stat.uiowa.edu/~luke/R/cluster/cluster.html>.
- Urbanek S (2009–present). "multicore: Parallel processing of R code on machines with multiple cores or CPUs." URL <http://cran.r-project.org/package=multicore>.

Octave

GNU Octave is a high-level interpreted language, primarily intended for numerical computations. It provides capabilities for the numerical solution of linear and nonlinear problems, and for performing other numerical experiments. It also provides extensive graphics capabilities for data visualization and manipulation. Octave is normally used through its interactive command line interface, but it can also be used to write non-interactive programs. The Octave language is quite similar to Matlab so that most programs are easily portable. Read more on <http://www.gnu.org/software/octave/> 

Two versions of octave are available on the cluster, via module

Status	Version
Stable	Octave 3.8.2

```
$ module load Octave
```

The octave on the cluster is linked to highly optimized MKL mathematical library. This provides threaded parallelization to many octave kernels, notably the linear algebra subroutines. Octave runs these heavy calculation kernels without any penalty. By default, octave would parallelize to 24 threads. You may control the threads by setting the OMP_NUM_THREADS environment variable.

To run octave interactively, log in with ssh -X parameter for X11 forwarding. Run octave:

```
$ octave
```

To run octave in batch mode, write an octave script, then write a bash jobscript and execute via the qsub command. By default, octave will use 16 threads when running MKL kernels.

```
#!/bin/bash

# change to local scratch directory
mkdir -p /scratch/work/user/$USER/$PBS_JOBID
cd /scratch/work/user/$USER/$PBS_JOBID || exit

# copy input file to scratch
cp $PBS_O_WORKDIR/octcode.m .

# load octave module
module load Octave

# execute the calculation
octave -q --eval octcode > output.out

# copy output file to home
cp output.out $PBS_O_WORKDIR/.

#exit
exit
```

This script may be submitted directly to the PBS workload manager via the qsub command. The inputs are in octcode.m file, outputs in output.out file. See the single node jobscript example in the Job execution section.

The octave c compiler mkcofile calls the GNU gcc 4.8.1 for compiling native c code. This is very useful for running native c subroutines in octave environment.

```
$ mkoctfile -v
```

Octave may use MPI for interprocess communication. This functionality is currently not supported on the cluster. In case you require the octave interface to MPI, please contact our cluster support [✉](#).

R

Introduction

The R is a language and environment for statistical computing and graphics. R provides a wide variety of statistical (linear and nonlinear modelling, classical statistical tests, time-series analysis, classification, clustering, ...) and graphical techniques, and is highly extensible.

One of R's strengths is the ease with which well-designed publication-quality plots can be produced, including mathematical symbols and formulae where needed. Great care has been taken over the defaults for the minor design choices in graphics, but the user retains full control.

Another convenience is the ease with which the C code or third party libraries may be integrated within R.

Extensive support for parallel computing is available within R.

Read more on <http://www.r-project.org/>, <http://cran.r-project.org/doc/manuals/r-release/R-lang.html>

Modules

The R version 3.1.1 is available on the cluster, along with GUI interface Rstudio

Application	Version
R	R 3.1.1
Rstudio	Rstudio 0.97

```
$ module load R
```

Execution

The R on Anselm is linked to highly optimized MKL mathematical library. This provides threaded parallelization to many R kernels, notably the linear algebra subroutines. The R runs these heavy calculation kernels without any penalty. By default, the R would parallelize to 24 threads. You may control the threads by setting the OMP_NUM_THREADS environment variable.

Interactive execution

To run R interactively, using Rstudio GUI, log in with ssh -X parameter for X11 forwarding. Run rstudio:

```
$ module load Rstudio
$ rstudio
```

Batch execution

To run R in batch mode, write an R script, then write a bash jobscript and execute via the qsub command. By default, R will use 24 threads when running MKL kernels.

Example jobscript:


```
#!/bin/bash

# change to local scratch directory
cd /lscratch/$PBS_JOBID || exit

# copy input file to scratch
cp $PBS_O_WORKDIR/rscript.R .

# load R module
module load R

# execute the calculation
R CMD BATCH rscript.R routput.out

# copy output file to home
cp routput.out $PBS_O_WORKDIR/.

#exit
exit
```

This script may be submitted directly to the PBS workload manager via the qsub command. The inputs are in rscript.R file, outputs in routput.out file. See the single node jobscript example in the Job execution section.

Parallel R

Parallel execution of R may be achieved in many ways. One approach is the implied parallelization due to linked libraries or specially enabled functions, as described above. In the following sections, we focus on explicit parallelization, where parallel constructs are directly stated within the R script.

Package parallel

The package parallel provides support for parallel computation, including by forking (taken from package multicore), by sockets (taken from package snow) and random-number generation.

The package is activated this way:

```
$ R
> library(parallel)
```

More information and examples may be obtained directly by reading the documentation available in R

```
> ?parallel
> library(help = "parallel")
> vignette("parallel")
```

Download the package parallel vignette.

The forking is the most simple to use. Forking family of functions provide parallelized, drop in replacement for the serial apply() family of functions.

!!! Note “Note” Forking via package parallel provides functionality similar to OpenMP construct omp parallel for

Only cores of single node can be utilized this way!

Forking example:

```
library(parallel)

#integrand function
f <- function(i,h) {
  x <- h*(i-0.5)
  return (4/(1 + x*x))
}

#initialize
size <- detectCores()

while (TRUE)
{
  #read number of intervals
  cat("Enter the number of intervals: (0 quits) ")
  fp<-file("stdin"); n<-scan(fp,nmax=1); close(fp)

  if(n<=0) break

  #run the calculation
  n <- max(n,size)
  h <- 1.0/n

  i <- seq(1,n);
  pi3 <- h*sum(simplify2array(mclapply(i,f,h,mc.cores=size)));

  #print results
  cat(sprintf("Value of PI %16.14f, diff= %16.14fn",pi3,pi3-pi))
}
```

The above example is the classic parallel example for calculating the number π . Note the **detectCores()** and **mclapply()** functions. Execute the example as:

```
$ R --slave --no-save --no-restore -f pi3p.R
```

Every evaluation of the integrand function runs in parallel on different process.

Package Rmpi

package Rmpi provides an interface (wrapper) to MPI APIs.

It also provides interactive R slave environment. On the cluster, Rmpi provides interface to the OpenMPI.

Read more on Rmpi at <http://cran.r-project.org/web/packages/Rmpi/>, reference manual is available at <http://cran.r-project.org/web/packages/Rmpi/Rmpi.pdf>

When using package Rmpi, both openmpi and R modules must be loaded

```
$ module load OpenMPI
$ module load R
```

Rmpi may be used in three basic ways. The static approach is identical to executing any other MPI program. In addition, there is Rslaves dynamic MPI approach and the mpi.apply approach. In the following section, we will use the number integration example, to illustrate all these concepts.

static Rmpi

Static Rmpi programs are executed via mpiexec, as any other MPI programs. Number of processes is static - given at the launch time.

Static Rmpi example:

```
library(Rmpi)

#integrand function
f <- function(i,h) {
  x <- h*(i-0.5)
  return (4/(1 + x*x))
}

#initialize
invisible(mpi.comm.dup(0,1))
rank <- mpi.comm.rank()
size <- mpi.comm.size()
n<-0

while (TRUE)
{
  #read number of intervals
  if (rank==0) {
    cat("Enter the number of intervals: (0 quits) ")
    fp<-file("stdin"); n<-scan(fp,nmax=1); close(fp)
  }

  #broadcast the intervals
  n <- mpi.bcast(as.integer(n),type=1)

  if(n<=0) break

  #run the calculation
  n <- max(n,size)
  h <- 1.0/n

  i <- seq(rank+1,n,size);
  mypi <- h*sum(sapply(i,f,h));

  pi3 <- mpi.reduce(mypi)

  #print results
  if (rank==0) cat(sprintf("Value of PI %16.14f, diff= %16.14fn",pi3,pi3-pi))
}

mpi.quit()
```

The above is the static MPI example for calculating the number π . Note the **library(Rmpi)** and **mpi.comm.dup()** function calls. Execute the example as:

```
$ mpirun R --slave --no-save --no-restore -f pi3.R
```

dynamic Rmpi

Dynamic Rmpi programs are executed by calling the R directly. OpenMPI module must be still loaded. The R slave processes will be spawned by a function call within the Rmpi program.

Dynamic Rmpi example:

```
#integrand function
f <- function(i,h) {
  x <- h*(i-0.5)
  return (4/(1 + x*x))
}

#the worker function
workerpi <- function()
{
  #initialize
  rank <- mpi.comm.rank()
  size <- mpi.comm.size()
  n<-0

  while (TRUE)
  {
    #read number of intervals
    if (rank==0) {
      cat("Enter the number of intervals: (0 quits) ")
      fp<-file("stdin"); n<-scan(fp,nmax=1); close(fp)
    }

    #broadcast the intervals
    n <- mpi.bcast(as.integer(n),type=1)

    if(n<=0) break

    #run the calculation
    n <- max(n,size)
    h <- 1.0/n

    i <- seq(rank+1,n,size);
    mypi <- h*sum(sapply(i,f,h));

    pi3 <- mpi.reduce(mypi)

    #print results
    if (rank==0) cat(sprintf("Value of PI %16.14f, diff= %16.14fn",pi3,pi3-pi))
  }
}
```

```

#main
library(Rmpi)

cat("Enter the number of slaves: ")
fp<-file("stdin"); ns<-scan(fp,nmax=1); close(fp)

mpi.spawn.Rslaves(nslaves=ns)
mpi.bcast.Robj2slave(f)
mpi.bcast.Robj2slave(workerpi)

mpi.bcast.cmd(workerpi())
workerpi()

mpi.quit()

```

The above example is the dynamic MPI example for calculating the number π . Both master and slave processes carry out the calculation. Note the `mpi.spawn.Rslaves()`, `mpi.bcast.Robj2slave()`** and the `mpi.bcast.cmd()`** function calls.

Execute the example as:

```
$ mpirun -np 1 R --slave --no-save --no-restore -f pi3Rslaves.R
```

Note that this method uses `MPI_Comm_spawn` (Dynamic process feature of MPI-2) to start the slave processes - the master process needs to be launched with MPI. In general, Dynamic processes are not well supported among MPI implementations, some issues might arise. Also, environment variables are not propagated to spawned processes, so they will not see paths from modules.

mpi.apply Rmpi

`mpi.apply` is a specific way of executing Dynamic Rmpi programs.

`mpi.apply()` family of functions provide MPI parallelized, drop in replacement for the serial `apply()` family of functions.

Execution is identical to other dynamic Rmpi programs.

`mpi.apply Rmpi` example:

```

#integrand function
f <- function(i,h) {
  x <- h*(i-0.5)
  return (4/(1 + x*x))
}

#the worker function
workerpi <- function(rank,size,n)
{
  #run the calculation
  n <- max(n,size)
  h <- 1.0/n

  i <- seq(rank,n,size);
  mypi <- h*sum(sapply(i,f,h));

  return(mypi)
}

```

```

}

#main
library(Rmpi)

cat("Enter the number of slaves: ")
fp<-file("stdin"); ns<-scan(fp,nmax=1); close(fp)

mpi.spawn.Rslaves(nslaves=ns)
mpi.bcast.Robj2slave(f)
mpi.bcast.Robj2slave(workerpi)

while (TRUE)
{
  #read number of intervals
  cat("Enter the number of intervals: (0 quits) ")
  fp<-file("stdin"); n<-scan(fp,nmax=1); close(fp)
  if(n<=0) break

  #run workerpi
  i=seq(1,2*ns)
  pi3=sum(mpi.parSapply(i,workerpi,2*ns,n))

  #print results
  cat(sprintf("Value of PI %16.14f, diff= %16.14fn",pi3,pi3-pi))
}

mpi.quit()

```

The above is the mpi.apply MPI example for calculating the number π . Only the slave processes carry out the calculation. Note the **mpi.parSapply()**, function call. The package parallel example above may be trivially adapted (for much better performance) to this structure using the **mclapply()** in place of **mpi.parSapply()**.

Execute the example as:

```
$ mpirun -np 1 R --slave --no-save --no-restore -f pi3parSapply.R
```

Combining parallel and Rmpi

Currently, the two packages can not be combined for hybrid calculations.

Parallel execution

The R parallel jobs are executed via the PBS queue system exactly as any other parallel jobs. User must create an appropriate jobscript and submit via the **qsub**

Example jobscript for static Rmpi parallel R execution, running 1 process per core:

```

#!/bin/bash
#PBS -q qprod
#PBS -N Rjob
#PBS -l select=100:ncpus=24:mpiprocs=24:ompthreads=1

```

```

# change to scratch directory
SCRDIR=/scratch/work/user/$USER/myjob
cd $SCRDIR || exit

# copy input file to scratch
cp $PBS_O_WORKDIR/rscript.R .

# load R and openmpi module
module load R
module load OpenMPI

# execute the calculation
mpiexec -bycore -bind-to-core R --slave --no-save --no-restore -f rscript.R

# copy output file to home
cp routput.out $PBS_O_WORKDIR/.

#exit
exit

```

For more information about jobscripts and MPI execution refer to the Job submission and general MPI sections.

Xeon Phi Offload

By leveraging MKL, R can accelerate certain computations, most notably linear algebra operations on the Xeon Phi accelerator by using Automated Offload. To use MKL Automated Offload, you need to first set this environment variable before R execution:

```
$ export MKL_MIC_ENABLE=1
```

Read more about automatic offload

Numerical languages

Interpreted languages for numerical computations and analysis

Introduction

This section contains a collection of high-level interpreted languages, primarily intended for numerical computations.

Matlab

MATLAB[®] is a high-level language and interactive environment for numerical computation, visualization, and programming.

```
$ module load MATLAB  
$ matlab
```

Read more at the Matlab page.

Octave

GNU Octave is a high-level interpreted language, primarily intended for numerical computations. The Octave language is quite similar to Matlab so that most programs are easily portable.

```
$ module load Octave  
$ octave
```

Read more at the Octave page.

R

The R is an interpreted language and environment for statistical computing and graphics.

```
$ module load R  
$ R
```

Read more at the R page.

Matlab

Introduction

Matlab is available in versions R2015a and R2015b. There are always two variants of the release:

- Non commercial or so called EDU variant, which can be used for common research and educational purposes.
- Commercial or so called COM variant, which can be used also for commercial activities. The licenses for commercial variant are much more expensive, so usually the commercial variant has only subset of features compared to the EDU available.

To load the latest version of Matlab load the module

```
$ module load MATLAB
```

By default the EDU variant is marked as default. If you need other version or variant, load the particular version. To obtain the list of available versions use

```
$ module avail MATLAB
```

If you need to use the Matlab GUI to prepare your Matlab programs, you can use Matlab directly on the login nodes. But for all computations use Matlab on the compute nodes via PBS Pro scheduler.

If you require the Matlab GUI, please follow the general informations about running graphical applications.

Matlab GUI is quite slow using the X forwarding built in the PBS (qsub -X), so using X11 display redirection either via SSH or directly by xauth (please see the “GUI Applications on Compute Nodes over VNC” part here) is recommended.

To run Matlab with GUI, use

```
$ matlab
```

To run Matlab in text mode, without the Matlab Desktop GUI environment, use

```
$ matlab -nodesktop -nosplash
```

plots, images, etc... will be still available.

Running parallel Matlab using Distributed Computing Toolbox / Engine

Distributed toolbox is available only for the EDU variant

The MPIEXEC mode available in previous versions is no longer available in MATLAB 2015. Also, the programming interface has changed. Refer to Release Notes [here](#).

Delete previously used file mpiLibConf.m, we have observed crashes when using Intel MPI.

To use Distributed Computing, you first need to setup a parallel profile. We have provided the profile for you, you can either import it in MATLAB command line:

```
> parallel.importProfile('/apps/all/MATLAB/2015b-EDU/SalomonPBSPro.settings')  
  
ans =  
  
SalomonPBSPro
```

Or in the GUI, go to tab HOME -> Parallel -> Manage Cluster Profiles..., click Import and navigate to :

```
/apps/all/MATLAB/2015b-EDU/SalomonPBSPro.settings
```

With the new mode, MATLAB itself launches the workers via PBS, so you can either use interactive mode or a batch mode on one node, but the actual parallel processing will be done in a separate job started by MATLAB itself. Alternatively, you can use “local” mode to run parallel code on just a single node.

Parallel Matlab interactive session

Following example shows how to start interactive session with support for Matlab GUI. For more information about GUI based applications on Anselm see this page.

```
$ xhost +
$ qsub -I -v DISPLAY=$(uname -n):$(echo $DISPLAY | cut -d ':' -f 2) -A NONE-0-0 -q qe
-l feature__matlab__MATLAB=1
```

This qsub command example shows how to run Matlab on a single node.

The second part of the command shows how to request all necessary licenses. In this case 1 Matlab-EDU license and 48 Distributed Computing Engines licenses.

Once the access to compute nodes is granted by PBS, user can load following modules and start Matlab:

```
r1i0n17$ module load MATLAB/2015a-EDU
r1i0n17$ matlab &
```

Parallel Matlab batch job in Local mode

To run matlab in batch mode, write an matlab script, then write a bash jobscript and execute via the qsub command. By default, matlab will execute one matlab worker instance per allocated core.

```
#!/bin/bash
#PBS -A PROJECT ID
#PBS -q qprod
#PBS -l select=1:ncpus=24:mpiprocs=24:ompthreads=1

# change to shared scratch directory
SCR=/scratch/work/user/$USER/$PBS_JOBID
mkdir -p $SCR ; cd $SCR || exit

# copy input file to scratch
cp $PBS_O_WORKDIR/matlabcode.m .

# load modules
module load MATLAB/2015a-EDU

# execute the calculation
matlab -nodisplay -r matlabcode > output.out

# copy output file to home
cp output.out $PBS_O_WORKDIR/.
```

This script may be submitted directly to the PBS workload manager via the qsub command. The inputs and matlab script are in matlabcode.m file, outputs in output.out file. Note the missing .m extension in the matlab -r matlabcodefile call, **the .m must not be included**. Note that the **shared /scratch must be used**. Further, it is **important to include quit** statement at the end of the matlabcode.m script.

Submit the jobscript using qsub

```
$ qsub ./jobscript
```

Parallel Matlab Local mode program example

The last part of the configuration is done directly in the user Matlab script before Distributed Computing Toolbox is started.

```
cluster = parcluster('local')
```

This script creates scheduler object “cluster” of type “local” that starts workers locally.

Please note: Every Matlab script that needs to initialize/use matlabpool has to contain these three lines prior to calling parpool(sched, ...) function.

The last step is to start matlabpool with “cluster” object and correct number of workers. We have 24 cores per node, so we start 24 workers.

```
parpool(cluster,24);
```

```
... parallel code ...
```

```
parpool close
```

The complete example showing how to use Distributed Computing Toolbox in local mode is shown here.

```
cluster = parcluster('local');
cluster

parpool(cluster,24);

n=2000;

W = rand(n,n);
W = distributed(W);
x = (1:n)';
x = distributed(x);
spmd
[~, name] = system('hostname')

    T = W*x; % Calculation performed on labs, in parallel.
           % T and W are both codistributed arrays here.
end
T;
whos           % T and W are both distributed arrays here.
```

```
parpool close
quit
```

You can copy and paste the example in a .m file and execute. Note that the parpool size should correspond to **total number of cores** available on allocated nodes.

Parallel Matlab Batch job using PBS mode (workers spawned in a separate job)

This mode uses PBS scheduler to launch the parallel pool. It uses the SalomonPBSPro profile that needs to be imported to Cluster Manager, as mentioned before. This method uses MATLAB's PBS Scheduler interface - it spawns the workers in a separate job submitted by MATLAB using qsub.

This is an example of m-script using PBS mode:

```
cluster = parcluster('SalomonPBSPro');
set(cluster, 'SubmitArguments', '-A OPEN-0-0');
set(cluster, 'ResourceTemplate', '-q qprod -l select=10:ncpus=24');
set(cluster, 'NumWorkers', 240);

pool = parpool(cluster, 240);

n=2000;

W = rand(n,n);
W = distributed(W);
x = (1:n)';
x = distributed(x);
spmd
[~, name] = system('hostname')

    T = W*x; % Calculation performed on labs, in parallel.
           % T and W are both codistributed arrays here.
end
whos      % T and W are both distributed arrays here.

% shut down parallel pool
delete(pool)
```

Note that we first construct a cluster object using the imported profile, then set some important options, namely : SubmitArguments, where you need to specify accounting id, and ResourceTemplate, where you need to specify number of nodes to run the job.

You can start this script using batch mode the same way as in Local mode example.

Parallel Matlab Batch with direct launch (workers spawned within the existing job)

This method is a “hack” invented by us to emulate the mpiexec functionality found in previous MATLAB versions. We leverage the MATLAB Generic Scheduler interface, but instead of submitting the workers to PBS, we launch the workers directly within the running job, thus we avoid the issues with master script and workers running in separate jobs (issues with license not available, waiting for the worker's job to spawn etc.)

Please note that this method is experimental.

For this method, you need to use SalomonDirect profile, import it using the same way as SalomonPB-SPro 

This is an example of m-script using direct mode:

```
parallel.importProfile('/apps/all/MATLAB/2015b-EDU/SalomonDirect.settings')
cluster = parcluster('SalomonDirect');
set(cluster, 'NumWorkers', 48);

pool = parpool(cluster, 48);

n=2000;

W = rand(n,n);
W = distributed(W);
x = (1:n)';
x = distributed(x);
spmd
[~, name] = system('hostname')

    T = W*x; % Calculation performed on labs, in parallel.
        % T and W are both codistributed arrays here.
end
whos      % T and W are both distributed arrays here.

% shut down parallel pool
delete(pool)
```

Non-interactive Session and Licenses

If you want to run batch jobs with Matlab, be sure to request appropriate license features with the PBS Pro scheduler, at least the "-l ___feature___matlab___MATLAB=1" for EDU variant of Matlab. More information about how to check the license features states and how to request them with PBS Pro, please look here.

The licensing feature of PBS is currently disabled.

In case of non-interactive session please read the following information on how to modify the qsub command to test for available licenses prior getting the resource allocation.

Matlab Distributed Computing Engines start up time

Starting Matlab workers is an expensive process that requires certain amount of time. For your information please see the following table:

compute nodes	number of workers	start-up time[s]
16	384	831
8	192	807
4	96	483
2	48	16

MATLAB on UV2000

UV2000 machine available in queue “qfat” can be used for MATLAB computations. This is a SMP NUMA machine with large amount of RAM, which can be beneficial for certain types of MATLAB jobs. CPU cores are allocated in chunks of 8 for this machine.

You can use MATLAB on UV2000 in two parallel modes:

Threaded mode

Since this is a SMP machine, you can completely avoid using Parallel Toolbox and use only MATLAB's threading. MATLAB will automatically detect the number of cores you have allocated and will set `maxNumCompThreads` accordingly and certain operations, such as `fft`, `eig`, `svd`, etc. will be automatically run in threads. The advantage of this mode is that you don't need to modify your existing sequential codes.

Local cluster mode

You can also use Parallel Toolbox on UV2000. Use local cluster mode, “SalomonPBSPPro” profile will not work.

Setting license preferences

Some ANSYS tools allow you to explicitly specify usage of academic or commercial licenses in the command line (eg. `ansys161 -p aa_r` to select Academic Research license). However, we have observed that not all tools obey this option and choose commercial license.

Thus you need to configure preferred license order with ANSLIC_ADMIN. Please follow these steps and move Academic Research license to the top or bottom of the list accordingly.

Launch the ANSLIC_ADMIN utility in a graphical environment:

```
$ANSYSLIC_DIR/lic_admin/anslic_admin
```

ANSLIC_ADMIN Utility will be run

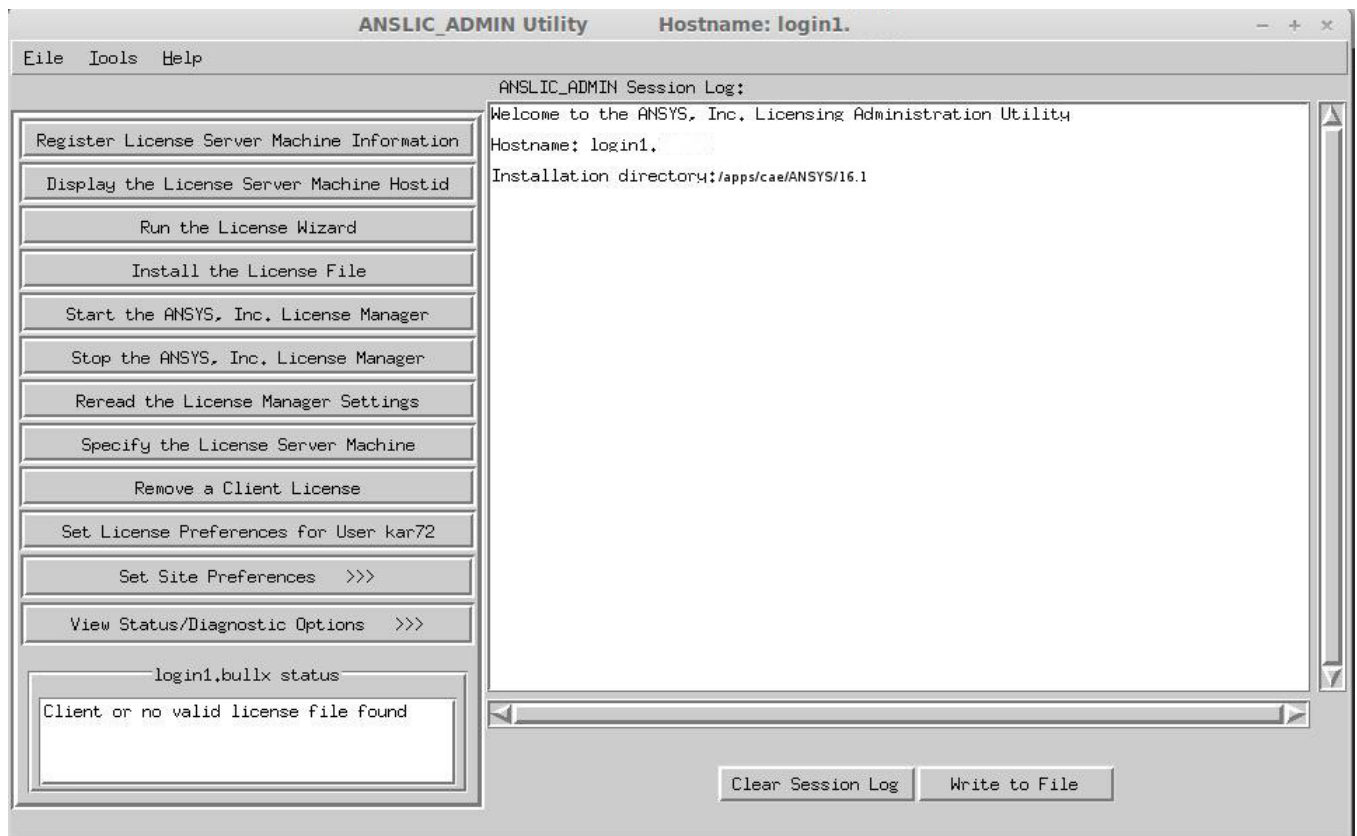


Figure 1:

ANSYS Academic Research license should be moved up to the top or down to the bottom of the list.

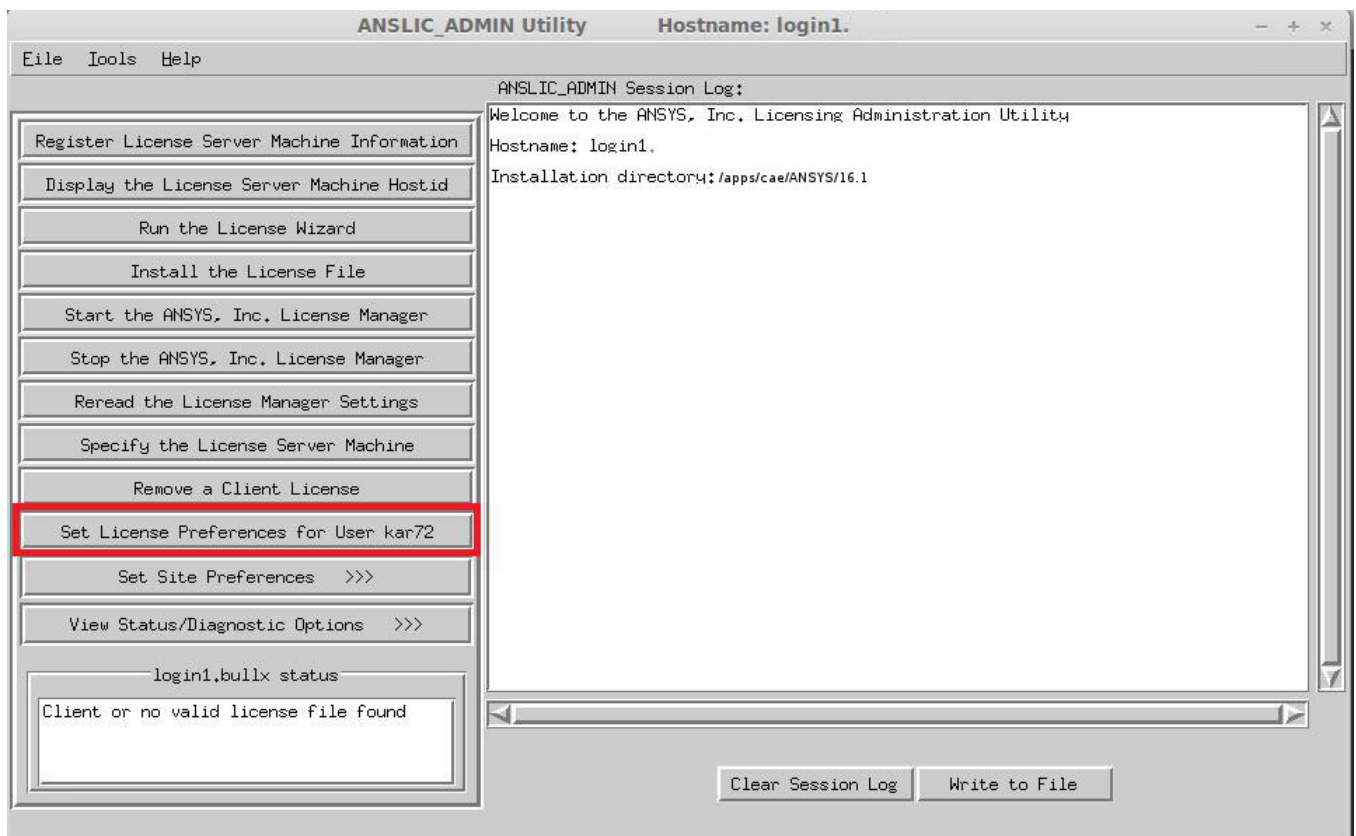


Figure 2:

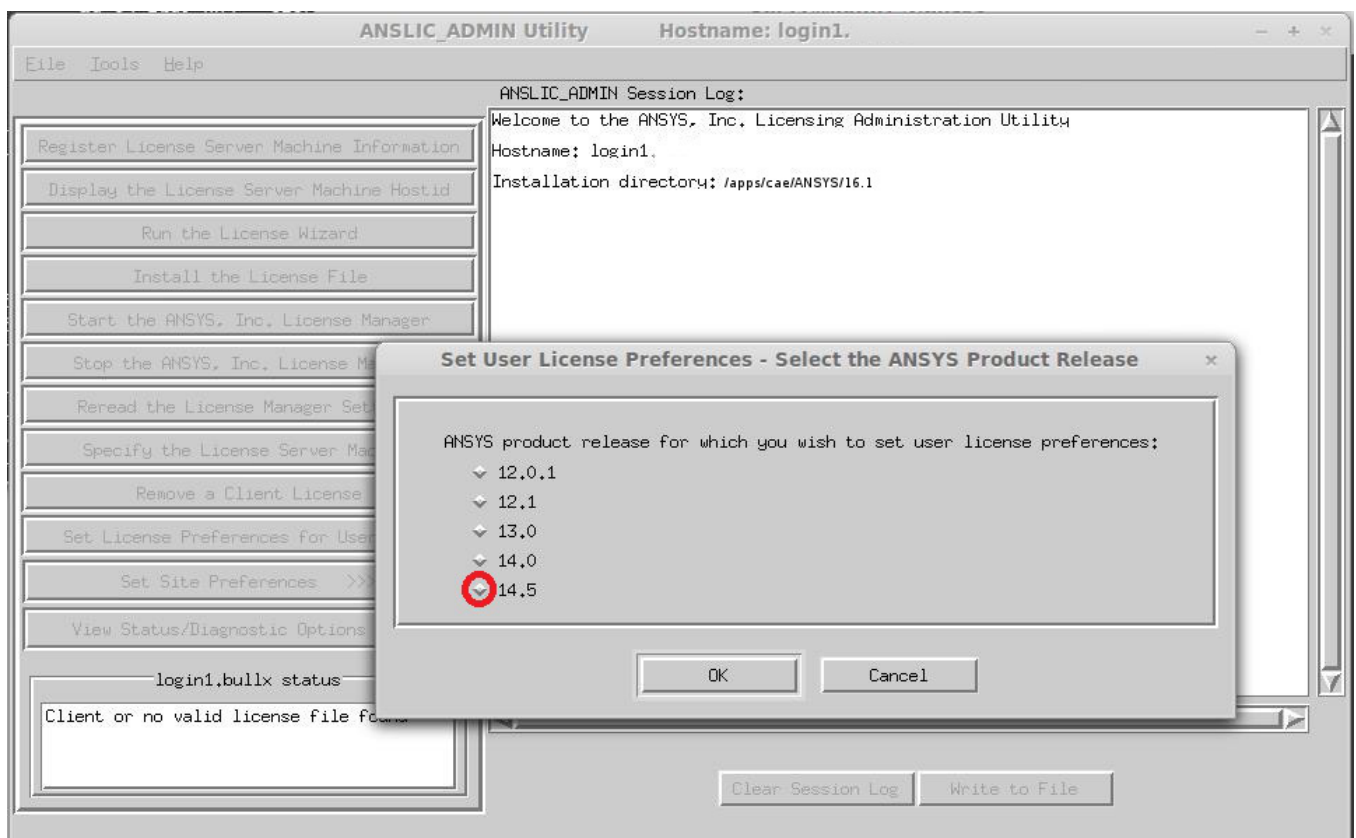


Figure 3:

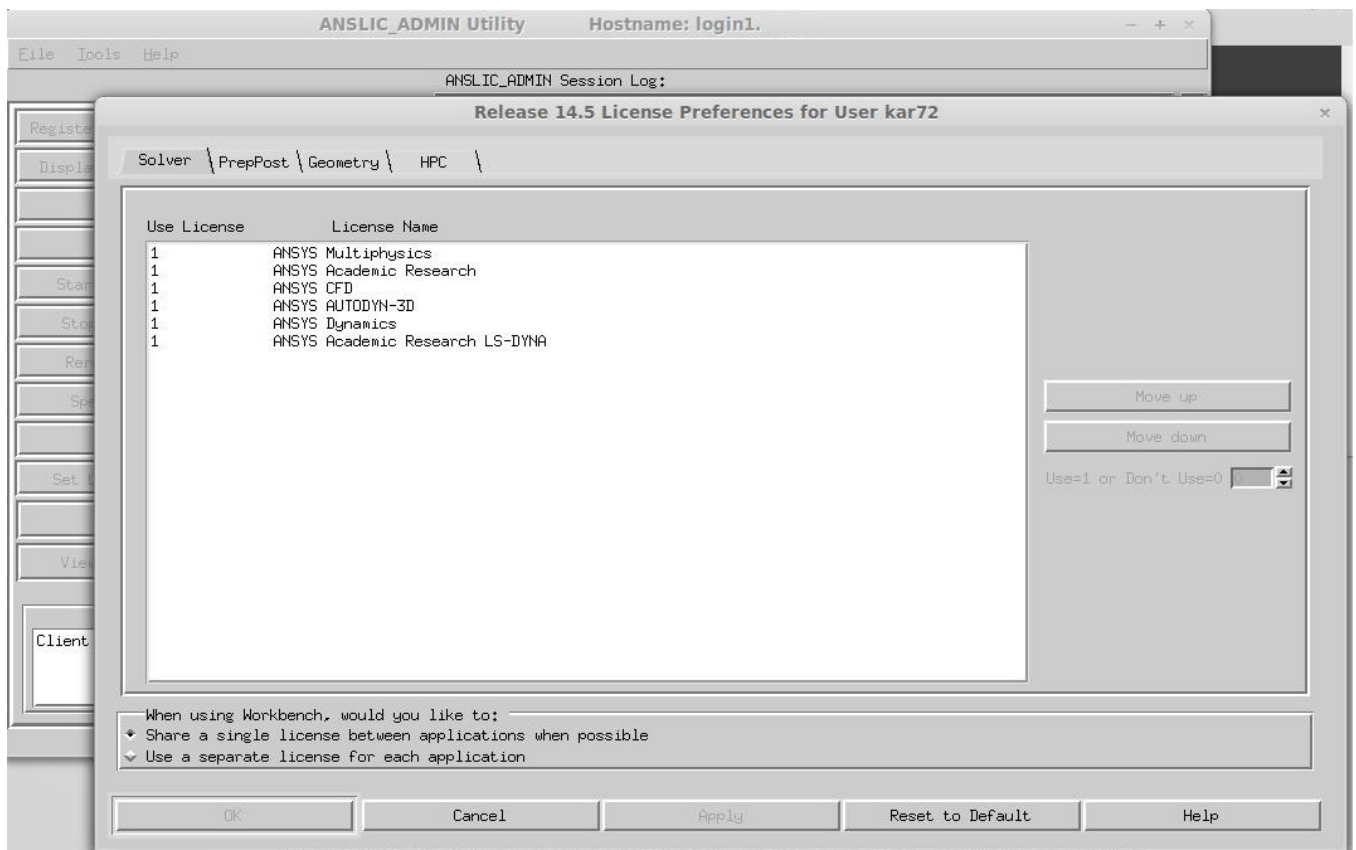


Figure 4:

Licensing and Available Versions

ANSYS licence can be used by:

- all persons in the carrying out of the CE IT4Innovations Project (In addition to the primary licensee, which is VSB - Technical University of Ostrava, users are CE IT4Innovations third parties - CE IT4Innovations project partners, particularly the University of Ostrava, the Brno University of Technology - Faculty of Informatics, the Silesian University in Opava, Institute of Geonics AS CR.)
- all persons who have a valid license
- students of the Technical University

ANSYS Academic Research

The licence intended to be used for science and research, publications, students' projects (academic licence).

ANSYS COM

The licence intended to be used for science and research, publications, students' projects, commercial research with no commercial use restrictions.


Available Versions

- 16.1
- 17.0

License Preferences

Please see this page to set license preferences.

ANSYS Fluent

ANSYS Fluent  software contains the broad physical modeling capabilities needed to model flow, turbulence, heat transfer, and reactions for industrial applications ranging from air flow over an aircraft wing to combustion in a furnace, from bubble columns to oil platforms, from blood flow to semiconductor manufacturing, and from clean room design to wastewater treatment plants. Special models that give the software the ability to model in-cylinder combustion, aeroacoustics, turbomachinery, and multiphase systems have served to broaden its reach.

1. Common way to run Fluent over pbs file

To run ANSYS Fluent in batch mode you can utilize/modify the default fluent.pbs script and execute it via the qsub command.

```
#!/bin/bash
#PBS -S /bin/bash
#PBS -l nodes=2:ppn=16
#PBS -q qprod
#PBS -N $USER-Fluent-Project
#PBS -A XX-YY-ZZ

#! Mail to user when job terminate or abort
#PBS -m ae


#!change the working directory (default is home directory)
#cd <working directory> (working directory must exists)
WORK_DIR="/scratch/$USER/work"
cd $WORK_DIR

echo Running on host `hostname`
echo Time is `date`
echo Directory is `pwd`
echo This jobs runs on the following processors:
echo `cat $PBS_NODEFILE`

#### Load ansys module so that we find the cfx5solve command
module load ansys

# Use following line to specify MPI for message-passing instead
NCORES=`wc -l $PBS_NODEFILE |awk '{print $1}'`

/ansys_inc/v145/fluent/bin/fluent 3d -t$NCORES -cnf=$PBS_NODEFILE -g -i fluent.jou
```

Header of the pbs file (above) is common and description can be find on this site. SVS FEM  recommends to utilize sources by keywords: nodes, ppn. These keywords allows to address directly the number of nodes (computers) and cores (ppn) which will be utilized in the job. Also the rest of code assumes such structure of allocated resources.

Working directory has to be created before sending pbs job into the queue. Input file should be in working directory or full path to input file has to be specified. Input file has to be defined by common Fluent journal file which is attached to the Fluent solver via parameter -i fluent.jou

Journal file with definition of the input geometry and boundary conditions and defined process of

solution has e.g. the following structure:

```
/file/read-case aircraft_2m.cas.gz
/solve/init
init
/solve/iterate
10
/file/write-case-dat aircraft_2m-solution
/exit yes
```

The appropriate dimension of the problem has to be set by parameter (2d/3d).

2. Fast way to run Fluent from command line

```
fluent solver_version [FLUENT_options] -i journal_file -pbs
```

This syntax will start the ANSYS FLUENT job under PBS Professional using the qsub command in a batch manner. When resources are available, PBS Professional will start the job and return a job ID, usually in the form of *job_ID.hostname*. This job ID can then be used to query, control, or stop the job using standard PBS Professional commands, such as qstat or qdel. The job will be run out of the current working directory, and all output will be written to the file *fluent.o job_ID*.

3. Running Fluent via user's config file

The sample script uses a configuration file called *pbs_fluent.conf* if no command line arguments are present. This configuration file should be present in the directory from which the jobs are submitted (which is also the directory in which the jobs are executed). The following is an example of what the content of *pbs_fluent.conf* can be:

```
input="example_small.flin"
case="Small-1.65m.cas"
fluent_args="3d -pmyninet"
outfile="fluent_test.out"
mpp="true"
```

The following is an explanation of the parameters:

input is the name of the input file.

case is the name of the .cas file that the input file will utilize.

fluent_args are extra ANSYS FLUENT arguments. As shown in the previous example, you can specify the interconnect by using the -p interconnect command. The available interconnects include ethernet (the default), myrinet, infiniband, vendor, altix, and crayx. The MPI is selected automatically, based on the specified interconnect.

outfile is the name of the file to which the standard output will be sent.

mpp="true" will tell the job script to execute the job across multiple processors.

To run ANSYS Fluent in batch mode with user's config file you can utilize/modify the following script and execute it via the qsub command.

```
#!/bin/sh
#PBS -l nodes=2:ppn=4
#PBS -1 qprod
#PBS -N $USE-Fluent-Project
```

```

#PBS -A XX-YY-ZZ

cd $PBS_O_WORKDIR

#We assume that if they didn't specify arguments then they should use the
#config file if [ "xx${input}${case}${mpp}${fluent_args}zz" = "xxzz" ]; then
  if [ -f pbs_fluent.conf ]; then
    . pbs_fluent.conf
  else
    printf "No command line arguments specified, "
    printf "and no configuration file found. Exiting n"
  fi
fi

#Augment the ANSYS FLUENT command line arguments case "$mpp" in
  true)
    #MPI job execution scenario
    num_nodes='cat $PBS_NODEFILE | sort -u | wc -l'
    cpus='expr $num_nodes * $NCPUS'
    #Default arguments for mpp jobs, these should be changed to suit your
    #needs.
    fluent_args="-t${cpus} $fluent_args -cnf=$PBS_NODEFILE"
    ;;
  *)
    #SMP case
    #Default arguments for smp jobs, should be adjusted to suit your
    #needs.
    fluent_args="-t$NCPUS $fluent_args"
    ;;
esac
#Default arguments for all jobs
fluent_args="-ssh -g -i $input $fluent_args"

echo "----- Going to start a fluent job with the following settings:
Input: $input
Case: $case
Output: $outfile
Fluent arguments: $fluent_args"

#run the solver
/ansys_inc/v145/fluent/bin/fluent $fluent_args > $outfile

```

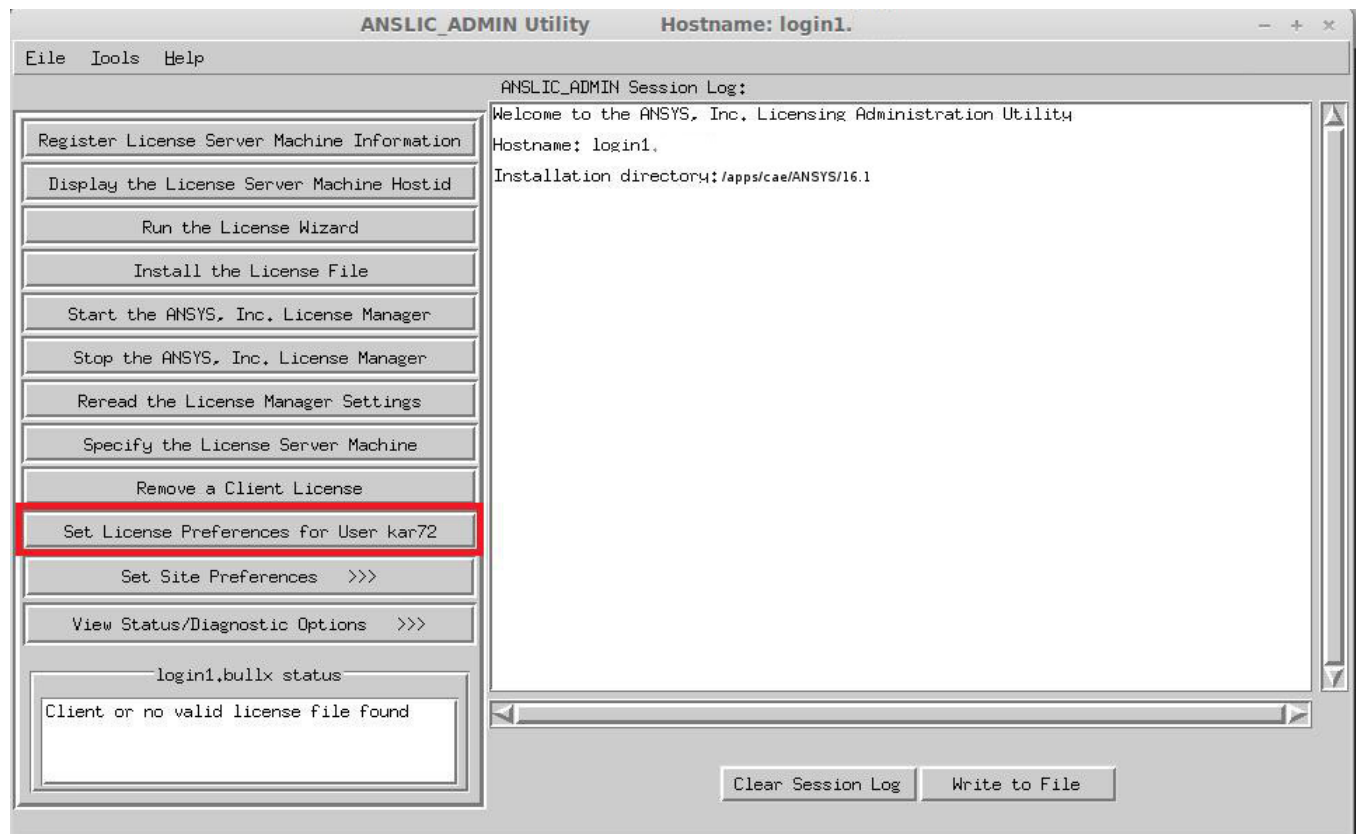
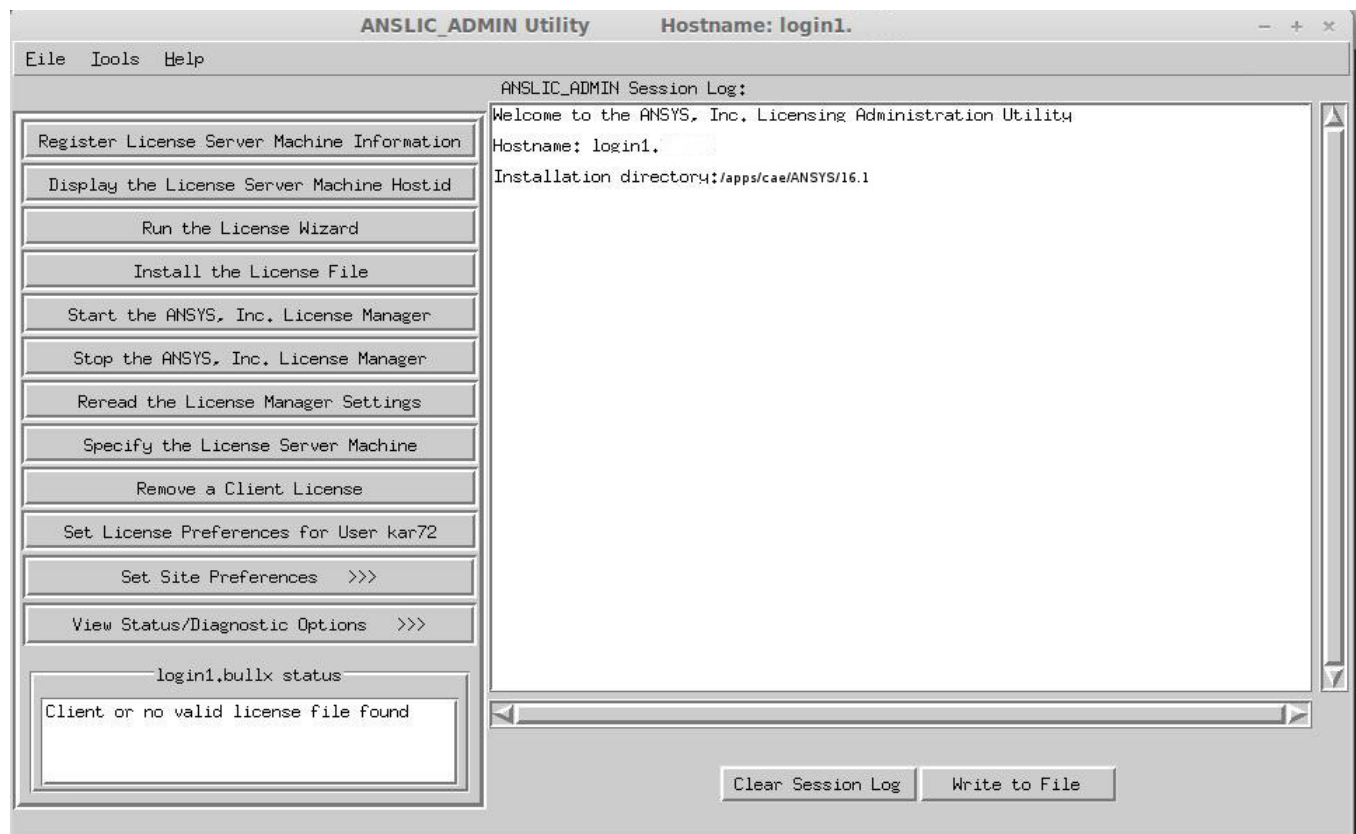
It runs the jobs out of the directory from which they are submitted (PBS_O_WORKDIR).

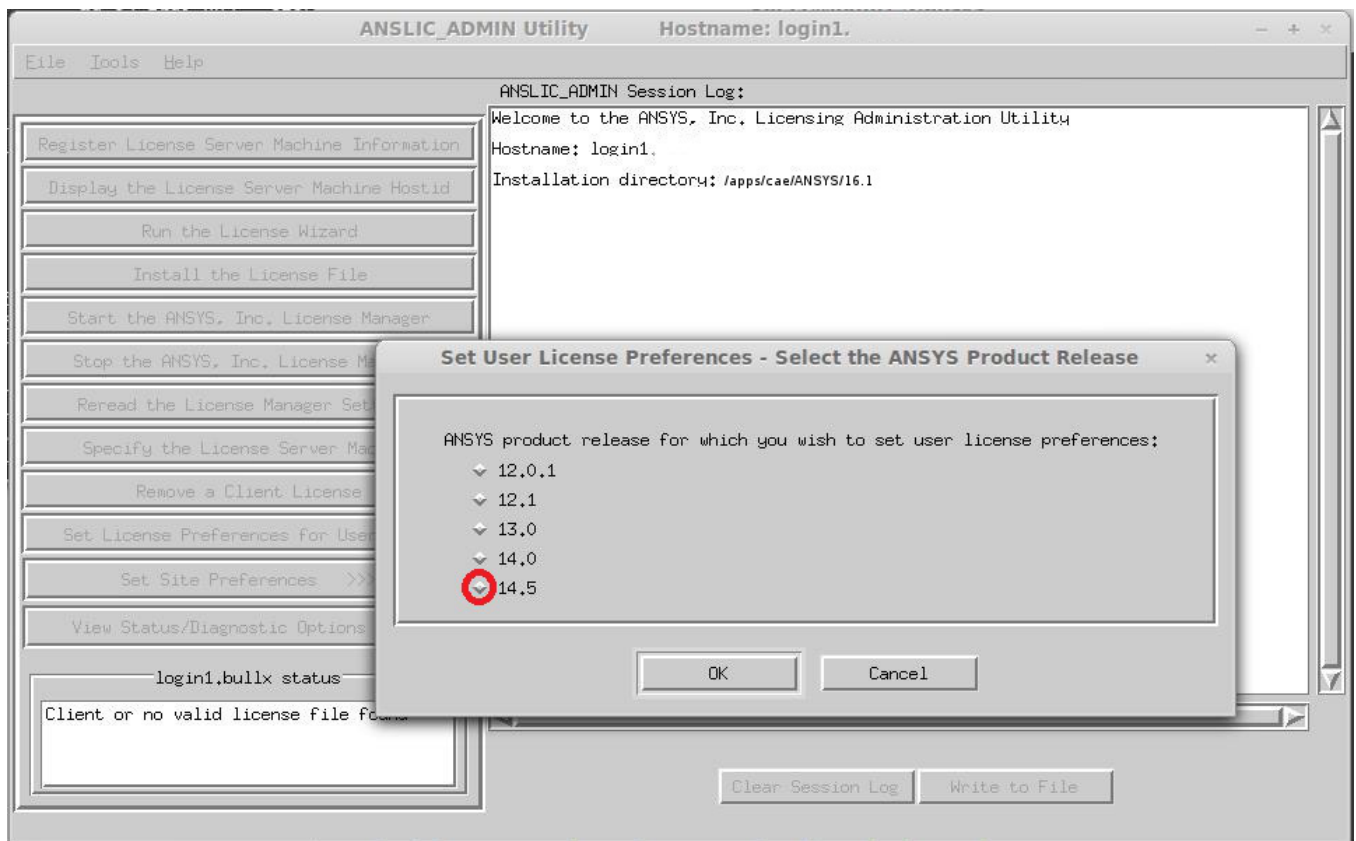
4. Running Fluent in parallel

Fluent could be run in parallel only under Academic Research license. To do so this ANSYS Academic Research license must be placed before ANSYS CFD license in user preferences. To make this change `anslic_admin` utility should be run

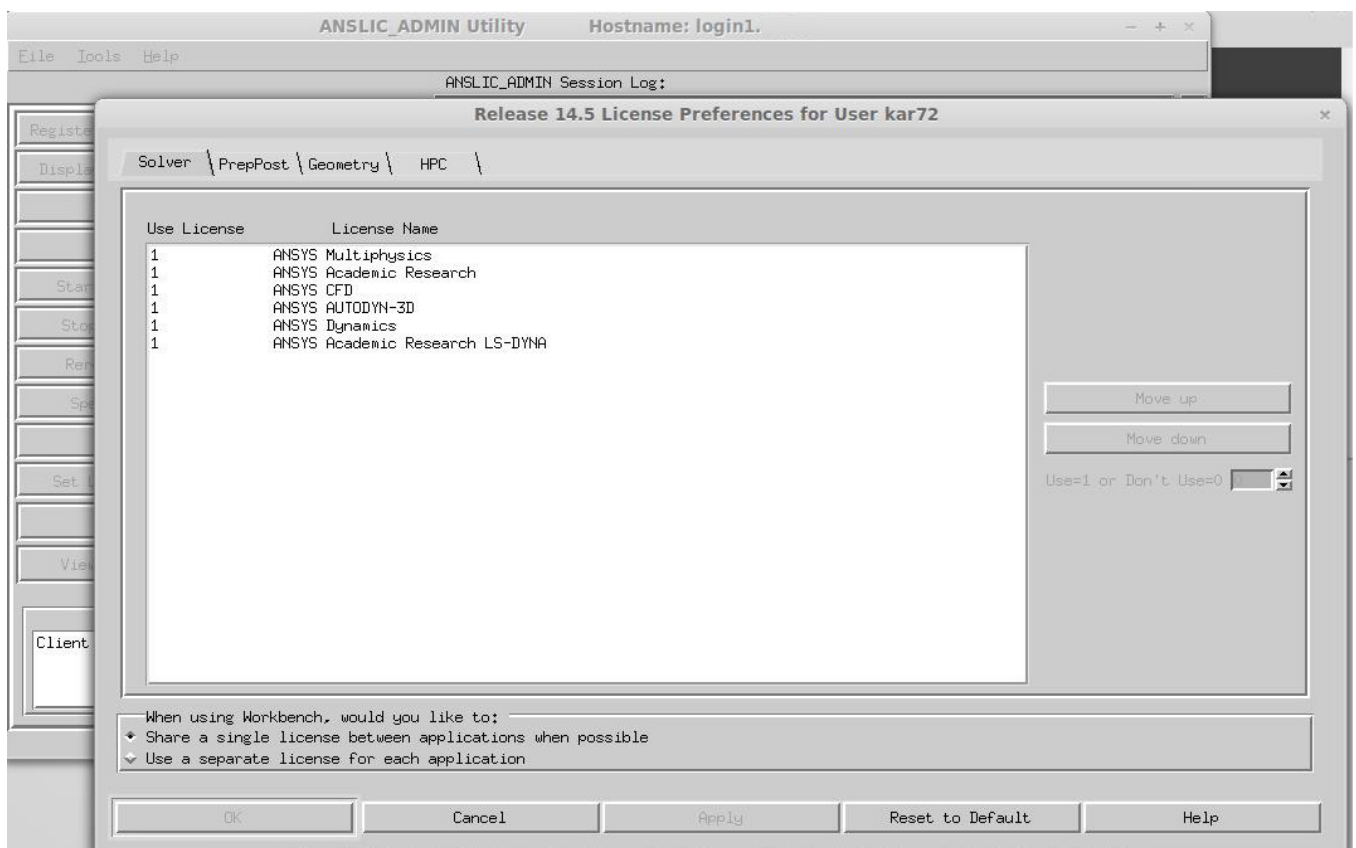
```
/ansys_inc/shared_les/licensing/lic_admin/anslic_admin
```

ANSLIC_ADMIN Utility will be run






ANSYS Academic Research license should be moved up to the top of the list.



ANSYS LS-DYNA

ANSYS LS-DYNA  software provides convenient and easy-to-use access to the technology-rich, time-tested explicit solver without the need to contend with the complex input requirements of this sophisticated program. Introduced in 1996, ANSYS LS-DYNA capabilities have helped customers in numerous industries to resolve highly intricate design issues. ANSYS Mechanical users have been able take advantage of complex explicit solutions for a long time utilizing the traditional ANSYS Parametric Design Language (APDL) environment. These explicit capabilities are available to ANSYS Workbench users as well. The Workbench platform is a powerful, comprehensive, easy-to-use environment for engineering simulation. CAD import from all sources, geometry cleanup, automatic meshing, solution, parametric optimization, result visualization and comprehensive report generation are all available within a single fully interactive modern graphical user environment.

To run ANSYS LS-DYNA in batch mode you can utilize/modify the default `ansysdyna.pbs` script and execute it via the `qsub` command.

```
#!/bin/bash
#PBS -l nodes=2:ppn=16
#PBS -q qprod
#PBS -N $USER-DYNA-Project
#PBS -A XX-YY-ZZ

#! Mail to user when job terminate or abort
#PBS -m ae

#!change the working directory (default is home directory)
#cd <working directory>
WORK_DIR="/scratch/$USER/work"
cd $WORK_DIR

echo Running on host `hostname`
echo Time is `date`
echo Directory is `pwd`
echo This jobs runs on the following processors:
echo `cat $PBS_NODEFILE`

#! Counts the number of processors
NPROCS=`wc -l < $PBS_NODEFILE`

echo This job has allocated $NPROCS nodes

module load ansys

#### Set number of processors per host listing
#### (set to 1 as $PBS_NODEFILE lists each node twice if :ppn=2)
procs_per_host=1
#### Create host list
hl=""
for host in `cat $PBS_NODEFILE`
do
    if [ "$hl" = "" ]
    then hl="$host:$procs_per_host"
    else hl="$hl:$host:$procs_per_host"
```



```
fi
done
```


```
echo Machines: $hl
```

```
/ansys_inc/v145/ansys/bin/ansys145 -dis -lsdynampp i=input.k -machines $hl
```

Header of the pbs file (above) is common and description can be find on this site. SVS FEM [\[4\]](#) recommends to utilize sources by keywords: nodes, ppn. These keywords allows to address directly the number of nodes (computers) and cores (ppn) which will be utilized in the job. Also the rest of code assumes such structure of allocated resources.

Working directory has to be created before sending pbs job into the queue. Input file should be in working directory or full path to input file has to be specified. Input file has to be defined by common LS-DYNA **.k** file which is attached to the ansys solver via parameter i=

ANSYS MAPDL

ANSYS Multiphysics software offers a comprehensive product solution for both multiphysics and single-physics analysis. The product includes structural, thermal, fluid and both high- and low-frequency electromagnetic analysis. The product also contains solutions for both direct and sequentially coupled physics problems including direct coupled-field elements and the ANSYS multi-field solver.

To run ANSYS MAPDL in batch mode you can utilize/modify the default mapdl.pbs script and execute it via the qsub command.

```
#!/bin/bash
#PBS -l nodes=2:ppn=16
#PBS -q qprod
#PBS -N $USER-ANSYS-Project
#PBS -A XX-YY-ZZ

#! Mail to user when job terminate or abort
#PBS -m ae

#!change the working directory (default is home directory)
#cd <working directory> (working directory must exists)
WORK_DIR="/scratch/$USER/work"
cd $WORK_DIR

echo Running on host `hostname`
echo Time is `date`
echo Directory is `pwd`
echo This jobs runs on the following processors:
echo `cat $PBS_NODEFILE`

module load ansys

#### Set number of processors per host listing
#### (set to 1 as $PBS_NODEFILE lists each node twice if :ppn=2)
procs_per_host=1
#### Create host list
hl=""
for host in `cat $PBS_NODEFILE`
do
    if [ "$hl" = "" ]
    then hl="$host:$procs_per_host"
    else hl="{hl}:$host:$procs_per_host"
    fi
done

echo Machines: $hl

#-i input.dat includes the input of analysis in APDL format
#-o file.out is output file from ansys where all text outputs will be redirected
#-p the name of license feature (aa_r=ANSYS Academic Research, ane3fl=Multiphysics(comm
/ansys_inc/v145/ansys/bin/ansys145 -b -dis -p aa_r -i input.dat -o file.out -machines $hl
```


Header of the pbs file (above) is common and description can be find on this site. SVS FEM

recommends to utilize sources by keywords: nodes, ppn. These keywords allows to address directly the number of nodes (computers) and cores (ppn) which will be utilized in the job. Also the rest of code assumes such structure of allocated resources.

Working directory has to be created before sending pbs job into the queue. Input file should be in working directory or full path to input file has to be specified. Input file has to be defined by common APDL file which is attached to the ansys solver via parameter -i

License should be selected by parameter -p. Licensed products are the following: aa_r (ANSYS **Academic** Research), ane3fl (ANSYS Multiphysics)-**Commercial**, aa_r_dy (ANSYS **Academic** AUTODYN) More about licensing here

ANSYS CFX

ANSYS CFX  software is a high-performance, general purpose fluid dynamics program that has been applied to solve wide-ranging fluid flow problems for over 20 years. At the heart of ANSYS CFX is its advanced solver technology, the key to achieving reliable and accurate solutions quickly and robustly. The modern, highly parallelized solver is the foundation for an abundant choice of physical models to capture virtually any type of phenomena related to fluid flow. The solver and its many physical models are wrapped in a modern, intuitive, and flexible GUI and user environment, with extensive capabilities for customization and automation using session files, scripting and a powerful expression language.

To run ANSYS CFX in batch mode you can utilize/modify the default cfx.pbs script and execute it via the qsub command.

```
#!/bin/bash
#PBS -l nodes=2:ppn=16
#PBS -q qprod
#PBS -N $USER-CFX-Project
#PBS -A XX-YY-ZZ

#! Mail to user when job terminate or abort
#PBS -m ae

#!change the working directory (default is home directory)
#cd <working directory> (working directory must exists)
WORK_DIR="/scratch/$USER/work"
cd $WORK_DIR

echo Running on host `hostname`
echo Time is `date`
echo Directory is `pwd`
echo This jobs runs on the following processors:
echo `cat $PBS_NODEFILE`

module load ansys

#### Set number of processors per host listing
#### (set to 1 as $PBS_NODEFILE lists each node twice if :ppn=2)
procs_per_host=1
#### Create host list
hl=""
for host in `cat $PBS_NODEFILE`
do
    if [ "$hl" = "" ]
    then hl="$host:$procs_per_host"
    else hl="{hl}:$host:$procs_per_host"
    fi
done

echo Machines: $hl

#-dev input.def includes the input of CFX analysis in DEF format
#-P the name of preferred license feature (aa_r=ANSYS Academic Research, ane3fl=Multiphysics)
```

```
/ansys_inc/v145/CFX/bin/cfx5solve -def input.def -size 4 -size-ni 4x -part-large -start-m
```

Header of the pbs file (above) is common and description can be find on this site. SVS FEM recommends to utilize sources by keywords: nodes, ppn. These keywords allows to address directly the number of nodes (computers) and cores (ppn) which will be utilized in the job. Also the rest of code assumes such structure of allocated resources.

Working directory has to be created before sending pbs job into the queue. Input file should be in working directory or full path to input file has to be specified. >Input file has to be defined by common CFX def file which is attached to the cfx solver via parameter -def

License should be selected by parameter -P (Big letter **P**). Licensed products are the following: aa_r (ANSYS **Academic** Research), ane3fl (ANSYS Multiphysics)-**Commercial**. More about licensing here

Workbench

Workbench Batch Mode

It is possible to run Workbench scripts in batch mode. You need to configure solvers of individual components to run in parallel mode. Open your project in Workbench. Then, for example, in Mechanical, go to Tools - Solve Process Settings ...

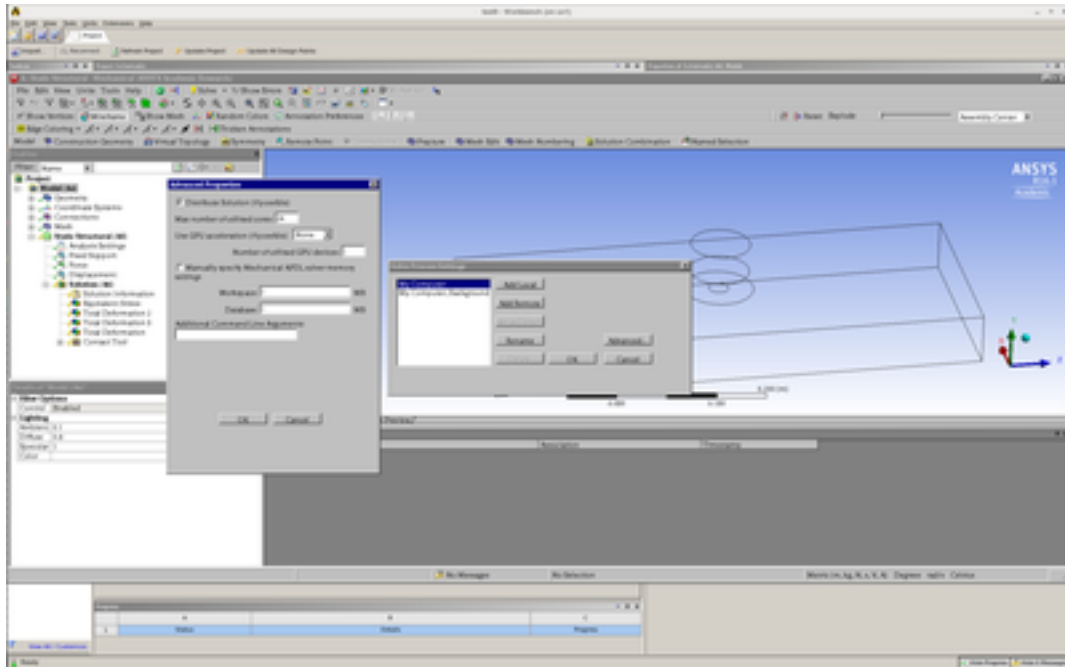


Figure 1:

Enable Distribute Solution checkbox and enter number of cores (eg. 48 to run on two Salomon nodes). If you want the job to run on more than 1 node, you must also provide a so called MPI appfile. In the Additional Command Line Arguments input field, enter:

```
-mpifile /path/to/my/job/mpifile.txt
```

Where /path/to/my/job is the directory where your project is saved. We will create the file mpi-file.txt programatically later in the batch script. For more information, refer to *ANSYS Mechanical APDL Parallel Processing Guide*.

Now, save the project and close Workbench. We will use this script to launch the job:

```
#!/bin/bash
#PBS -l select=2:ncpus=24
#PBS -q qprod
#PBS -N test9_mpi_2
#PBS -A OPEN-0-0

# Mail to user when job terminate or abort
#PBS -m a

# change the working directory
WORK_DIR="$PBS_O_WORKDIR"
cd $WORK_DIR

echo Running on host `hostname`
```

```

echo Time is `date`
echo Directory is `pwd`
echo This jobs runs on the following nodes:
echo `cat $PBS_NODEFILE`

module load ANSYS

#### Set number of processors per host listing
procs_per_host=24
#### Create MPI appfile
echo -n "" > mpifile.txt
for host in `cat $PBS_NODEFILE`
do
    echo "-h $host -np $procs_per_host $ANSYS160_DIR/bin/ansysdis161 -dis" > mpifile.txt
done

#-i input.dat includes the input of analysis in APDL format
#-o file.out is output file from ansys where all text outputs will be redirected
#-p the name of license feature (aa_r=ANSYS Academic Research, ane3fl=Multiphysics)

# prevent using scsif0 interface on accelerated nodes
export MPI_IC_ORDER="UDAPL"
# spawn remote process using SSH (default is RSH)
export MPI_REMSH="/usr/bin/ssh"

runwb2 -R jou6.wbjn -B -F test9.wbpj

```

The solver settings are saved in file solvehandlers.xml, which is not located in the project directory. Verify your solved settings when uploading a project from your local computer.

Overview of ANSYS Products

SVS FEM  as **ANSYS Channel partner**  for Czech Republic provided all ANSYS licenses for ANSELM cluster and supports of all ANSYS Products (Multiphysics, Mechanical, MAPDL, CFX, Fluent, Maxwell, LS-DYNA...) to IT staff and ANSYS users. If you are challenging to problem of ANSYS functionality contact please hotline@svsfem.cz 

Anselm provides as commercial as academic variants. Academic variants are distinguished by “**Academic...**” word in the name of license or by two letter preposition “**aa_**” in the license feature name. Change of license is realized on command line respectively directly in user’s pbs file (see individual products). More about licensing here

To load the latest version of any ANSYS product (Mechanical, Fluent, CFX, MAPDL,...) load the module:


```
$ module load ansys
```






ANSYS supports interactive regime, but due to assumed solution of extremely difficult tasks it is not recommended.

If user needs to work in interactive regime we recommend to configure the RSM service on the client machine which allows to forward the solution to the Anselm directly from the client’s Workbench project (see ANSYS RSM service).

COMSOL Multiphysics®

Introduction

COMSOL  is a powerful environment for modelling and solving various engineering and scientific problems based on partial differential equations. COMSOL is designed to solve coupled or multi-physics phenomena. For many standard engineering problems COMSOL provides add-on products such as electrical, mechanical, fluid flow, and chemical applications.

- Structural Mechanics Module ,
- Heat Transfer Module ,
- CFD Module ,
- Acoustics Module ,
- and many others .

COMSOL also allows an interface support for equation-based modelling of partial differential equations.

Execution

On the clusters COMSOL is available in the latest stable version. There are two variants of the release:

- **Non commercial** or so called **>EDU variant>**, which can be used for research and educational purposes.
- **Commercial** or so called **COM variant**, which can be used also for commercial activities. **COM variant** has only subset of features compared to the **EDU variant** available. More about licensing will be posted here soon.

To load the of COMSOL load the module

```
$ module load COMSOL/51-EDU
```

By default the **EDU variant** will be loaded. If user needs other version or variant, load the particular version. To obtain the list of available versions use

```
$ module avail COMSOL
```

If user needs to prepare COMSOL jobs in the interactive mode it is recommend to use COMSOL on the compute nodes via PBS Pro scheduler. In order run the COMSOL Desktop GUI on Windows is recommended to use the Virtual Network Computing (VNC).

```
$ xhost +  
$ qsub -I -X -A PROJECT_ID -q qprod -l select=1:ppn=24  
$ module load COMSOL  
$ comsol
```

To run COMSOL in batch mode, without the COMSOL Desktop GUI environment, user can utilized the default (comsol.pbs) job script and execute it via the qsub command.

```
#!/bin/bash  
#PBS -l select=3:ppn=24  
#PBS -q qprod  
#PBS -N JOB_NAME  
#PBS -A PROJECT_ID
```

```
cd /scratch/work/user/$USER/ || exit
```

```
echo Time is `date`  
echo Directory is `pwd`  
echo '**PBS_NODEFILE***START*****'  
cat $PBS_NODEFILE  
echo '**PBS_NODEFILE***END*****'
```

```
text_nodes < cat $PBS_NODEFILE
```

```
module load COMSOL  
# module load COMSOL/51-EDU
```

```
ntask=$(wc -l $PBS_NODEFILE)
```

```
comsol -nn ${ntask} batch -configuration /tmp -mpiarg -rmk -mpiarg pbs -tmpdir /scratch/$
```

Working directory has to be created before sending the (comsol.pbs) job script into the queue. Input file (name_input_f.mph) has to be in working directory or full path to input file has to be specified. The appropriate path to the temp directory of the job has to be set by command option (-tmpdir).

LiveLink™ for MATLAB®

COMSOL is the software package for the numerical solution of the partial differential equations. LiveLink for MATLAB allows connection to the COMSOL®API (Application Programming Interface) with the benefits of the programming language and computing environment of the MATLAB.

LiveLink for MATLAB is available in both **EDU** and **COM variant** of the COMSOL release. On the clusters 1 commercial (**COM**) license and the 5 educational (**EDU**) licenses of LiveLink for MATLAB (please see the ISV Licenses) are available. Following example shows how to start COMSOL model from MATLAB via LiveLink in the interactive mode.

```
$ xhost +  
$ qsub -I -X -A PROJECT_ID -q qexp -l select=1:ppn=24  
$ module load MATLAB  
$ module load COMSOL  
$ comsol server MATLAB
```

At the first time to launch the LiveLink for MATLAB (client-MATLAB/server-COMSOL connection) the login and password is requested and this information is not requested again.

To run LiveLink for MATLAB in batch mode with (comsol_matlab.pbs) job script you can utilize/modify the following script and execute it via the qsub command.

```
#!/bin/bash  
#PBS -l select=3:ppn=24  
#PBS -q qprod  
#PBS -N JOB_NAME  
#PBS -A PROJECT_ID  
  
cd /scratch/work/user/$USER || exit  
  
echo Time is `date`  
echo Directory is `pwd`  
echo '**PBS_NODEFILE***START*****'
```

```

cat $PBS_NODEFILE
echo '**PBS_NODEFILE***END*****'

text_nodes < cat $PBS_NODEFILE

module load MATLAB
module load COMSOL/51-EDU

ntask=$(wc -l $PBS_NODEFILE)

comsol -nn ${ntask} server -configuration /tmp -mpiarg -rmk -mpiarg pbs -tmpdir /scratch/
cd /apps/cae/COMSOL/51/mli
matlab -nodesktop -nosplash -r "mphstart; addpath /scratch/work/user/$USER/work; test_job"

```

This example shows how to run Livelink for MATLAB with following configuration: 3 nodes and 16 cores per node. Working directory has to be created before submitting (comsol_matlab.pbs) job script into the queue. Input file (test_job.m) has to be in working directory or full path to input file has to be specified. The Matlab command option (-r "mphstart") created a connection with a COMSOL server using the default port number.

Licensing and Available Versions

Comsol licence can be used by:

- all persons in the carrying out of the CE IT4Innovations Project (In addition to the primary licensee, which is VSB - Technical University of Ostrava, users are CE IT4Innovations third parties - CE IT4Innovations project partners, particularly the University of Ostrava, the Brno University of Technology - Faculty of Informatics, the Silesian University in Opava, Institute of Geonics AS CR.)
- all persons who have a valid license
- students of the Technical University

Comsol EDU Network Licence

The licence intended to be used for science and research, publications, students' projects, teaching (academic licence).

Comsol COM Network Licence

The licence intended to be used for science and research, publications, students' projects, commercial research with no commercial use restrictions. Enables the solution of at least one job by one user in one program start.

Available Versions

- ver. 51

Java

Java on the cluster

Java is available on the cluster. Activate java by loading the Java module

```
$ module load Java
```

Note that the Java module must be loaded on the compute nodes as well, in order to run java on compute nodes.

Check for java version and path

```
$ java -version
$ which java
```

With the module loaded, not only the runtime environment (JRE), but also the development environment (JDK) with the compiler is available.

```
$ javac -version
$ which javac
```

Java applications may use MPI for interprocess communication, in conjunction with OpenMPI. Read more on <http://www.open-mpi.org/faq/?category=java>. This functionality is currently not supported on Anselm cluster. In case you require the java interface to MPI, please contact cluster support.

Allinea Forge (DDT,MAP)

Allinea Forge consist of two tools - debugger DDT and profiler MAP.

Allinea DDT, is a commercial debugger primarily for debugging parallel MPI or OpenMP programs. It also has a support for GPU (CUDA) and Intel Xeon Phi accelerators. DDT provides all the standard debugging features (stack trace, breakpoints, watches, view variables, threads etc.) for every thread running as part of your program, or for every process - even if these processes are distributed across a cluster using an MPI implementation.

Allinea MAP is a profiler for C/C++/Fortran HPC codes. It is designed for profiling parallel code, which uses pthreads, OpenMP or MPI.

License and Limitations for Anselm Users

On Anselm users can debug OpenMP or MPI code that runs up to 64 parallel processes. In case of debugging GPU or Xeon Phi accelerated codes the limit is 8 accelerators. These limitation means that:

- 1 user can debug up 64 processes, or
- 32 users can debug 2 processes, etc.

In case of debugging on accelerators:

- 1 user can debug on up to 8 accelerators, or
- 8 users can debug on single accelerator.

Compiling Code to run with DDT

Modules

Load all necessary modules to compile the code. For example:

```
$ module load intel
$ module load impi ... or ... module load openmpi/X.X.X-icc
```

Load the Allinea DDT module:

```
$ module load Forge
```

Compile the code:

```
$ mpicc -g -O0 -o test_debug test.c
```

```
$ mpif90 -g -O0 -o test_debug test.f
```

Compiler flags

Before debugging, you need to compile your code with theses flags:

!!! Note “Note” - **g** : Generates extra debugging information usable by GDB. -g3 includes even more debugging information. This option is available for GNU and INTEL C/C++ and Fortran compilers.

- - ****O0**** : Suppress all optimizations.

Starting a Job with DDT

Be sure to log in with an X window forwarding enabled. This could mean using the -X in the ssh:

```
$ ssh -X username@anselm.it4i.cz
```

Other options is to access login node using VNC. Please see the detailed information on how to use graphic user interface on Anselm

From the login node an interactive session **with X windows forwarding** (-X option) can be started by following command:

```
$ qsub -I -X -A NONE-0-0 -q qexp -lselect=1:ncpus=16:mpiprocs=16,walltime=01:00:00
```

Then launch the debugger with the ddt command followed by the name of the executable to debug:

```
$ ddt test_debug
```

A submission window that appears have a prefilled path to the executable to debug. You can select the number of MPI processors and/or OpenMP threads on which to run and press run. Command line arguments to a program can be entered to the “Arguments” box.

To start the debugging directly without the submission window, user can specify the debugging and execution parameters from the command line. For example the number of MPI processes is set by option “-np 4”. Skipping the dialog is done by “-start” option. To see the list of the “ddt” command line parameters, run “ddt -help”.

```
ddt -start -np 4 ./hello_debug impi
```

Documentation

Users can find original User Guide after loading the DDT module:

```
$DDTPATH/doc/userguide.pdf
```

[1] Discipline, Magic, Inspiration and Science: Best Practice Debugging with Allinea DDT, Workshop conducted at LLNL by Allinea on May 10, 2013, link [🔗](#)

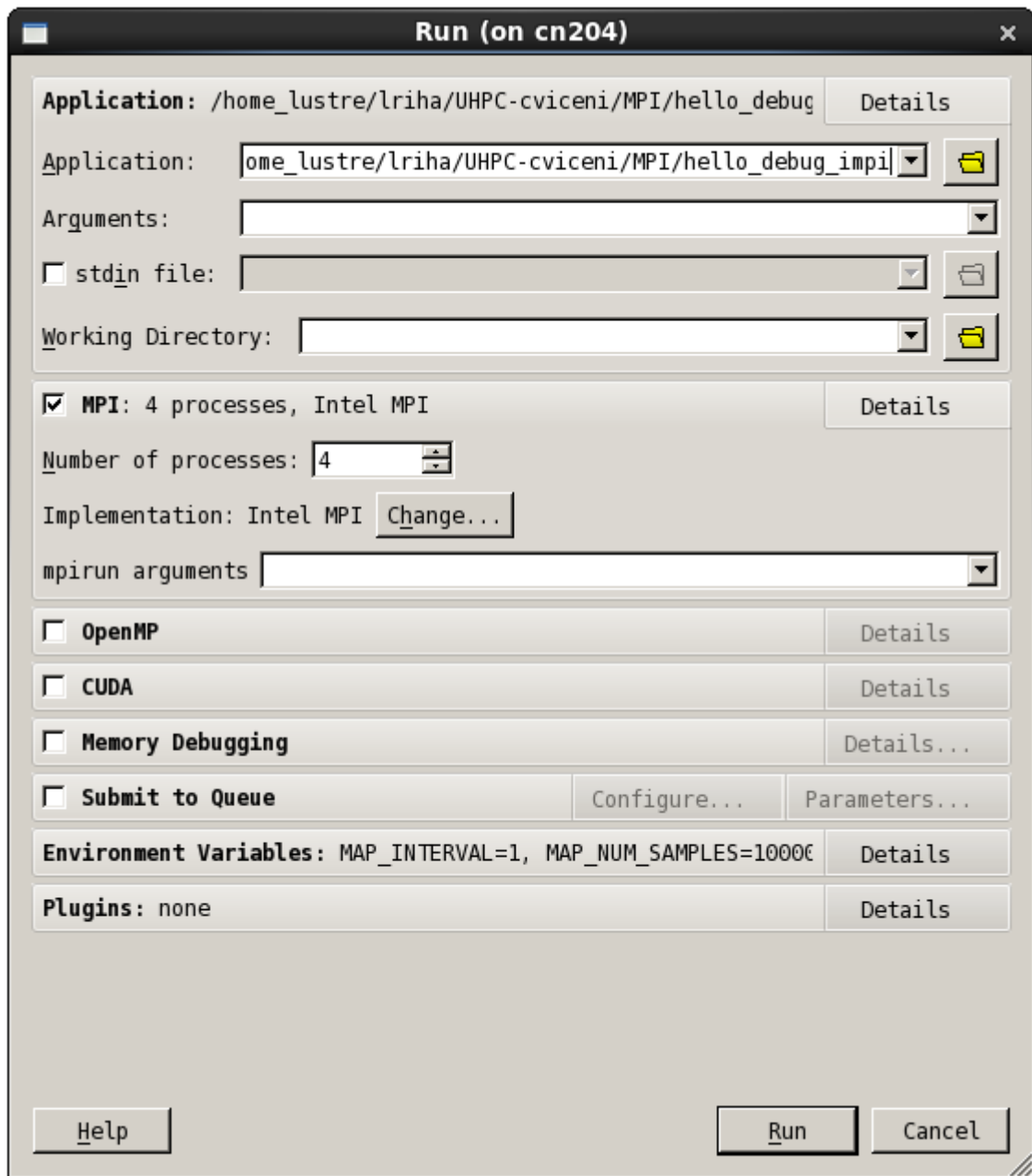


Figure 1:

Allinea Performance Reports

Introduction

Allinea Performance Reports characterize the performance of HPC application runs. After executing your application through the tool, a synthetic HTML report is generated automatically, containing information about several metrics along with clear behavior statements and hints to help you improve the efficiency of your runs.

The Allinea Performance Reports is most useful in profiling MPI programs.

Our license is limited to 64 MPI processes.

Modules

Allinea Performance Reports version 6.0 is available

```
$ module load PerformanceReports/6.0
```

The module sets up environment variables, required for using the Allinea Performance Reports.

Usage

Use the the perf-report wrapper on your (MPI) program.

Instead of running your MPI program the usual way, use the the perf report wrapper:

```
$ perf-report mpirun ./mympiprogram.x
```

The mpi program will run as usual. The perf-report creates two additional files, in *.txt* and *.html* format, containing the performance report. Note that demanding MPI codes should be run within the queue system.

Example

In this example, we will be profiling the mympiprogram.x MPI program, using Allinea performance reports. Assume that the code is compiled with intel compilers and linked against intel MPI library:

First, we allocate some nodes via the express queue:

```
$ qsub -q qexp -l select=2:ppn=24:mpiprocs=24:ompthreads=1 -I
qsub: waiting for job 262197.dm2 to start
qsub: job 262197.dm2 ready
```

Then we load the modules and run the program the usual way:


```
$ module load intel impi PerfReports/6.0
$ mpirun ./mympiprogram.x
```

Now lets profile the code:

```
$ perf-report mpirun ./mympiprogram.x
```

Performance report files mympiprogram_32p*.txt and mympiprogram_32p*.html were created. We can see that the code is very efficient on MPI and is CPU bounded.

Aislinn

- Aislinn is a dynamic verifier for MPI programs. For a fixed input it covers all possible runs with respect to nondeterminism introduced by MPI. It allows to detect bugs (for sure) that occurs very rare in normal runs.
- Aislinn detects problems like invalid memory accesses, deadlocks, misuse of MPI, and resource leaks.
- Aislinn is open-source software; you can use it without any licensing limitations.
- Web page of the project: <http://verif.cs.vsb.cz/aislinn/> 

!!! Note “Note” Aislinn is software developed at IT4Innovations and some parts are still considered experimental. If you have any questions or experienced any problems, please contact the author: stanislav.bohm@vsb.cz.

Usage

Let us have the following program that contains a bug that is not manifested in all runs:

```
#include <mpi.h>
#include <stdlib.h>

int main(int argc, char **argv) {
    int rank;

    MPI_Init(&argc, &argv);
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);

    if (rank == 0) {
        int *mem1 = (int*) malloc(sizeof(int) * 2);
        int *mem2 = (int*) malloc(sizeof(int) * 3);
        int data;
        MPI_Recv(&data, 1, MPI_INT, MPI_ANY_SOURCE, 1,
                 MPI_COMM_WORLD, MPI_STATUS_IGNORE);
        mem1[data] = 10; // <----- Possible invalid memory write
        MPI_Recv(&data, 1, MPI_INT, MPI_ANY_SOURCE, 1,
                 MPI_COMM_WORLD, MPI_STATUS_IGNORE);
        mem2[data] = 10;
        free(mem1);
        free(mem2);
    }

    if (rank == 1 || rank == 2) {
        MPI_Send(&rank, 1, MPI_INT, 0, 1, MPI_COMM_WORLD);
    }

    MPI_Finalize();
    return 0;
}
```

The program does the following: process 0 receives two messages from anyone and processes 1 and 2 send a message to process 0. If a message from process 1 is received first, then the run does not expose the error. If a message from process 2 is received first, then invalid memory write occurs at line 16.

To verify this program by Aislinn, we first load Aislinn itself:

```
$ module load aislinn
```

Now we compile the program by Aislinn implementation of MPI. There are `mpicc` for C programs and `mpicxx` for C++ programs. Only MPI parts of the verified application has to be recompiled; non-MPI parts may remain untouched. Let us assume that our program is in `test.cpp`.

```
$ mpicc -g test.cpp -o test
```

The `-g` flag is not necessary, but it puts more debugging information into the program, hence Aislinn may provide more detailed report. The command produces executable file `test`.

Now we run the Aislinn itself. The argument `-p 3` specifies that we want to verify our program for the case of three MPI processes

```
$ aislinn -p 3 ./test
==AN== INFO: Aislinn v0.3.0
==AN== INFO: Found error 'Invalid write'
==AN== INFO: 1 error(s) found
==AN== INFO: Report written into 'report.html'
```

Aislinn found an error and produced HTML report. To view it, we can use any browser, e.g.:

```
$ firefox report.html
```

At the beginning of the report there are some basic summaries of the verification. In the second part (depicted in the following picture), the error is described.

Error: Invalid write (mem/invalid-write)

Description	Invalid write of size 4 at address 0x802401038		
Process rank in COMM_WORLD	0		
Stack trace	0x40090B: main (test.cpp:16)		
Output (stdout) for pid 1			
Output (stdout) for pid 2			
Output (stdout) for pid 3			
Error output (stderr) for pid 1			
Error output (stderr) for pid 2			
Error output (stderr) for pid 3			
Events			
#	0	1	2
1	MPI_Comm_rank	MPI_Comm_rank	MPI_Comm_rank
2	MPI_Recv	MPI_Send	MPI_Send
3	Continue	Continue	Continue
4		MPI_Finalize	MPI_Finalize
5		Exit	Exit

Arguments: 68702686988,1,267387145,0,1,52226
0x403703: aislinn_call (aislinn.c:61)
0x401D21: MPI_Send (mpi_generated.c:104)
0x4009AA: main (test.cpp:25)

Figure 1:

It shows us: - Error occurs in process 0 in `test.cpp` on line 16. - Stdout and stderr streams are empty. (The program does not write anything). - The last part shows MPI calls for each process

that occurs in the invalid run. The more detailed information about each call can be obtained by mouse cursor.

Limitations

Since the verification is a non-trivial process there are some of limitations.

- The verified process has to terminate in all runs, i.e. we cannot answer the halting problem.
- The verification is a computationally and memory demanding process. We put an effort to make it efficient and it is an important point for further research. However covering all runs will be always more demanding than techniques that examines only a single run. The good practise is to start with small instances and when it is feasible, make them bigger. The Aislinn is good to find bugs that are hard to find because they occur very rarely (only in a rare scheduling). Such bugs often do not need big instances.
- Aislinn expects that your program is a “standard MPI” program, i.e. processes communicate only through MPI, the verified program does not interacts with the system in some unusual ways (e.g. opening sockets).

There are also some limitations bounded to the current version and they will be removed in the future:

- All files containing MPI calls have to be recompiled by MPI implementation provided by Aislinn. The files that does not contain MPI calls, they do not have to recompiled. Aislinn MPI implementation supports many commonly used calls from MPI-2 and MPI-3 related to point-to-point communication, collective communication, and communicator management. Unfortunately, MPI-IO and one-side communication is not implemented yet.
- Each MPI can use only one thread (if you use OpenMP, set OMP_NUM_THREADS to 1).
- There are some limitations for using files, but if the program just reads inputs and writes results, it is ok.

Vampir

Vampir is a commercial trace analysis and visualisation tool. It can work with traces in OTF and OTF2 formats. It does not have the functionality to collect traces, you need to use a trace collection tool (such as Score-P) first to collect the traces.

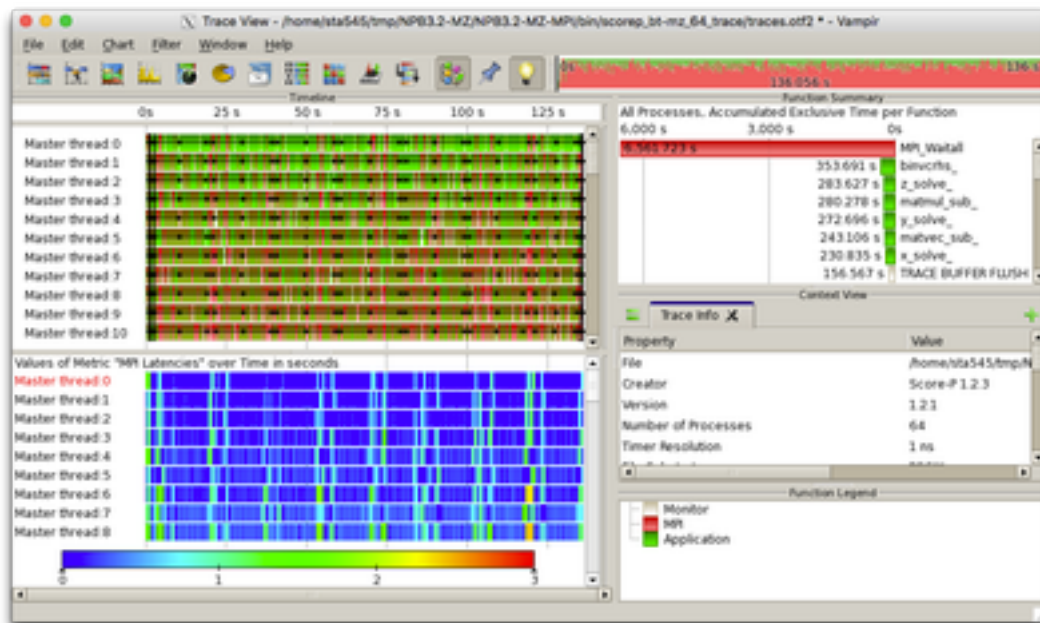


Figure 1:

Installed versions

Version 8.5.0 is currently installed as module Vampir/8.5.0 :

```
$ module load Vampir/8.5.0
$ vampir &
```

User manual

You can find the detailed user manual in PDF format in `$EBROOTVAMPIR/doc/vampir-manual.pdf`

References

1. <https://www.vampir.eu>

Debuggers and profilers summary

Introduction

We provide state of the art programmes and tools to develop, profile and debug HPC codes at IT4Innovations. On these pages, we provide an overview of the profiling and debugging tools available on Anslem at IT4I.

Intel debugger

Intel debugger is no longer available since Parallel Studio version 2015

The intel debugger version 13.0 is available, via module intel. The debugger works for applications compiled with C and C++ compiler and the ifort fortran 77/90/95 compiler. The debugger provides java GUI environment.

```
$ module load intel
$ idb
```

Read more at the Intel Debugger page.

Allinea Forge (DDT/MAP)

Allinea DDT, is a commercial debugger primarily for debugging parallel MPI or OpenMP programs. It also has a support for GPU (CUDA) and Intel Xeon Phi accelerators. DDT provides all the standard debugging features (stack trace, breakpoints, watches, view variables, threads etc.) for every thread running as part of your program, or for every process - even if these processes are distributed across a cluster using an MPI implementation.

```
$ module load Forge
$ forge
```

Read more at the Allinea DDT page.

Allinea Performance Reports

Allinea Performance Reports characterize the performance of HPC application runs. After executing your application through the tool, a synthetic HTML report is generated automatically, containing information about several metrics along with clear behavior statements and hints to help you improve the efficiency of your runs. Our license is limited to 64 MPI processes.

```
$ module load PerformanceReports/6.0
$ perf-report mpirun -n 64 ./my_application argument01 argument02
```

Read more at the Allinea Performance Reports page.

RougeWave Totalview

TotalView is a source- and machine-level debugger for multi-process, multi-threaded programs. Its wide range of tools provides ways to analyze, organize, and test programs, making it easy to isolate and identify problems in individual threads and processes in programs of great complexity.

```
$ module load TotalView/8.15.4-6-linux-x86-64  
$ totalview
```

Read more at the Totalview page.

Vampir trace analyzer

Vampir is a GUI trace analyzer for traces in OTF format.

```
$ module load Vampir/8.5.0  
$ vampir
```

Read more at the Vampir page.

Valgrind

About Valgrind

Valgrind is an open-source tool, used mainly for debuggig memory-related problems, such as memory leaks, use of uninitialized memory etc. in C/C++ applications. The toolchain was however extended over time with more functionality, such as debugging of threaded applications, cache profiling, not limited only to C/C++.

Valgind is an extremely useful tool for debugging memory errors such as off-by-one[☞](#). Valgrind uses a virtual machine and dynamic recompilation of binary code, because of that, you can expect that programs being debugged by Valgrind run 5-100 times slower.

The main tools available in Valgrind are :

- **Memcheck**, the original, must used and default tool. Verifies memory access in you program and can detect use of uninitialized memory, out of bounds memory access, memory leaks, double free, etc.
- **Massif**, a heap profiler.
- **Hellgrind** and **DRD** can detect race conditions in multi-threaded applications.
- **Cachegrind**, a cache profiler.
- **Callgrind**, a callgraph analyzer.
- For a full list and detailed documentation, please refer to the official Valgrind documentation[☞](#).

Installed versions

There are two versions of Valgrind available on the cluster.

- Version 3.8.1, installed by operating system vendor in /usr/bin/valgrind. This version is available by default, without the need to load any module. This version however does not provide additional MPI support. Also, it does not support AVX2 instructions, debugging of an AVX2-enabled executable with this version will fail
- Version 3.11.0 built by ICC with support for Intel MPI, available in module Valgrind/3.11.0-intel-2015b. After loading the module, this version replaces the default valgrind.
- Version 3.11.0 built by GCC with support for Open MPI, module Valgrind/3.11.0-foss-2015b

Usage

Compile the application which you want to debug as usual. It is advisable to add compilation flags -g (to add debugging information to the binary so that you will see original source code lines in the output) and -O0 (to disable compiler optimizations).

For example, lets look at this C code, which has two problems:

```
#include <stdlib.h>

void f(void)
{
    int* x = malloc(10 * sizeof(int));
    x[10] = 0; // problem 1: heap block overrun
}             // problem 2: memory leak -- x not freed
```



```

int main(void)
{
    f();
    return 0;
}

```

Now, compile it with Intel compiler:

```

$ module add intel
$ icc -g valgrind-example.c -o valgrind-example

```

Now, lets run it with Valgrind. The syntax is:

```
valgrind [valgrind options] <your program binary> [your program options]
```

If no Valgrind options are specified, Valgrind defaults to running Memcheck tool. Please refer to the Valgrind documentation for a full description of command line options.

```

$ valgrind ./valgrind-example
==12652== Memcheck, a memory error detector
==12652== Copyright (C) 2002-2013, and GNU GPL'd, by Julian Seward et al.
==12652== Using Valgrind-3.9.0 and LibVEX; rerun with -h for copyright info
==12652== Command: ./valgrind-example
==12652==
==12652== Invalid write of size 4
==12652== at 0x40053E: f (valgrind-example.c:6)
==12652== by 0x40054E: main (valgrind-example.c:11)
==12652== Address 0x5861068 is 0 bytes after a block of size 40 alloc'd
==12652== at 0x4C27AAA: malloc (vg_replace_malloc.c:291)
==12652== by 0x400528: f (valgrind-example.c:5)
==12652== by 0x40054E: main (valgrind-example.c:11)
==12652==
==12652==
==12652== HEAP SUMMARY:
==12652== in use at exit: 40 bytes in 1 blocks
==12652== total heap usage: 1 allocs, 0 frees, 40 bytes allocated
==12652==
==12652== LEAK SUMMARY:
==12652== definitely lost: 40 bytes in 1 blocks
==12652== indirectly lost: 0 bytes in 0 blocks
==12652== possibly lost: 0 bytes in 0 blocks
==12652== still reachable: 0 bytes in 0 blocks
==12652== suppressed: 0 bytes in 0 blocks
==12652== Rerun with --leak-check=full to see details of leaked memory
==12652==
==12652== For counts of detected and suppressed errors, rerun with: -v
==12652== ERROR SUMMARY: 1 errors from 1 contexts (suppressed: 6 from 6)

```

In the output we can see that Valgrind has detected both errors - the off-by-one memory access at line 5 and a memory leak of 40 bytes. If we want a detailed analysis of the memory leak, we need to run Valgrind with `--leak-check=full` option:

```

$ valgrind --leak-check=full ./valgrind-example
==23856== Memcheck, a memory error detector
==23856== Copyright (C) 2002-2010, and GNU GPL'd, by Julian Seward et al.
==23856== Using Valgrind-3.6.0 and LibVEX; rerun with -h for copyright info

```

```

==23856== Command: ./valgrind-example
==23856==
==23856== Invalid write of size 4
==23856== at 0x40067E: f (valgrind-example.c:6)
==23856== by 0x40068E: main (valgrind-example.c:11)
==23856== Address 0x66e7068 is 0 bytes after a block of size 40 alloc'd
==23856== at 0x4C26FDE: malloc (vg_replace_malloc.c:236)
==23856== by 0x400668: f (valgrind-example.c:5)
==23856== by 0x40068E: main (valgrind-example.c:11)
==23856==
==23856==
==23856== HEAP SUMMARY:
==23856== in use at exit: 40 bytes in 1 blocks
==23856== total heap usage: 1 allocs, 0 frees, 40 bytes allocated
==23856==
==23856== 40 bytes in 1 blocks are definitely lost in loss record 1 of 1
==23856== at 0x4C26FDE: malloc (vg_replace_malloc.c:236)
==23856== by 0x400668: f (valgrind-example.c:5)
==23856== by 0x40068E: main (valgrind-example.c:11)
==23856==
==23856== LEAK SUMMARY:
==23856== definitely lost: 40 bytes in 1 blocks
==23856== indirectly lost: 0 bytes in 0 blocks
==23856== possibly lost: 0 bytes in 0 blocks
==23856== still reachable: 0 bytes in 0 blocks
==23856== suppressed: 0 bytes in 0 blocks
==23856==
==23856== For counts of detected and suppressed errors, rerun with: -v
==23856== ERROR SUMMARY: 2 errors from 2 contexts (suppressed: 6 from 6)

```

Now we can see that the memory leak is due to the `malloc()` at line 6.

Usage with MPI

Although Valgrind is not primarily a parallel debugger, it can be used to debug parallel applications as well. When launching your parallel applications, prepend the `valgrind` command. For example:

```
$ mpirun -np 4 valgrind myapplication
```

The default version without MPI support will however report a large number of false errors in the MPI library, such as:

```

==30166== Conditional jump or move depends on uninitialised value(s)
==30166== at 0x4C287E8: strlen (mc_replace_strmem.c:282)
==30166== by 0x55443BD: I_MPI_Processor_model_number (init_interface.c:427)
==30166== by 0x55439E0: I_MPI_Processor_arch_code (init_interface.c:171)
==30166== by 0x558D5AE: MPID_nem_mpi_init_shm_configuration (mpid_nem_mpi_extension.c:100)
==30166== by 0x5598F4C: MPID_nem_init_ckpt (mpid_nem_init.c:566)
==30166== by 0x5598B65: MPID_nem_init (mpid_nem_init.c:489)
==30166== by 0x539BD75: MPIDI_CH3_Init (ch3_init.c:64)
==30166== by 0x5578743: MPID_Init (mpid_init.c:193)
==30166== by 0x554650A: MPIR_Init_thread (initthread.c:539)
==30166== by 0x553369F: PMPI_Init (init.c:195)
==30166== by 0x4008BD: main (valgrind-example-mpi.c:18)

```

so it is better to use the MPI-enabled valgrind from module. The MPI versions requires library:

`$EBROOTVALGRIND/lib/valgrind/libmpiwrap-amd64-linux.so`

which must be included in the `LD_PRELOAD` environment variable.

Lets look at this MPI example:

```
#include <stdlib.h>
#include <mpi.h>

int main(int argc, char *argv[])
{
    int *data = malloc(sizeof(int)*99);

    MPI_Init(&argc, &argv);
    MPI_Bcast(data, 100, MPI_INT, 0, MPI_COMM_WORLD);
    MPI_Finalize();

    return 0;
}
```

There are two errors - use of uninitialized memory and invalid length of the buffer. Lets debug it with valgrind :

```
$ module add intel impi
$ mpiicc -g valgrind-example-mpi.c -o valgrind-example-mpi
$ module add Valgrind/3.11.0-intel-2015b
$ mpirun -np 2 -env LD_PRELOAD $EBROOTVALGRIND/lib/valgrind/libmpiwrap-amd64-linux.so
```

Prints this output : (note that there is output printed for every launched MPI process)

```
==31318== Memcheck, a memory error detector
==31318== Copyright (C) 2002-2013, and GNU GPL'd, by Julian Seward et al.
==31318== Using Valgrind-3.9.0 and LibVEX; rerun with -h for copyright info
==31318== Command: ./valgrind-example-mpi
==31318==
==31319== Memcheck, a memory error detector
==31319== Copyright (C) 2002-2013, and GNU GPL'd, by Julian Seward et al.
==31319== Using Valgrind-3.9.0 and LibVEX; rerun with -h for copyright info
==31319== Command: ./valgrind-example-mpi
==31319==
valgrind MPI wrappers 31319: Active for pid 31319
valgrind MPI wrappers 31319: Try MPIWRAP_DEBUG=help for possible options
valgrind MPI wrappers 31318: Active for pid 31318
valgrind MPI wrappers 31318: Try MPIWRAP_DEBUG=help for possible options
==31319== Unaddressable byte(s) found during client check request
==31319== at 0x4E35974: check_mem_is_addressable_untyped (libmpiwrap.c:960)
==31319== by 0x4E5D0FE: PMPI_Bcast (libmpiwrap.c:908)
==31319== by 0x400911: main (valgrind-example-mpi.c:20)
==31319== Address 0x69291cc is 0 bytes after a block of size 396 alloc'd
==31319== at 0x4C27AAA: malloc (vg_replace_malloc.c:291)
==31319== by 0x4007BC: main (valgrind-example-mpi.c:8)
==31319==
==31318== Uninitialised byte(s) found during client check request
==31318== at 0x4E3591D: check_mem_is_defined_untyped (libmpiwrap.c:952)
```

```

==31318== by 0x4E5D06D: PMPI_Bcast (libmpiwrap.c:908)
==31318== by 0x400911: main (valgrind-example-mpi.c:20)
==31318== Address 0x6929040 is 0 bytes inside a block of size 396 alloc'd
==31318== at 0x4C27AAA: malloc (vg_replace_malloc.c:291)
==31318== by 0x4007BC: main (valgrind-example-mpi.c:8)
==31318==
==31318== Unaddressable byte(s) found during client check request
==31318== at 0x4E3591D: check_mem_is_defined_untyped (libmpiwrap.c:952)
==31318== by 0x4E5D06D: PMPI_Bcast (libmpiwrap.c:908)
==31318== by 0x400911: main (valgrind-example-mpi.c:20)
==31318== Address 0x69291cc is 0 bytes after a block of size 396 alloc'd
==31318== at 0x4C27AAA: malloc (vg_replace_malloc.c:291)
==31318== by 0x4007BC: main (valgrind-example-mpi.c:8)
==31318==
==31318==
==31318== HEAP SUMMARY:
==31318== in use at exit: 3,172 bytes in 67 blocks
==31318== total heap usage: 191 allocs, 124 frees, 81,203 bytes allocated
==31318==
==31319==
==31319== HEAP SUMMARY:
==31319== in use at exit: 3,172 bytes in 67 blocks
==31319== total heap usage: 175 allocs, 108 frees, 48,435 bytes allocated
==31319==
==31318== LEAK SUMMARY:
==31318== definitely lost: 408 bytes in 3 blocks
==31318== indirectly lost: 256 bytes in 1 blocks
==31318== possibly lost: 0 bytes in 0 blocks
==31318== still reachable: 2,508 bytes in 63 blocks
==31318== suppressed: 0 bytes in 0 blocks
==31318== Rerun with --leak-check=full to see details of leaked memory
==31318==
==31318== For counts of detected and suppressed errors, rerun with: -v
==31318== Use --track-origins=yes to see where uninitialised values come from
==31318== ERROR SUMMARY: 2 errors from 2 contexts (suppressed: 4 from 4)
==31319== LEAK SUMMARY:
==31319== definitely lost: 408 bytes in 3 blocks
==31319== indirectly lost: 256 bytes in 1 blocks
==31319== possibly lost: 0 bytes in 0 blocks
==31319== still reachable: 2,508 bytes in 63 blocks
==31319== suppressed: 0 bytes in 0 blocks
==31319== Rerun with --leak-check=full to see details of leaked memory
==31319==
==31319== For counts of detected and suppressed errors, rerun with: -v
==31319== ERROR SUMMARY: 1 errors from 1 contexts (suppressed: 4 from 4)

```

We can see that Valgrind has reported use of uninitialised memory on the master process (which reads the array to be broadcasted) and use of unaddressable memory on both processes.

Intel VTune Amplifier XE

Introduction

Intel® VTune™ Amplifier, part of Intel Parallel studio, is a GUI profiling tool designed for Intel processors. It offers a graphical performance analysis of single core and multithreaded applications. A highlight of the features:

- Hotspot analysis
- Locks and waits analysis
- Low level specific counters, such as branch analysis and memory bandwidth
- Power usage analysis - frequency and sleep states.

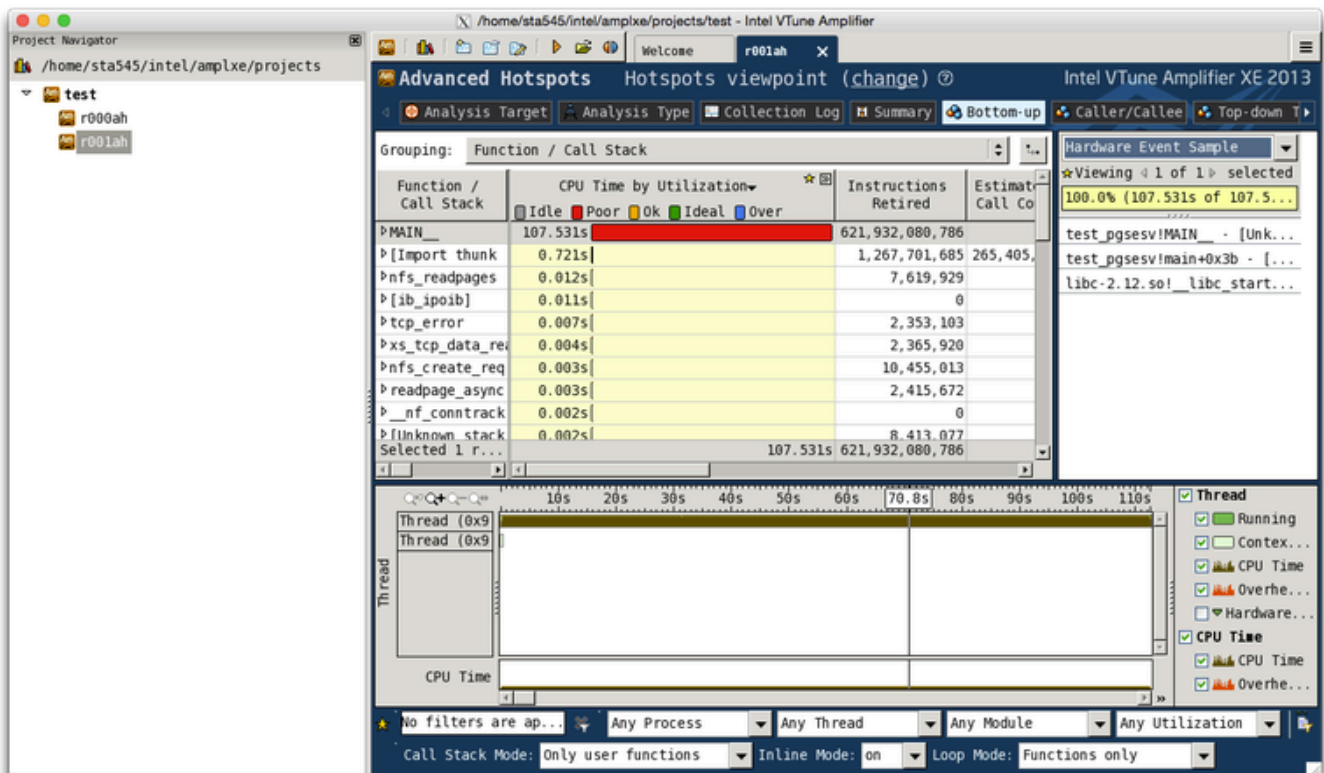


Figure 1:

Usage

To profile an application with VTune Amplifier, special kernel modules need to be loaded. The modules are not loaded on the login nodes, thus direct profiling on login nodes is not possible. By default, the kernel modules are not loaded on compute nodes neither. In order to have the modules loaded, you need to specify vtune=version PBS resource at job submit. The version is the same as for environment module. For example to use VTune/2016_update1:

```
$ qsub -q qexp -A OPEN-0-0 -I -l select=1,vtune=2016_update1
```

After that, you can verify the modules sep*, pax and vtsspp are present in the kernel :

```
$ lsmod | grep -e sep -e pax -e vtsspp
vtsspp 362000 0
sep3_15 546657 0
pax 4312 0
```

To launch the GUI, first load the module:

```
$ module add VTune/2016_update1
```

and launch the GUI :

```
$ amplxe-gui
```

The GUI will open in new window. Click on “New Project...” to create a new project. After clicking OK, a new window with project properties will appear. At “Application:”, select the path to your binary you want to profile (the binary should be compiled with -g flag). Some additional options such as command line arguments can be selected. At “Managed code profiling mode:” select “Native” (unless you want to profile managed mode .NET/Mono applications). After clicking OK, your project is created.

To run a new analysis, click “New analysis...”. You will see a list of possible analysis. Some of them will not be possible on the current CPU (eg. Intel Atom analysis is not possible on Sandy bridge CPU), the GUI will show an error box if you select the wrong analysis. For example, select “Advanced Hotspots”. Clicking on Start will start profiling of the application.

Remote Analysis

VTune Amplifier also allows a form of remote analysis. In this mode, data for analysis is collected from the command line without GUI, and the results are then loaded to GUI on another machine. This allows profiling without interactive graphical jobs. To perform a remote analysis, launch a GUI somewhere, open the new analysis window and then click the button “Command line” in bottom right corner. It will show the command line needed to perform the selected analysis.

The command line will look like this:

```
/apps/all/VTune/2016_update1/vtune_amplifier_xe_2016.1.1.434111/bin64/amplxe-cl -col
```

Copy the line to clipboard and then you can paste it in your jobscript or in command line. After the collection is run, open the GUI once again, click the menu button in the upper right corner, and select “Open > Result...”. The GUI will load the results from the run.

Xeon Phi

It is possible to analyze both native and offloaded Xeon Phi applications.

Native mode

This mode is useful for native Xeon Phi applications launched directly on the card. In *Analysis Target* window, select *Intel Xeon Phi coprocessor (native)*, choose path to the binary and MIC card to run on.

Offload mode

This mode is useful for applications that are launched from the host and use offload, OpenCL or mpirun. In *Analysis Target* window, select *Intel Xeon Phi coprocessor (native)*, choose path to the binary and MIC card to run on.

!!! Note “Note” If the analysis is interrupted or aborted, further analysis on the card might be impossible and you will get errors like “ERROR connecting to MIC card”. In this case please contact our support to reboot the MIC card.

You may also use remote analysis to collect data from the MIC and then analyze it in the GUI later :

Native launch:




```
$ /apps/all/VTune/2016_update1/vtune_amplifier_xe_2016.1.1.434111/bin64/amplxe-cl -t
```

Host launch:

```
$ /apps/all/VTune/2016_update1/vtune_amplifier_xe_2016.1.1.434111/bin64/amplxe-cl -t
```

You can obtain this command line by pressing the “Command line...” button on Analysis Type screen.

References

1. <https://www.rcac.purdue.edu/tutorials/phi/PerformanceTuningXeonPhi-Tullos.pdf>  Performance Tuning for Intel® Xeon Phi™ Coprocessors
2. <https://software.intel.com/en-us/intel-vtune-amplifier-xe-support/documentation>  >Intel® VTune™ Amplifier Support
3. https://software.intel.com/en-us/amplifier_help_linux 

Total View

TotalView is a GUI-based source code multi-process, multi-thread debugger.

License and Limitations for cluster Users

On the cluster users can debug OpenMP or MPI code that runs up to 64 parallel processes. These limitation means that:

- 1 user can debug up 64 processes, or
- 32 users can debug 2 processes, etc.

Debugging of GPU accelerated codes is also supported.

You can check the status of the licenses here [🔗](#).

Compiling Code to run with TotalView

Modules

Load all necessary modules to compile the code. For example:

```
module load intel
```

```
module load impi    ... or ... module load OpenMPI/X.X.X-icc
```

Load the TotalView module:

```
module load TotalView/8.15.4-6-linux-x86-64
```

Compile the code:

```
mpicc -g -O0 -o test_debug test.c
```

```
mpif90 -g -O0 -o test_debug test.f
```

Compiler flags

Before debugging, you need to compile your code with theses flags:

!!! Note “Note” **-g** : Generates extra debugging information usable by GDB. -g3 includes even more debugging information. This option is available for GNU and INTEL C/C++ and Fortran compilers.

**** -O0 **** : Suppress all optimizations.

Starting a Job with TotalView

Be sure to log in with an X window forwarding enabled. This could mean using the -X in the ssh:

```
ssh -X username@salomon.it4i.cz
```


Other options is to access login node using VNC. Please see the detailed information on how to use graphic user interface on Anselm.

From the login node an interactive session with X windows forwarding (-X option) can be started by following command:

```
qsub -I -X -A NONE-0-0 -q qexp -lselect=1:ncpus=24:mpiprocs=24,walltime=01:00:00
```

Then launch the debugger with the totalview command followed by the name of the executable to debug.

Debugging a serial code

To debug a serial code use:

```
totalview test_debug
```

Debugging a parallel code - option 1

To debug a parallel code compiled with **OpenMPI** you need to setup your TotalView environment:

!!! Note “Note” **Please note:** To be able to run parallel debugging procedure from the command line without stopping the debugger in the mpiexec source code you have to add the following function to your ~/.tvdrc file:

```
proc mpi_auto_run_starter {loaded_id} {
    set starter_programs {mpirun mpiexec orterun}
    set executable_name [TV::symbol get $loaded_id full_pathname]
    set file_component [file tail $executable_name]

    if {[lsearch -exact $starter_programs $file_component] != -1} {
        puts "*****"
        puts "Automatically starting $file_component"
        puts "*****"
        dgo
    }
}

# Append this function to TotalView's image load callbacks so that
# TotalView run this program automatically.

dlappend TV::image_load_callbacks mpi_auto_run_starter
```

The source code of this function can be also found in

```
/apps/all/OpenMPI/1.10.1-GNU-4.9.3-2.25/etc/openmpi-totalview.tcl
```

You can also add only following line to you ~/.tvdrc file instead of the entire function:

```
source /apps/all/OpenMPI/1.10.1-GNU-4.9.3-2.25/etc/openmpi-totalview.tcl
```

You need to do this step only once. See also OpenMPI FAQ entry [🔗](#)

Now you can run the parallel debugger using:

```
mpirun -tv -n 5 ./test_debug
```

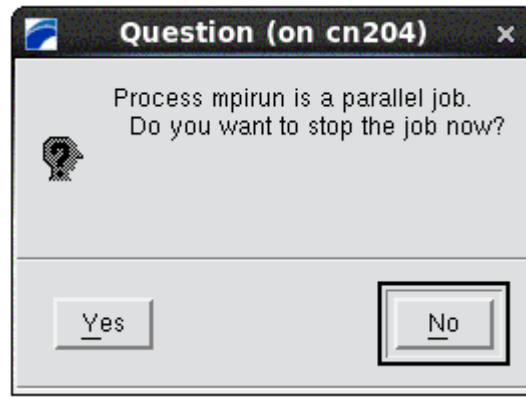


Figure 1:

When following dialog appears click on “Yes”

At this point the main TotalView GUI window will appear and you can insert the breakpoints and start debugging:

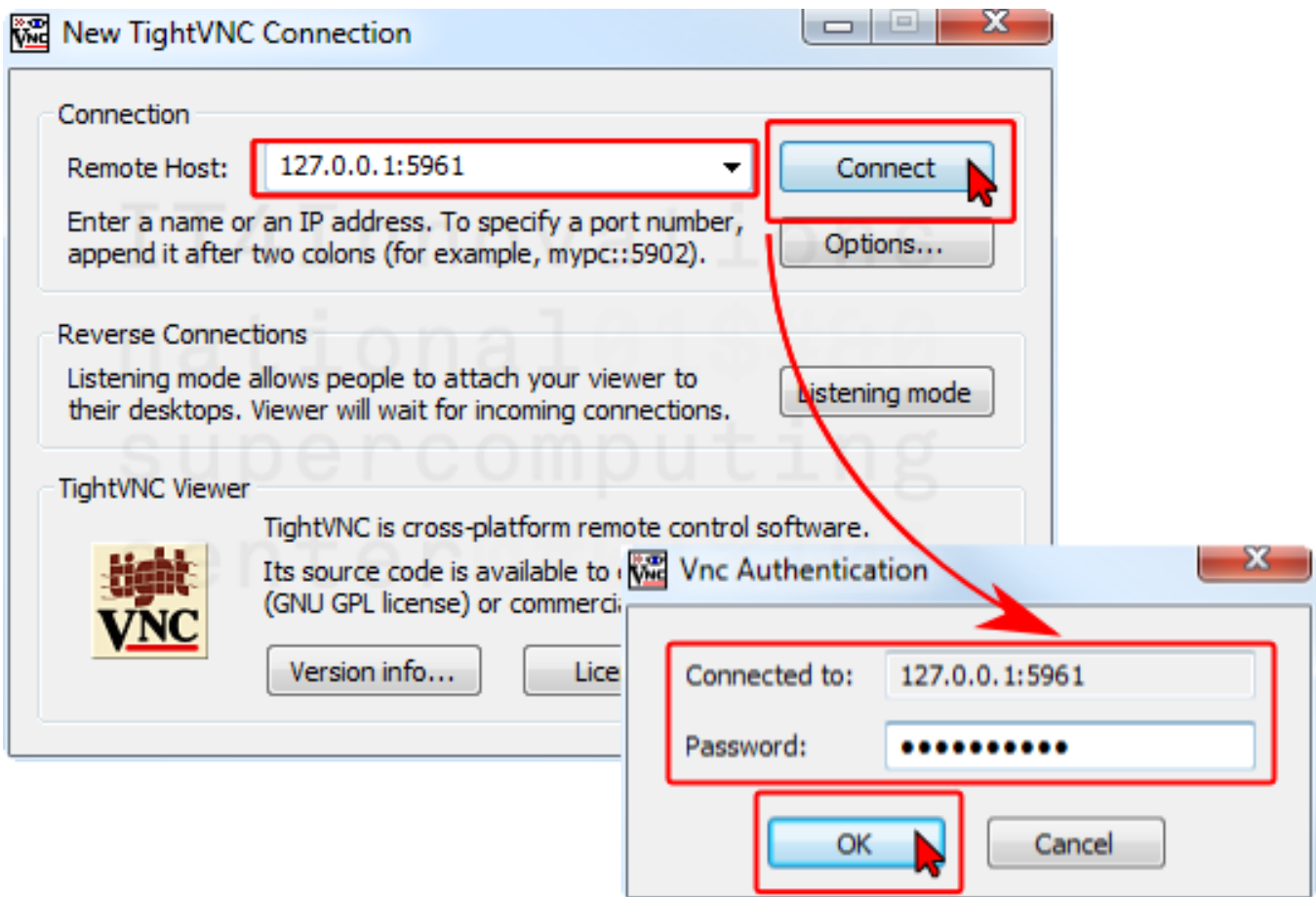


Figure 2:

Debugging a parallel code - option 2

Other option to start new parallel debugging session from a command line is to let TotalView to execute mpirun by itself. In this case user has to specify a MPI implementation used to compile the source code.

The following example shows how to start debugging session with Intel MPI:

```
module load intel/2015b-intel-2015b impi/5.0.3.048-iccifort-2015.3.187-GNU-5.1.0-2.25
```

```
totalview -mpi "Intel MPI-Hydra" -np 8 ./hello_debug impi
```

After running previous command you will see the same window as shown in the screenshot above.

More information regarding the command line parameters of the TotalView can be found TotalView Reference Guide, Chapter 7: TotalView Command Syntax.

Documentation

[1] The TotalView documentation [web page](#) is a good resource for learning more about some of the advanced TotalView features.

Phono3py

Introduction

This GPL software calculates phonon-phonon interactions via the third order force constants. It allows to obtain lattice thermal conductivity, phonon lifetime/linewidth, imaginary part of self energy at the lowest order, joint density of states (JDOS) and weighted-JDOS. For details see Phys. Rev. B 91, 094306 (2015) and <http://atztego.github.io/phono3py/index.html>

!!! Note “Note” Load the phono3py/0.9.14-ictce-7.3.5-Python-2.7.9 module

```
$ module load phono3py/0.9.14-ictce-7.3.5-Python-2.7.9
```

Example of calculating thermal conductivity of Si using VASP code.

Calculating force constants

One needs to calculate second order and third order force constants using the diamond structure of silicon stored in POSCAR (the same form as in VASP) using single displacement calculations within supercell.

```
$ cat POSCAR
Si
1.0
5.4335600309153529 0.0000000000000000 0.0000000000000000
0.0000000000000000 5.4335600309153529 0.0000000000000000
0.0000000000000000 0.0000000000000000 5.4335600309153529
Si
8
Direct
0.8750000000000000 0.8750000000000000 0.8750000000000000
0.8750000000000000 0.3750000000000000 0.3750000000000000
0.3750000000000000 0.8750000000000000 0.3750000000000000
0.3750000000000000 0.3750000000000000 0.8750000000000000
0.1250000000000000 0.1250000000000000 0.1250000000000000
0.1250000000000000 0.6250000000000000 0.6250000000000000
0.6250000000000000 0.1250000000000000 0.6250000000000000
0.6250000000000000 0.6250000000000000 0.1250000000000000
```

Generating displacement using 2x2x2 supercell for both second and third order force constants

```
$ phono3py -d --dim="2 2 2" -c POSCAR
```

111 displacements is created stored in disp_fc3.yaml, and the structure input files with this displacements are POSCAR-00XXX, where the XXX=111.

disp_fc3.yaml	POSCAR-00008	POSCAR-00017	POSCAR-00026	POSCAR-00035	POSCAR-00044	POSCAR-00053
POSCAR	POSCAR-00009	POSCAR-00018	POSCAR-00027	POSCAR-00036	POSCAR-00045	POSCAR-00054
POSCAR-00001	POSCAR-00010	POSCAR-00019	POSCAR-00028	POSCAR-00037	POSCAR-00046	POSCAR-00055
POSCAR-00002	POSCAR-00011	POSCAR-00020	POSCAR-00029	POSCAR-00038	POSCAR-00047	POSCAR-00056
POSCAR-00003	POSCAR-00012	POSCAR-00021	POSCAR-00030	POSCAR-00039	POSCAR-00048	POSCAR-00057
POSCAR-00004	POSCAR-00013	POSCAR-00022	POSCAR-00031	POSCAR-00040	POSCAR-00049	POSCAR-00058

POSCAR-00005	POSCAR-00014	POSCAR-00023	POSCAR-00032	POSCAR-00041	POSCAR-00050	POSCAR-00059
POSCAR-00006	POSCAR-00015	POSCAR-00024	POSCAR-00033	POSCAR-00042	POSCAR-00051	POSCAR-00060
POSCAR-00007	POSCAR-00016	POSCAR-00025	POSCAR-00034	POSCAR-00043	POSCAR-00052	POSCAR-00061

For each displacement the forces needs to be calculated, i.e. in form of the output file of VASP (vasprun.xml). For a single VASP calculations one needs KPOINTS, POTCAR, INCAR in your case directory (where you have POSCARS) and those 111 displacements calculations can be generated by prepare.sh script. Then each of the single 111 calculations is submitted run.sh by submit.sh.

```
$. /prepare.sh
```

```
$ls
```

disp-00001	disp-00009	disp-00017	disp-00025	disp-00033	disp-00041	disp-00049	disp-00057
disp-00002	disp-00010	disp-00018	disp-00026	disp-00034	disp-00042	disp-00050	disp-00058
disp-00003	disp-00011	disp-00019	disp-00027	disp-00035	disp-00043	disp-00051	disp-00059
disp-00004	disp-00012	disp-00020	disp-00028	disp-00036	disp-00044	disp-00052	disp-00060
disp-00005	disp-00013	disp-00021	disp-00029	disp-00037	disp-00045	disp-00053	disp-00061
disp-00006	disp-00014	disp-00022	disp-00030	disp-00038	disp-00046	disp-00054	disp-00062
disp-00007	disp-00015	disp-00023	disp-00031	disp-00039	disp-00047	disp-00055	disp-00063
disp-00008	disp-00016	disp-00024	disp-00032	disp-00040	disp-00048	disp-00056	disp-00064

Taylor your run.sh script to fit into your project and other needs and submit all 111 calculations using submit.sh script

```
$ ./submit.sh
```

Collecting results and post-processing with phono3py

Once all jobs are finished and vasprun.xml is created in each disp-XXXXXX directory the collection is done by

```
$ phono3py --cf3 disp-{00001..00111}/vasprun.xml
```

and disp_fc2.yaml, FORCES_FC2, FORCES_FC3 and disp_fc3.yaml should appear and put into the hdf format by

```
$ phono3py --dim="2 2 2" -c POSCAR
```

resulting in fc2.hdf5 and fc3.hdf5

Thermal conductivity

The phonon lifetime calculations takes some time, however is independent on grid points, so could be splitted:

```
$ phono3py --fc3 --fc2 --dim="2 2 2" --mesh="9 9 9" --sigma 0.1 --wgp
```

Inspecting ir_grid_points.yaml

```
$ grep grid_point ir_grid_points.yaml
num_reduced_ir_grid_points: 35
ir_grid_points: # [address, weight]
- grid_point: 0
- grid_point: 1
- grid_point: 2
- grid_point: 3
```

- grid_point: 4
- grid_point: 10
- grid_point: 11
- grid_point: 12
- grid_point: 13
- grid_point: 20
- grid_point: 21
- grid_point: 22
- grid_point: 30
- grid_point: 31
- grid_point: 40
- grid_point: 91
- grid_point: 92
- grid_point: 93
- grid_point: 94
- grid_point: 101
- grid_point: 102
- grid_point: 103
- grid_point: 111
- grid_point: 112
- grid_point: 121
- grid_point: 182
- grid_point: 183
- grid_point: 184
- grid_point: 192
- grid_point: 193
- grid_point: 202
- grid_point: 273
- grid_point: 274
- grid_point: 283
- grid_point: 364

one finds which grid points needed to be calculated, for instance using following

```
$ phono3py --fc3 --fc2 --dim="2 2 2" --mesh="9 9 9" -c POSCAR --sigma 0.1 --br --write-g
```

one calculates grid points 0, 1, 2. To automatize one can use for instance scripts to submit 5 points in series, see gofree-cond1.sh

```
$ qsub gofree-cond1.sh
```

Finally the thermal conductivity result is produced by grouping single conductivity per grid calculations using

```
$ phono3py --fc3 --fc2 --dim="2 2 2" --mesh="9 9 9" --br --read_gamma
```

Molpro

Molpro is a complete system of ab initio programs for molecular electronic structure calculations.

About Molpro

Molpro is a software package used for accurate ab-initio quantum chemistry calculations. More information can be found at the official webpage[\[1\]](#).

License

Molpro software package is available only to users that have a valid license. Please contact support to enable access to Molpro if you have a valid license appropriate for running on our cluster (eg. academic research group licence, parallel execution).

To run Molpro, you need to have a valid license token present in " \$HOME/.molpro/token". You can download the token from Molpro website[\[2\]](#).

Installed version

Currently on Anselm is installed version 2010.1, patch level 45, parallel version compiled with Intel compilers and Intel MPI.

Compilation parameters are default:

Parameter	Value
max number of atoms	200
max number of valence orbitals	300
max number of basis functions	4095
max number of states per symmetry	20
max number of state symmetries	16
max number of records	200
max number of primitives	maxbfm x [2]

Running

Molpro is compiled for parallel execution using MPI and OpenMP. By default, Molpro reads the number of allocated nodes from PBS and launches a data server on one node. On the remaining allocated nodes, compute processes are launched, one process per node, each with 16 threads. You can modify this behavior by using -n, -t and helper-server options. Please refer to the Molpro documentation[\[3\]](#) for more details.

!!! Note "Note" The OpenMP parallelization in Molpro is limited and has been observed to produce limited scaling. We therefore recommend to use MPI parallelization only. This can be achieved by passing option mpirprocs=16:ompthreads=1 to PBS.

You are advised to use the -d option to point to a directory in SCRATCH filesystem. Molpro can produce a large amount of temporary data during its run, and it is important that these are placed in the fast scratch filesystem.

Example jobscript

```
#PBS -A IT4I-0-0
#PBS -q qprod
#PBS -l select=1:ncpus=16:mpiprocs=16:ompthreads=1

cd $PBS_O_WORKDIR

# load Molpro module
module add molpro

# create a directory in the SCRATCH filesystem
mkdir -p /scratch/$USER/$PBS_JOBID

# copy an example input
cp /apps/chem/molpro/2010.1/molprop_2010_1_Linux_x86_64_i8/examples/caffeine_opt_diis

# run Molpro with default options
molpro -d /scratch/$USER/$PBS_JOBID caffeine_opt_diis.com

# delete scratch directory
rm -rf /scratch/$USER/$PBS_JOBID
```


NWChem

High-Performance Computational Chemistry

Introduction

NWChem aims to provide its users with computational chemistry tools that are scalable both in their ability to treat large scientific computational chemistry problems efficiently, and in their use of available parallel computing resources from high-performance parallel supercomputers to conventional workstation clusters.

Homepage 

Installed versions

The following versions are currently installed:

- NWChem/6.3.revision2-2013-10-17-Python-2.7.8, current release. Compiled with Intel compilers, MKL and Intel MPI
- NWChem/6.5.revision26243-intel-2015b-2014-09-10-Python-2.7.8

For a current list of installed versions, execute:

```
module avail NWChem
```

The recommend to use version 6.5. Version 6.3 fails on Salomon nodes with accelerator, because it attempts to communicate over scif0 interface. In 6.5 this is avoided by setting `ARMCI_OPENIB_DEVICE=mlx4_0`, this setting is included in the module.


Running

NWChem is compiled for parallel MPI execution. Normal procedure for MPI jobs applies. Sample jobscript :

```
#PBS -A IT4I-0-0
#PBS -q qprod
#PBS -l select=1:ncpus=24:mpiprocs=24

cd $PBS_O_WORKDIR
module add NWChem/6.5.revision26243-intel-2015b-2014-09-10-Python-2.7.8
mpirun nwchem h2o.nw
```

Options

Please refer to the documentation  and in the input file set the following directives :

- MEMORY : controls the amount of memory NWChem will use
- SCRATCH_DIR : set this to a directory in SCRATCH filesystem (or run the calculation completely in a scratch directory). For certain calculations, it might be advisable to reduce I/O by forcing “direct” mode, eg. “scf direct”

Intel Xeon Phi

A guide to Intel Xeon Phi usage

Intel Xeon Phi can be programmed in several modes. The default mode on Anselm is offload mode, but all modes described in this document are supported.

Intel Utilities for Xeon Phi

To get access to a compute node with Intel Xeon Phi accelerator, use the PBS interactive session

```
$ qsub -I -q qmic -A NONE-0-0
```

To set up the environment module “Intel” has to be loaded

```
$ module load intel/13.5.192
```

Information about the hardware can be obtained by running the micinfo program on the host.

```
$ /usr/bin/micinfo
```

The output of the “micinfo” utility executed on one of the Anselm node is as follows. (note: to get PCIe related details the command has to be run with root privileges)

```
MicInfo Utility Log
```

```
Created Mon Jul 22 00:23:50 2013
```

System Info

```
HOST OS           : Linux
OS Version        : 2.6.32-279.5.2.el6.Bull.33.x86_64
Driver Version    : 6720-15
MPSS Version      : 2.1.6720-15
Host Physical Memory : 98843 MB
```

```
Device No: 0, Device Name: mic0
```

Version

```
Flash Version      : 2.1.03.0386
SMC Firmware Version : 1.15.4830
SMC Boot Loader Version : 1.8.4326
uOS Version        : 2.6.38.8-g2593b11
Device Serial Number : ADKC30102482
```

Board

```
Vendor ID          : 0x8086
Device ID          : 0x2250
Subsystem ID       : 0x2500
Coproprocessor Stepping ID : 3
PCIe Width         : x16
PCIe Speed         : 5 GT/s
PCIe Max payload size : 256 bytes
PCIe Max read req size : 512 bytes
Coproprocessor Model : 0x01
```

```

Coprocessor Model Ext      : 0x00
Coprocessor Type          : 0x00
Coprocessor Family        : 0x0b
Coprocessor Family Ext    : 0x00
Coprocessor Stepping      : B1
Board SKU                 : B1PRQ-5110P/5120D
ECC Mode                  : Enabled
SMC HW Revision           : Product 225W Passive CS

```

Cores

```

Total No of Active Cores : 60
Voltage                  : 1032000 uV
Frequency                : 1052631 kHz

```

Thermal

```

Fan Speed Control       : N/A
Fan RPM                 : N/A
Fan PWM                 : N/A
Die Temp                : 49 C

```

GDDR

```

GDDR Vendor             : Elpida
GDDR Version            : 0x1
GDDR Density            : 2048 Mb
GDDR Size               : 7936 MB
GDDR Technology         : GDDR5
GDDR Speed              : 5.000000 GT/s
GDDR Frequency          : 2500000 kHz
GDDR Voltage            : 1501000 uV

```

Offload Mode

To compile a code for Intel Xeon Phi a MPSS stack has to be installed on the machine where compilation is executed. Currently the MPSS stack is only installed on compute nodes equipped with accelerators.

```

$ qsub -I -q qmic -A NONE-0-0
$ module load intel/13.5.192

```

For debugging purposes it is also recommended to set environment variable “OFFLOAD_REPORT”. Value can be set from 0 to 3, where higher number means more debugging information.

```
export OFFLOAD_REPORT=3
```

A very basic example of code that employs offload programming technique is shown in the next listing. Please note that this code is sequential and utilizes only single core of the accelerator.

```

$ vim source-offload.cpp

#include <iostream>

int main(int argc, char* argv[])
{
    const int niter = 100000;

```

```

    double result = 0;

    #pragma offload target(mic)
    for (int i = 0; i < niter; ++i) {
        const double t = (i + 0.5) / niter;
        result += 4.0 / (t * t + 1.0);
    }
    result /= niter;
    std::cout << "Pi ~ " << result << '\n';
}

```

To compile a code using Intel compiler run

```
$ icc source-offload.cpp -o bin-offload
```

To execute the code, run the following command on the host

```
./bin-offload
```

Parallelization in Offload Mode Using OpenMP

One way of parallelization a code for Xeon Phi is using OpenMP directives. The following example shows code for parallel vector addition.

```

$ vim ./vect-add

#include <stdio.h>

typedef int T;

#define SIZE 1000

#pragma offload_attribute(push, target(mic))
T in1[SIZE];
T in2[SIZE];
T res[SIZE];
#pragma offload_attribute(pop)

// MIC function to add two vectors
__attribute__((target(mic))) add_mic(T *a, T *b, T *c, int size) {
    int i = 0;
    #pragma omp parallel for
    for (i = 0; i < size; i++)
        c[i] = a[i] + b[i];
}

// CPU function to add two vectors
void add_cpu (T *a, T *b, T *c, int size) {
    int i;
    for (i = 0; i < size; i++)
        c[i] = a[i] + b[i];
}

// CPU function to generate a vector of random numbers

```

```

void random_T (T *a, int size) {
    int i;
    for (i = 0; i < size; i++)
        a[i] = rand() % 10000; // random number between 0 and 9999
}

// CPU function to compare two vectors
int compare(T *a, T *b, T size ){
    int pass = 0;
    int i;
    for (i = 0; i < size; i++){
        if (a[i] != b[i]) {
            printf("Value mismatch at location %d, values %d and %dn",i, a[i], b[i]);
            pass = 1;
        }
    }
    if (pass == 0) printf ("Test passedn"); else printf ("Test Failedn");
    return pass;
}

int main()
{
    int i;
    random_T(in1, SIZE);
    random_T(in2, SIZE);

    #pragma offload target(mic) in(in1,in2) inout(res)
    {

        // Parallel loop from main function
        #pragma omp parallel for
        for (i=0; i<SIZE; i++)
            res[i] = in1[i] + in2[i];

        // or parallel loop is called inside the function
        add_mic(in1, in2, res, SIZE);

    }

    //Check the results with CPU implementation
    T res_cpu[SIZE];
    add_cpu(in1, in2, res_cpu, SIZE);
    compare(res, res_cpu, SIZE);
}

```

During the compilation Intel compiler shows which loops have been vectorized in both host and accelerator. This can be enabled with compiler option “-vec-report2”. To compile and execute the code run

```
$ icc vect-add.c -openmp_report2 -vec-report2 -o vect-add
```

```
$ ./vect-add
```

Some interesting compiler flags useful not only for code debugging are:

!!! Note “Note” Debugging `openmp_report[0|1|2]` - controls the compiler based vectorization diagnostic level
`vec-report[0|1|2]` - controls the OpenMP parallelizer diagnostic level

Performance optimization

`xhost` - FOR HOST ONLY - to generate AVX (Advanced Vector Extensions) instructions.

Automatic Offload using Intel MKL Library

Intel MKL includes an Automatic Offload (AO) feature that enables computationally intensive MKL functions called in user code to benefit from attached Intel Xeon Phi coprocessors automatically and transparently.

Behavioral of automatic offload mode is controlled by functions called within the program or by environmental variables. Complete list of controls is listed [here](#).

The Automatic Offload may be enabled by either an MKL function call within the code:

```
mkl_mic_enable();
```

or by setting environment variable

```
$ export MKL_MIC_ENABLE=1
```

To get more information about automatic offload please refer to “Using Intel® MKL Automatic Offload on Intel® Xeon Phi™ Coprocessors” [white paper](#) or Intel MKL [documentation](#).

Automatic offload example

At first get an interactive PBS session on a node with MIC accelerator and load “intel” module that automatically loads “mkl” module as well.

```
$ qsub -I -q qmic -A OPEN-0-0 -l select=1:ncpus=16  
$ module load intel
```

Following example show how to automatically offload an SGEMM (single precision - general matrix multiply) function to MIC coprocessor. The code can be copied to a file and compiled without any necessary modification.

```
$ vim sgemm-ao-short.c  
  
#include <stdio.h>  
#include <stdlib.h>  
#include <malloc.h>  
#include <stdint.h>  
  
#include "mkl.h"  
  
int main(int argc, char **argv)  
{  
    float *A, *B, *C; /* Matrices */  
  
    MKL_INT N = 2560; /* Matrix dimensions */  
    MKL_INT LD = N; /* Leading dimension */  
    int matrix_bytes; /* Matrix size in bytes */
```

```

int matrix_elements; /* Matrix size in elements */

float alpha = 1.0, beta = 1.0; /* Scaling factors */
char transa = 'N', transb = 'N'; /* Transposition options */

int i, j; /* Counters */

matrix_elements = N * N;
matrix_bytes = sizeof(float) * matrix_elements;

/* Allocate the matrices */
A = malloc(matrix_bytes); B = malloc(matrix_bytes); C = malloc(matrix_bytes);

/* Initialize the matrices */
for (i = 0; i < matrix_elements; i++) {
    A[i] = 1.0; B[i] = 2.0; C[i] = 0.0;
}

printf("Computing SGEMM on the host\n");
sgemm(&transa, &transb, &N, &N, &N, &alpha, A, &N, B, &N, &beta, C, &N);

printf("Enabling Automatic Offload\n");
/* Alternatively, set environment variable MKL_MIC_ENABLE=1 */
mkl_mic_enable();

int ndevices = mkl_mic_get_device_count(); /* Number of MIC devices */
printf("Automatic Offload enabled: %d MIC devices present\n", ndevices);

printf("Computing SGEMM with automatic workdivision\n");
sgemm(&transa, &transb, &N, &N, &N, &alpha, A, &N, B, &N, &beta, C, &N);

/* Free the matrix memory */
free(A); free(B); free(C);

printf("Done\n");

return 0;
}

```

!!! Note “Note” Please note: This example is simplified version of an example from MKL. The expanded version can be found here: [\\$MKL_EXAMPLES/mic_ao/blas/source/sgemm.c](#)

To compile a code using Intel compiler use:

```
$ icc -mkl sgemm-ao-short.c -o sgemm
```

For debugging purposes enable the offload report to see more information about automatic offloading.

```
$ export OFFLOAD_REPORT=2
```

The output of a code should look similar to following listing, where lines starting with [MKL] are generated by offload reporting:

```

Computing SGEMM on the host
Enabling Automatic Offload

```

```

Automatic Offload enabled: 1 MIC devices present
Computing SGEMM with automatic workdivision
[MKL] [MIC --] [AO Function]      SGEMM
[MKL] [MIC --] [AO SGEMM Workdivision]  0.00 1.00
[MKL] [MIC 00] [AO SGEMM CPU Time]      0.463351 seconds
[MKL] [MIC 00] [AO SGEMM MIC Time]      0.179608 seconds
[MKL] [MIC 00] [AO SGEMM CPU->MIC Data] 52428800 bytes
[MKL] [MIC 00] [AO SGEMM MIC->CPU Data] 26214400 bytes
Done

```

Native Mode

In the native mode a program is executed directly on Intel Xeon Phi without involvement of the host machine. Similarly to offload mode, the code is compiled on the host computer with Intel compilers.

To compile a code user has to be connected to a compute with MIC and load Intel compilers module. To get an interactive session on a compute node with an Intel Xeon Phi and load the module use following commands:

```
$ qsub -I -q qmic -A NONE-0-0
```

```
$ module load intel/13.5.192
```

!!! Note “Note” Please note that particular version of the Intel module is specified. This information is used later to specify the correct library paths.

To produce a binary compatible with Intel Xeon Phi architecture user has to specify “-mmic” compiler flag. Two compilation examples are shown below. The first example shows how to compile OpenMP parallel code “vect-add.c” for host only:

```
$ icc -xhost -no-offload -fopenmp vect-add.c -o vect-add-host
```

To run this code on host, use:

```
$ ./vect-add-host
```

The second example shows how to compile the same code for Intel Xeon Phi:

```
$ icc -mmic -fopenmp vect-add.c -o vect-add-mic
```

Execution of the Program in Native Mode on Intel Xeon Phi

The user access to the Intel Xeon Phi is through the SSH. Since user home directories are mounted using NFS on the accelerator, users do not have to copy binary files or libraries between the host and accelerator.

To connect to the accelerator run:

```
$ ssh mic0
```

If the code is sequential, it can be executed directly:

```
mic0 $ ~/path_to_binary/vect-add-seq-mic
```

If the code is parallelized using OpenMP a set of additional libraries is required for execution. To locate these libraries new path has to be added to the LD_LIBRARY_PATH environment variable prior to the execution:


```
mic0 $ export LD_LIBRARY_PATH=/apps/intel/composer_xe_2013.5.192/compiler/lib/mic:$LD
```

!!! Note “Note” Please note that the path exported in the previous example contains path to a specific compiler (here the version is 5.192). This version number has to match with the version number of the Intel compiler module that was used to compile the code on the host computer.

For your information the list of libraries and their location required for execution of an OpenMP parallel code on Intel Xeon Phi is:

!!! Note “Note” /apps/intel/composer_xe_2013.5.192/compiler/lib/mic

- libiomp5.so
- libimf.so
- libsvml.so
- libirng.so
- libintlc.so.5

Finally, to run the compiled code use:

```
$ ~/path_to_binary/vect-add-mic
```

OpenCL

OpenCL (Open Computing Language) is an open standard for general-purpose parallel programming for diverse mix of multi-core CPUs, GPU coprocessors, and other parallel processors. OpenCL provides a flexible execution model and uniform programming environment for software developers to write portable code for systems running on both the CPU and graphics processors or accelerators like the Intel® Xeon Phi.

On Anselm OpenCL is installed only on compute nodes with MIC accelerator, therefore OpenCL code can be compiled only on these nodes.

```
module load openccl-sdk openccl-rt
```

Always load “openccl-sdk” (providing devel files like headers) and “openccl-rt” (providing dynamic library libOpenCL.so) modules to compile and link OpenCL code. Load “openccl-rt” for running your compiled code.

There are two basic examples of OpenCL code in the following directory:

```
/apps/intel/openccl-examples/
```

First example “CapsBasic” detects OpenCL compatible hardware, here CPU and MIC, and prints basic information about the capabilities of it.

```
/apps/intel/openccl-examples/CapsBasic/capsbasic
```

To compile and run the example copy it to your home directory, get a PBS interactive session on of the nodes with MIC and run make for compilation. Make files are very basic and shows how the OpenCL code can be compiled on Anselm.

```
$ cp /apps/intel/openccl-examples/CapsBasic/* .  
$ qsub -I -q qmic -A NONE-0-0  
$ make
```

The compilation command for this example is:

```
$ g++ capsbasic.cpp -lOpenCL -o capsbasic -I/apps/intel/openccl/include/
```

After executing the compiled binary file, following output should be displayed.

```
./capsbasic
```

```
Number of available platforms: 1
```

```
Platform names:
```

```
[0] Intel(R) OpenCL [Selected]
```

```
Number of devices available for each type:
```

```
CL_DEVICE_TYPE_CPU: 1
```

```
CL_DEVICE_TYPE_GPU: 0
```

```
CL_DEVICE_TYPE_ACCELERATOR: 1
```

```
** Detailed information for each device ***
```

```
CL_DEVICE_TYPE_CPU[0]
```

```
CL_DEVICE_NAME: Intel(R) Xeon(R) CPU E5-2470 0 @ 2.30GHz
```

```
CL_DEVICE_AVAILABLE: 1
```

```
...
```

```
CL_DEVICE_TYPE_ACCELERATOR[0]
```

```
CL_DEVICE_NAME: Intel(R) Many Integrated Core Acceleration Card
```

```
CL_DEVICE_AVAILABLE: 1
```

```
...
```

!!! Note “Note” More information about this example can be found on Intel website: <http://software.intel.com/en-us/vcsource/samples/caps-basic/>

The second example that can be found in “/apps/intel/opencl-examples” directory is General Matrix Multiply. You can follow the the same procedure to download the example to your directory and compile it.

```
$ cp -r /apps/intel/opencl-examples/* .
```

```
$ qsub -I -q qmic -A NONE-0-0
```

```
$ cd GEMM
```

```
$ make
```

The compilation command for this example is:

```
$ g++ cmdoptions.cpp gemm.cpp ../common/basic.cpp ../common/cmdparser.cpp ../common/o
```

To see the performance of Intel Xeon Phi performing the DGEMM run the example as follows:

```
./gemm -d 1
```

```
Platforms (1):
```

```
[0] Intel(R) OpenCL [Selected]
```

```
Devices (2):
```

```
[0] Intel(R) Xeon(R) CPU E5-2470 0 @ 2.30GHz
```

```
[1] Intel(R) Many Integrated Core Acceleration Card [Selected]
```

```
Build program options: "-DT=float -DTILE_SIZE_M=1 -DTILE_GROUP_M=16 -DTILE_SIZE_N=128
```

```
Running gemm_nn kernel with matrix size: 3968x3968
```

```
Memory row stride to ensure necessary alignment: 15872 bytes
```

```
Size of memory region for one matrix: 62980096 bytes
```

```
Using alpha = 0.57599 and beta = 0.872412
```

```
...
```

```
Host time: 0.292953 sec.
```

```
Host perf: 426.635 GFLOPS
```

```
Host time: 0.293334 sec.
Host perf: 426.081 GFLOPS
...
```

!!! Note “Note” Please note: GNU compiler is used to compile the OpenCL codes for Intel MIC. You do not need to load Intel compiler module.

MPI

Environment setup and compilation

Again an MPI code for Intel Xeon Phi has to be compiled on a compute node with accelerator and MPSS software stack installed. To get to a compute node with accelerator use:

```
$ qsub -I -q qmic -A NONE-0-0
```

The only supported implementation of MPI standard for Intel Xeon Phi is Intel MPI. To setup a fully functional development environment a combination of Intel compiler and Intel MPI has to be used. On a host load following modules before compilation:

```
$ module load intel/13.5.192 impi/4.1.1.036
```

To compile an MPI code for host use:

```
$ mpiicc -xhost -o mpi-test mpi-test.c
```

To compile the same code for Intel Xeon Phi architecture use:

```
$ mpiicc -mmic -o mpi-test-mic mpi-test.c
```

An example of basic MPI version of “hello-world” example in C language, that can be executed on both host and Xeon Phi is (can be directly copy and pasted to a .c file)

```
#include <stdio.h>
#include <mpi.h>

int main (argc, argv)
{
    int argc;
    char *argv[];

    int rank, size;

    int len;
    char node[MPI_MAX_PROCESSOR_NAME];

    MPI_Init (&argc, &argv);          /* starts MPI */
    MPI_Comm_rank (MPI_COMM_WORLD, &rank);    /* get current process id */
    MPI_Comm_size (MPI_COMM_WORLD, &size);    /* get number of processes */

    MPI_Get_processor_name(node,&len);

    printf( "Hello world from process %d of %d on host %s\n", rank, size, node );
    MPI_Finalize();
    return 0;
}
```

MPI programming models

Intel MPI for the Xeon Phi coprocessors offers different MPI programming models:

!!! Note “Note” **Host-only model** - all MPI ranks reside on the host. The coprocessors can be used by using offload pragmas. (Using MPI calls inside offloaded code is not supported.)

****Coprocesor-only model**** - all MPI ranks reside only on the coprocessors.

****Symmetric model**** - the MPI ranks reside on both the host and the coprocessor. Most general M

Host-only model

In this case all environment variables are set by modules, so to execute the compiled MPI program on a single node, use:

```
$ mpirun -np 4 ./mpi-test
```

The output should be similar to:

```
Hello world from process 1 of 4 on host cn207
Hello world from process 3 of 4 on host cn207
Hello world from process 2 of 4 on host cn207
Hello world from process 0 of 4 on host cn207
```

Coprocesor-only model

There are two ways how to execute an MPI code on a single coprocessor: 1.) lunch the program using “**mpirun**” from the coprocessor; or 2.) lunch the task using “**mpiexec.hydra**” from a host.

Execution on coprocessor

Similarly to execution of OpenMP programs in native mode, since the environmental module are not supported on MIC, user has to setup paths to Intel MPI libraries and binaries manually. One time setup can be done by creating a “**.profile**” file in user’s home directory. This file sets up the environment on the MIC automatically once user access to the accelerator through the SSH.

```
$ vim ~/.profile
```

```
PS1='[u@h W]$ '
```

```
export PATH=/usr/bin:/usr/sbin:/bin:/sbin
```

```
#OpenMP
```

```
export LD_LIBRARY_PATH=/apps/intel/composer_xe_2013.5.192/compiler/lib/mic:$LD_LIBRARY_PATH
```

```
#Intel MPI
```

```
export LD_LIBRARY_PATH=/apps/intel/impi/4.1.1.036/mic/lib/:$LD_LIBRARY_PATH
```

```
export PATH=/apps/intel/impi/4.1.1.036/mic/bin/:$PATH
```

!!! Note “Note” Please note:

- this file sets up both environmental variable for both MPI and OpenMP libraries.
- this file sets up the paths to a particular version of Intel MPI library and particular versi

To access a MIC accelerator located on a node that user is currently connected to, use:

```
$ ssh mic0
```

or in case you need specify a MIC accelerator on a particular node, use:

```
$ ssh cn207-mic0
```

To run the MPI code in parallel on multiple core of the accelerator, use:

```
$ mpirun -np 4 ./mpi-test-mic
```

The output should be similar to:

```
Hello world from process 1 of 4 on host cn207-mic0
Hello world from process 2 of 4 on host cn207-mic0
Hello world from process 3 of 4 on host cn207-mic0
Hello world from process 0 of 4 on host cn207-mic0
```

Execution on host

If the MPI program is launched from host instead of the coprocessor, the environmental variables are not set using the “profile” file. Therefore user has to specify library paths from the command line when calling “mpiexec”.

First step is to tell mpiexec that the MPI should be executed on a local accelerator by setting up the environmental variable “I_MPI_MIC”

```
$ export I_MPI_MIC=1
```

Now the MPI program can be executed as:

```
$ mpiexec.hydra -genv LD_LIBRARY_PATH /apps/intel/impi/4.1.1.036/mic/lib/ -host mic0
```

or using mpirun

```
$ mpirun -genv LD_LIBRARY_PATH /apps/intel/impi/4.1.1.036/mic/lib/ -host mic0 -n 4 ~/
```

!!! Note “Note” Please note: - the full path to the binary has to specified (here: “>~/mpi-test-mic”) - the LD_LIBRARY_PATH has to match with Intel MPI module used to compile the MPI code

The output should be again similar to:

```
Hello world from process 1 of 4 on host cn207-mic0
Hello world from process 2 of 4 on host cn207-mic0
Hello world from process 3 of 4 on host cn207-mic0
Hello world from process 0 of 4 on host cn207-mic0
```

!!! Note “Note” Please note that the “mpiexec.hydra” requires a file the MIC filesystem. If the file is missing please contact the system administrators. A simple test to see if the file is present is to execute:

```
$ ssh mic0 ls /bin/pmi_proxy
/bin/pmi_proxy
```

Execution on host - MPI processes distributed over multiple accelerators on multiple nodes

To get access to multiple nodes with MIC accelerator, user has to use PBS to allocate the resources. To start interactive session, that allocates 2 compute nodes = 2 MIC accelerators run qsub command with following parameters:

```
$ qsub -I -q qmic -A NONE-0-0 -l select=2:ncpus=16
```

```
$ module load intel/13.5.192 impi/4.1.1.036
```

This command connects user through ssh to one of the nodes immediately. To see the other nodes that have been allocated use:

```
$ cat $PBS_NODEFILE
```

For example:

```
cn204.bullx
cn205.bullx
```

This output means that the PBS allocated nodes cn204 and cn205, which means that user has direct access to “cn204-mic0” and “cn-205-mic0” accelerators.

!!! Note “Note” Please note: At this point user can connect to any of the allocated nodes or any of the allocated MIC accelerators using ssh:

- to connect to the second node : `** $ ssh cn205**`
- to connect to the accelerator on the first node from the first node: `**$ ssh cn204-mic0**` or
- to connect to the accelerator on the second node from the first node: `**$ ssh cn205-mic0**`

At this point we expect that correct modules are loaded and binary is compiled. For parallel execution the mpiexec.hydra is used. Again the first step is to tell mpiexec that the MPI can be executed on MIC accelerators by setting up the environmental variable “I_MPI_MIC”

```
$ export I_MPI_MIC=1
```

The launch the MPI program use:

```
$ mpiexec.hydra -genv LD_LIBRARY_PATH /apps/intel/impi/4.1.1.036/mic/lib/
-genv I_MPI_FABRICS_LIST tcp
-genv I_MPI_FABRICS shm:tcp
-genv I_MPI_TCP_NETMASK=10.1.0.0/16
-host cn204-mic0 -n 4 ~/mpi-test-mic
: -host cn205-mic0 -n 6 ~/mpi-test-mic
```

or using mpirun:

```
$ mpirun -genv LD_LIBRARY_PATH /apps/intel/impi/4.1.1.036/mic/lib/
-genv I_MPI_FABRICS_LIST tcp
-genv I_MPI_FABRICS shm:tcp
-genv I_MPI_TCP_NETMASK=10.1.0.0/16
-host cn204-mic0 -n 4 ~/mpi-test-mic
: -host cn205-mic0 -n 6 ~/mpi-test-mic
```

In this case four MPI processes are executed on accelerator cn204-mic and six processes are executed on accelerator cn205-mic0. The sample output (sorted after execution) is:

```
Hello world from process 0 of 10 on host cn204-mic0
Hello world from process 1 of 10 on host cn204-mic0
Hello world from process 2 of 10 on host cn204-mic0
Hello world from process 3 of 10 on host cn204-mic0
Hello world from process 4 of 10 on host cn205-mic0
Hello world from process 5 of 10 on host cn205-mic0
Hello world from process 6 of 10 on host cn205-mic0
Hello world from process 7 of 10 on host cn205-mic0
Hello world from process 8 of 10 on host cn205-mic0
Hello world from process 9 of 10 on host cn205-mic0
```

The same way MPI program can be executed on multiple hosts:

```
$ mpiexec.hydra -genv LD_LIBRARY_PATH /apps/intel/impi/4.1.1.036/mic/lib/
-genv I_MPI_FABRICS_LIST tcp
-genv I_MPI_FABRICS shm:tcp
-genv I_MPI_TCP_NETMASK=10.1.0.0/16
-host cn204 -n 4 ~/mpi-test
: -host cn205 -n 6 ~/mpi-test
```

Symmetric model

In a symmetric mode MPI programs are executed on both host computer(s) and MIC accelerator(s). Since MIC has a different architecture and requires different binary file produced by the Intel compiler two different files has to be compiled before MPI program is executed.

In the previous section we have compiled two binary files, one for hosts “**mpi-test**” and one for MIC accelerators “**mpi-test-mic**”. These two binaries can be executed at once using mpiexec.hydra:

```
$ mpiexec.hydra
-genv I_MPI_FABRICS_LIST tcp
-genv I_MPI_FABRICS shm:tcp
-genv I_MPI_TCP_NETMASK=10.1.0.0/16
-genv LD_LIBRARY_PATH /apps/intel/impi/4.1.1.036/mic/lib/
-host cn205 -n 2 ~/mpi-test
: -host cn205-mic0 -n 2 ~/mpi-test-mic
```

In this example the first two parameters (line 2 and 3) sets up required environment variables for execution. The third line specifies binary that is executed on host (here cn205) and the last line specifies the binary that is execute on the accelerator (here cn205-mic0).

The output of the program is:

```
Hello world from process 0 of 4 on host cn205
Hello world from process 1 of 4 on host cn205
Hello world from process 2 of 4 on host cn205-mic0
Hello world from process 3 of 4 on host cn205-mic0
```

The execution procedure can be simplified by using the mpirun command with the machine file a a parameter. Machine file contains list of all nodes and accelerators that should used to execute MPI processes.

An example of a machine file that uses 2 >hosts (**cn205** and **cn206**) and 2 accelerators (**cn205-mic0** and **cn206-mic0**) to run 2 MPI processes on each of them:

```
$ cat hosts_file_mix
cn205:2
cn205-mic0:2
cn206:2
cn206-mic0:2
```

In addition if a naming convention is set in a way that the name of the binary for host is “**bin_name**” and the name of the binary for the accelerator is “**bin_name-mic**” then by setting up the environment variable **I_MPI_MIC_POSTFIX** to “**-mic**” user do not have to specify the names of booth binaries. In this case mpirun needs just the name of the host binary file (i.e. “mpi-test”) and uses the suffix to get a name of the binary for accelerator (i.e. “mpi-test-mic”).

```
$ export I_MPI_MIC_POSTFIX=-mic
```

To run the MPI code using mpirun and the machine file “hosts_file_mix” use:

```
$ mpirun
  -genv I_MPI_FABRICS shm:tcp
  -genv LD_LIBRARY_PATH /apps/intel/impi/4.1.1.036/mic/lib/
  -genv I_MPI_FABRICS_LIST tcp
  -genv I_MPI_FABRICS shm:tcp
  -genv I_MPI_TCP_NETMASK=10.1.0.0/16
  -machinefile hosts_file_mix
  ~/mpi-test
```

A possible output of the MPI “hello-world” example executed on two hosts and two accelerators is:

```
Hello world from process 0 of 8 on host cn204
Hello world from process 1 of 8 on host cn204
Hello world from process 2 of 8 on host cn204-mic0
Hello world from process 3 of 8 on host cn204-mic0
Hello world from process 4 of 8 on host cn205
Hello world from process 5 of 8 on host cn205
Hello world from process 6 of 8 on host cn205-mic0
Hello world from process 7 of 8 on host cn205-mic0
```

!!! Note “Note” Please note: At this point the MPI communication between MIC accelerators on different nodes uses 1Gb Ethernet only.

Using the PBS automatically generated node-files

PBS also generates a set of node-files that can be used instead of manually creating a new one every time. Three node-files are generated:

!!! Note “Note” **Host only node-file:**

- /lscratch/\${PBS_JOBID}/nodefile-cn MIC only node-file:
- /lscratch/\${PBS_JOBID}/nodefile-mic Host and MIC node-file:
- /lscratch/\${PBS_JOBID}/nodefile-mix

Please note each host or accelerator is listed only per files. User has to specify how many jobs should be executed per node using “-n” parameter of the mpirun command.

Optimization

For more details about optimization techniques please read Intel document Optimization and Performance Tuning for Intel® Xeon Phi™ Coprocessors [\[1\]](#)

Environment and Modules

Environment Customization

After logging in, you may want to configure the environment. Write your preferred path definitions, aliases, functions and module loads in the `.bashrc` file

```
# ./bashrc

# Source global definitions
if [ -f /etc/bashrc ]; then
    . /etc/bashrc
fi

# User specific aliases and functions
alias qs='qstat -a'
module load intel/2015b

# Display informations to standard output - only in interactive ssh session
if [ -n "$SSH_TTY" ]
then
    module list # Display loaded modules
fi
```

!!! Note “Note” Do not run commands outputting to standard output (echo, module list, etc) in `.bashrc` for non-interactive SSH sessions. It breaks fundamental functionality (scp, PBS) of your account! Take care for SSH session interactivity for such commands as stated in the previous example.

Application Modules

In order to configure your shell for running particular application on Salomon we use Module package interface.

Application modules on Salomon cluster are built using EasyBuild¹. The modules are divided into the following structure:

```
base: Default module class
bio: Bioinformatics, biology and biomedical
cae: Computer Aided Engineering (incl. CFD)
chem: Chemistry, Computational Chemistry and Quantum Chemistry
compiler: Compilers
data: Data management & processing tools
debugger: Debuggers
devel: Development tools
geo: Earth Sciences
ide: Integrated Development Environments (e.g. editors)
lang: Languages and programming aids
lib: General purpose libraries
math: High-level mathematical software
mpi: MPI stacks
numlib: Numerical Libraries
perf: Performance tools
```

phys: Physics and physical systems simulations
system: System utilities (e.g. highly depending on system OS and hardware)
toolchain: EasyBuild toolchains
tools: General purpose tools
vis: Visualization, plotting, documentation and typesetting

!!! Note “Note” The modules set up the application paths, library paths and environment variables for running particular application.

The modules may be loaded, unloaded and switched, according to momentary needs.

To check available modules use

```
$ module avail
```

To load a module, for example the OpenMPI module use

```
$ module load OpenMPI
```

loading the OpenMPI module will set up paths and environment variables of your active shell such that you are ready to run the OpenMPI software

To check loaded modules use

```
$ module list
```

To unload a module, for example the OpenMPI module use

```
$ module unload OpenMPI
```

Learn more on modules by reading the module man page

```
$ man module
```

EasyBuild Toolchains

As we wrote earlier, we are using EasyBuild for automatised software installation and module creation.

EasyBuild employs so-called **compiler toolchains** or, simply toolchains for short, which are a major concept in handling the build and installation processes.

A typical toolchain consists of one or more compilers, usually put together with some libraries for specific functionality, e.g., for using an MPI stack for distributed computing, or which provide optimized routines for commonly used math operations, e.g., the well-known BLAS/LAPACK APIs for linear algebra routines.

For each software package being built, the toolchain to be used must be specified in some way.

The EasyBuild framework prepares the build environment for the different toolchain components, by loading their respective modules and defining environment variables to specify compiler commands (e.g., via `$F90`), compiler and linker options (e.g., via `$CFLAGS` and `$LDFLAGS`), the list of library names to supply to the linker (via `$LIBS`), etc. This enables making easyblocks largely toolchain-agnostic since they can simply rely on these environment variables; that is, unless they need to be aware of, for example, the particular compiler being used to determine the build configuration options.

Recent releases of EasyBuild include out-of-the-box toolchain support for:

- various compilers, including GCC, Intel, Clang, CUDA
- common MPI libraries, such as Intel MPI, MPICH, MVAPICH2, OpenMPI

- various numerical libraries, including ATLAS, Intel MKL, OpenBLAS, ScalaPACK, FFTW

On Salomon, we have currently following toolchains installed:

Toolchain	Module(s)
GCC	GCC
ictce	icc, ifort, imkl, impi
intel	GCC, icc, ifort, imkl, impi
gomp	GCC, OpenMPI
golf	BLACS, FFTW, GCC, OpenBLAS, OpenMPI, ScaLAPACK
iomp	OpenMPI, icc, ifort
iccifort	icc, ifort

7D Enhanced Hypercube

More about Job submission - Placement by IB switch / Hypercube dimension.

Nodes may be selected via the PBS resource attribute ehc_[1-7]d .

Hypercube	dimension
1D	ehc_1d
2D	ehc_2d
3D	ehc_3d
4D	ehc_4d
5D	ehc_5d
6D	ehc_6d
7D	ehc_7d

Schematic representation of the Salomon cluster IB single-plain topology represents hypercube dimension 0.

7D Enhanced Hypercube

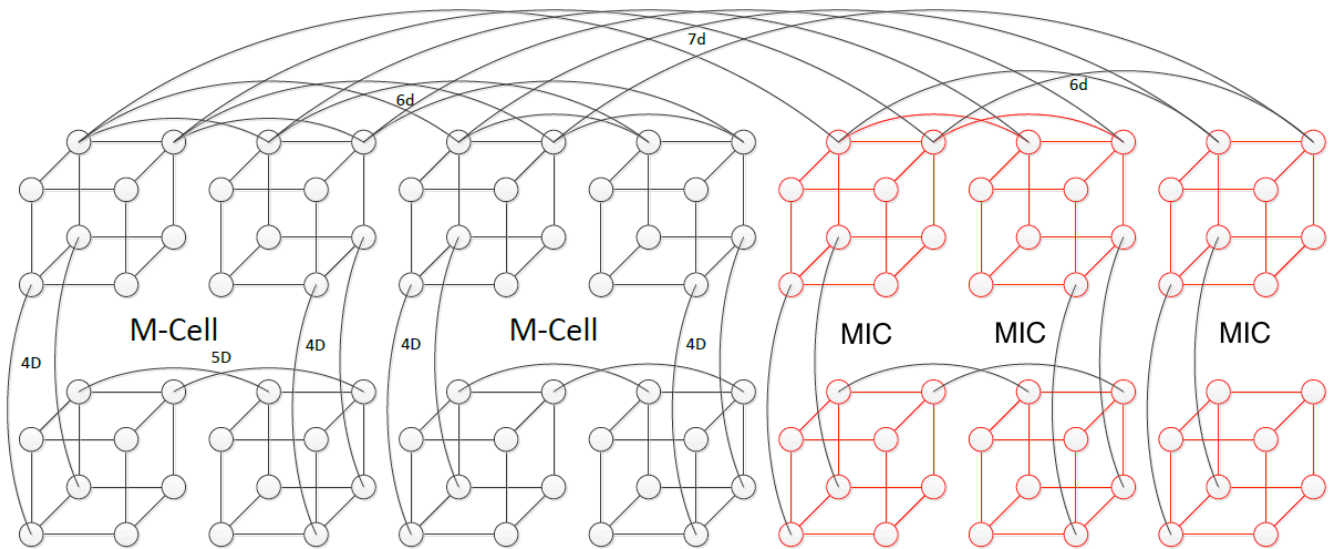


Figure 1:

Node type	Count
M-Cell compute nodes w/o ac- cel- era- tor	576
compute nodes MIC ac- cel- er- ated	432

IB Topology

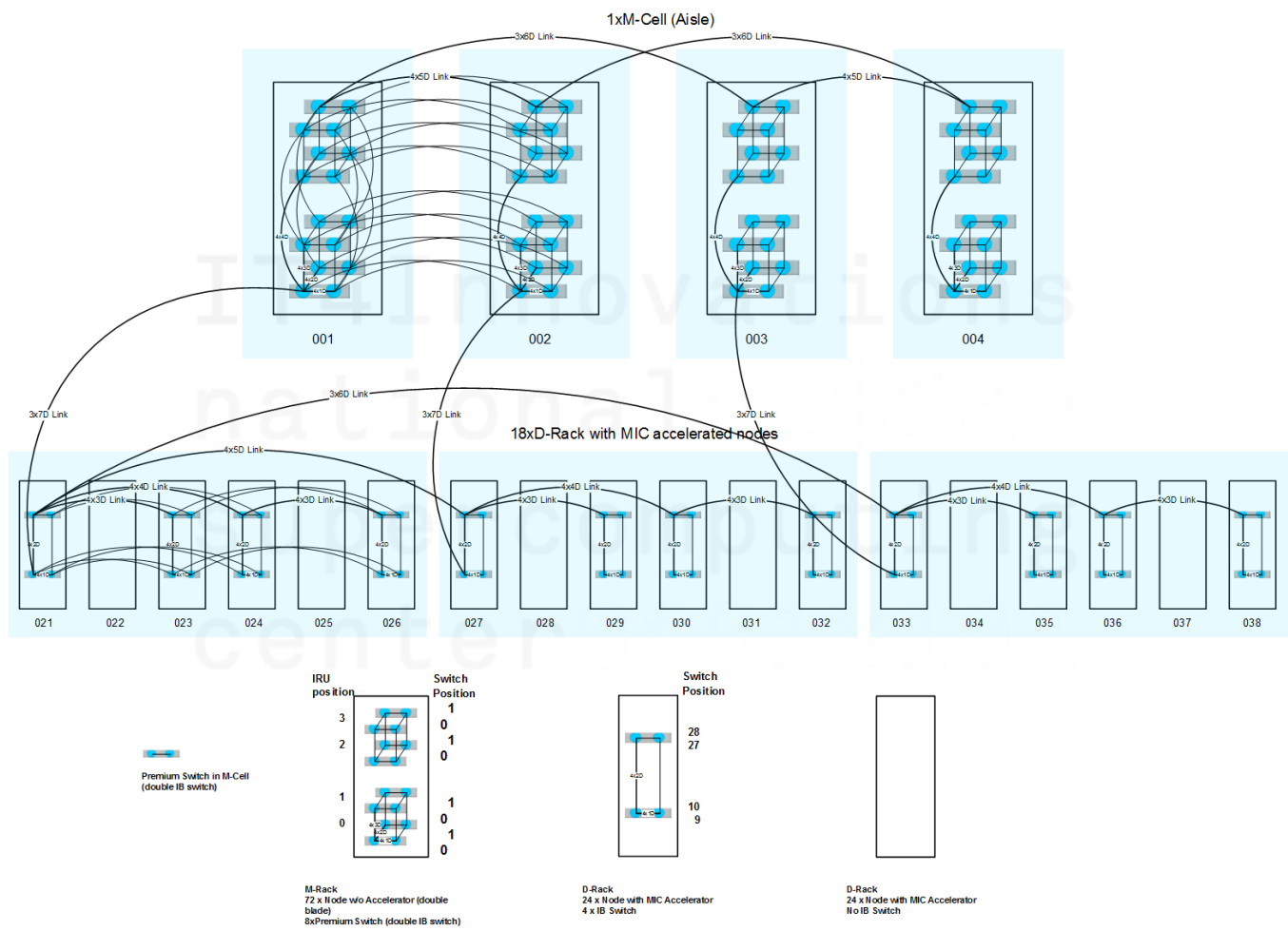


Figure 2:

IB single-plane topology

A complete M-Cell assembly consists of four compute racks. Each rack contains 4x physical IRUs - Independent rack units. Using one dual socket node per one blade slot leads to 8 logical IRUs. Each rack contains 4x2 SGI ICE X IB Premium Blades.

The SGI ICE X IB Premium Blade provides the first level of interconnection via dual 36-port Mellanox FDR InfiniBand ASIC switch with connections as follows:

- 9 ports from each switch chip connect to the unified backplane, to connect the 18 compute node slots
- 3 ports on each chip provide connectivity between the chips
- 24 ports from each switch chip connect to the external bulkhead, for a total of 48

IB single-plane topology - ICEX Mcell

Each colour in each physical IRU represents one dual-switch ASIC switch.

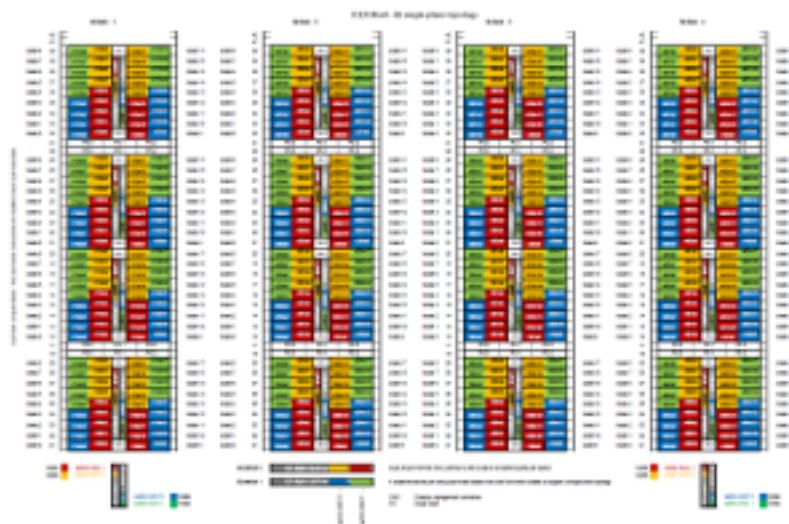
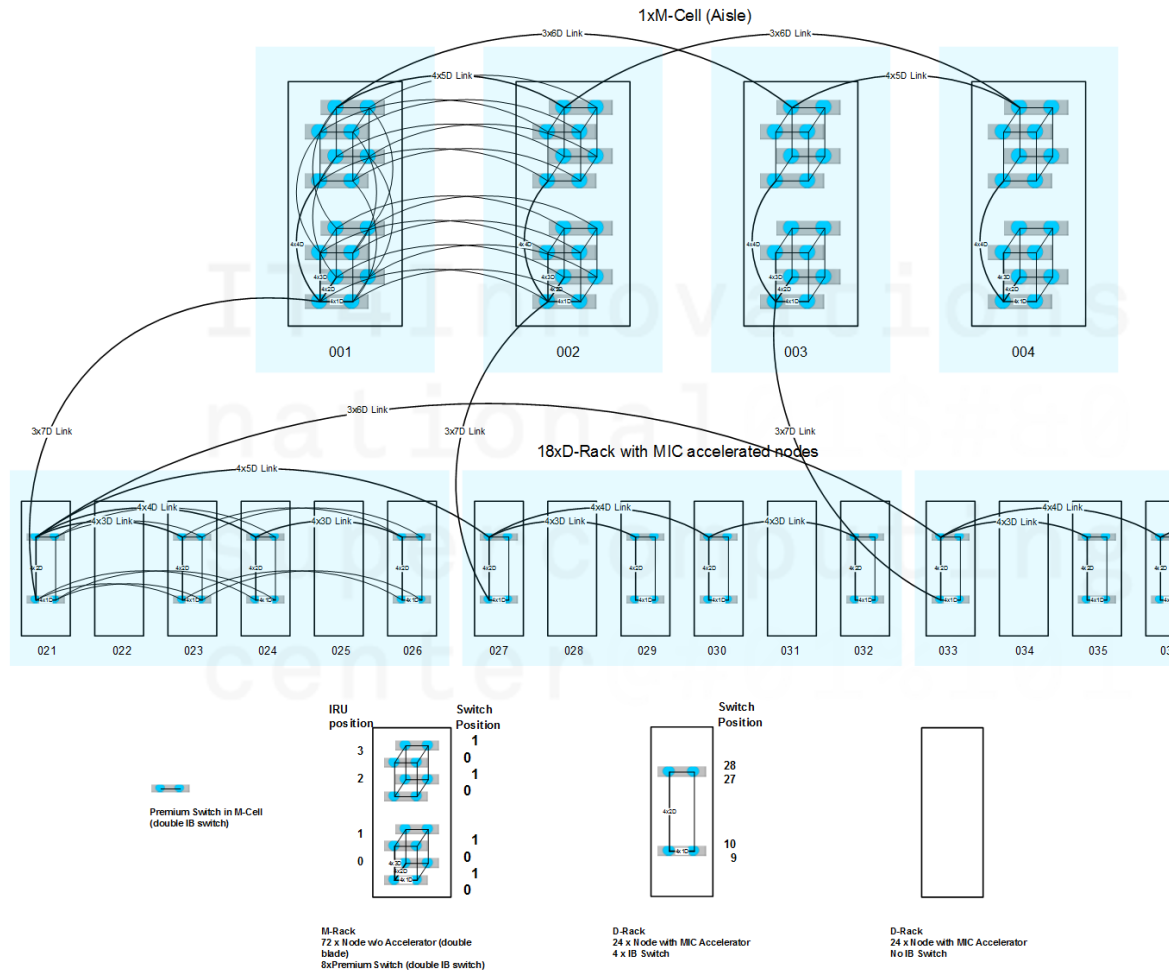


Figure 1:

IB single-plane topology - Accelerated nodes

Each of the 3 inter-connected D racks are equivalent to one half of Mcell rack. 18x D rack with MIC accelerated nodes [r21-r38] are equivalent to 3 Mcell racks as shown in a diagram 7D Enhanced Hypercube.



As shown in a diagram

- Racks 21, 22, 23, 24, 25, 26 are equivalent to one Mcell rack.
- Racks 27, 28, 29, 30, 31, 32 are equivalent to one Mcell rack.
- Racks 33, 34, 35, 36, 37, 38 are equivalent to one Mcell rack.

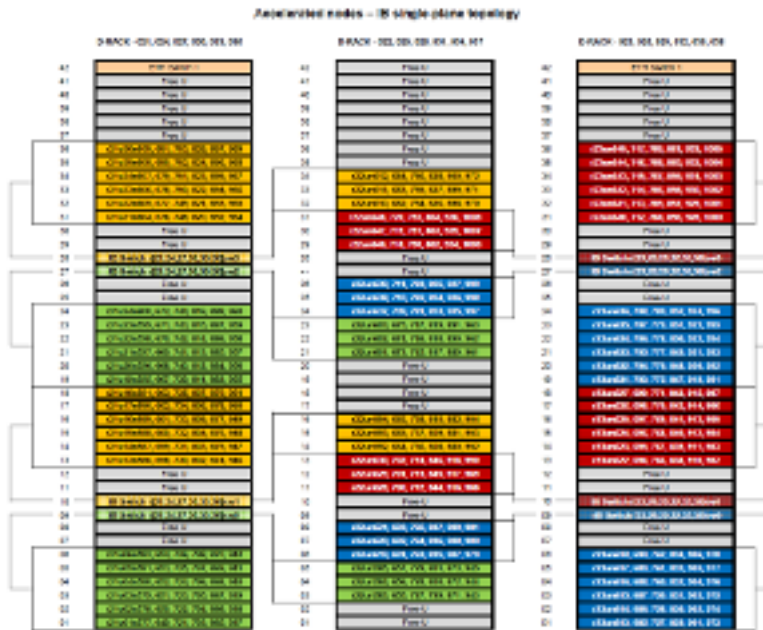


Figure 2:

Network

All compute and login nodes of Salomon are interconnected by 7D Enhanced hypercube Infiniband network and by Gigabit Ethernet network. Only Infiniband network may be used to transfer user data.

Infiniband Network

All compute and login nodes of Salomon are interconnected by 7D Enhanced hypercube Infiniband network (56 Gbps). The network topology is a 7D Enhanced hypercube.

Read more about schematic representation of the Salomon cluster IB single-plain topology (hypercube dimension 0).

The compute nodes may be accessed via the Infiniband network using ib0 network interface, in address range 10.17.0.0 (mask 255.255.224.0). The MPI may be used to establish native Infiniband connection among the nodes.

The network provides **2170MB/s** transfer rates via the TCP connection (single stream) and up to **3600MB/s** via native Infiniband protocol.

Example

```
$ qsub -q qexp -l select=4:ncpus=16 -N Name0 ./myjob
$ qstat -n -u username
```

Job ID	Username	Queue	Jobname	SessID	NDS	TSK	Memory	Req'd Time	Req'd S	Elap Time
15209.isrv5	username	qexp	Name0	5530	4	96	--	01:00	R	00:00
r4i1n0/0*24+r4i1n1/0*24+r4i1n2/0*24+r4i1n3/0*24										

In this example, we access the node r4i1n0 by Infiniband network via the ib0 interface.

```
$ ssh 10.17.35.19
```

In this example, we get information of the Infiniband network.

```
$ ifconfig
....
inet addr:10.17.35.19....
....
```

```
$ ip addr show ib0
```

```
....
inet 10.17.35.19....
....
```

Resource Allocation and Job Execution

To run a job, computational resources for this particular job must be allocated. This is done via the PBS Pro job workload manager software, which efficiently distributes workloads across the supercomputer. Extensive informations about PBS Pro can be found in the official documentation [here](#), especially in the PBS Pro User's Guide.

Resources Allocation Policy

The resources are allocated to the job in a fairshare fashion, subject to constraints set by the queue and resources available to the Project. The Fairshare at Salomon ensures that individual users may consume approximately equal amount of resources per week. The resources are accessible via several queues for queueing the jobs. The queues provide prioritized and exclusive access to the computational resources. Following queues are available to Anselm users:

- **qexp**, the Express queue
- **qprod**, the Production queue
- **qlong**, the Long queue
- **qmpp**, the Massively parallel queue
- **qfat**, the queue to access SMP UV2000 machine
- **qfree**, the Free resource utilization queue

!!! Note “Note” Check the queue status at <https://extranet.it4i.cz/rsweb/salomon/>

Read more on the Resource Allocation Policy page.

Job submission and execution

!!! Note “Note” Use the **qsub** command to submit your jobs.

The qsub submits the job into the queue. The qsub command creates a request to the PBS Job manager for allocation of specified resources. The **smallest allocation unit is entire node, 24 cores**, with exception of the qexp queue. The resources will be allocated when available, subject to allocation policies and constraints. **After the resources are allocated the jobscript or interactive shell is executed on first of the allocated nodes.**

Read more on the Job submission and execution page.

Capacity computing

Introduction

In many cases, it is useful to submit huge (100+) number of computational jobs into the PBS queue system. Huge number of (small) jobs is one of the most effective ways to execute embarrassingly parallel calculations, achieving best runtime, throughput and computer utilization.

However, executing huge number of jobs via the PBS queue may strain the system. This strain may result in slow response to commands, inefficient scheduling and overall degradation of performance and user experience, for all users. For this reason, the number of jobs is **limited to 100 per user, 1500 per job array**

!!! Note “Note” Please follow one of the procedures below, in case you wish to schedule more than 100 jobs at a time.

- Use Job arrays when running huge number of multithread (bound to one node only) or multinode (multithread across several nodes) jobs
- Use GNU parallel when running single core jobs
- Combine GNU parallel with Job arrays when running huge number of single core jobs

Policy

1. A user is allowed to submit at most 100 jobs. Each job may be a job array.
2. The array size is at most 1000 subjobs.

Job arrays

!!! Note “Note” Huge number of jobs may be easily submitted and managed as a job array.

A job array is a compact representation of many jobs, called subjobs. The subjobs share the same job script, and have the same values for all attributes and resources, with the following exceptions:

- each subjob has a unique index, \$PBS_ARRAY_INDEX
- job Identifiers of subjobs only differ by their indices
- the state of subjobs can differ (R,Q,...etc.)

All subjobs within a job array have the same scheduling priority and schedule as independent jobs. Entire job array is submitted through a single qsub command and may be managed by qdel, qalter, qhold, qrls and qsig commands as a single job.

Shared jobscript

All subjobs in job array use the very same, single jobscript. Each subjob runs its own instance of the jobscript. The instances execute different work controlled by \$PBS_ARRAY_INDEX variable.

Example:

Assume we have 900 input files with name beginning with “file” (e. g. file001, ..., file900). Assume we would like to use each of these input files with program executable myprog.x, each as a separate job.

First, we create a tasklist file (or subjobs list), listing all tasks (subjobs) - all input files in our example:

```
$ find . -name 'file*' > tasklist
```

Then we create jobscript:

```
#!/bin/bash
#PBS -A PROJECT_ID
#PBS -q qprod
#PBS -l select=1:ncpus=24,walltime=02:00:00

# change to local scratch directory
SCR=/scratch/work/user/$USER/$PBS_JOBID
mkdir -p $SCR ; cd $SCR || exit

# get individual tasks from tasklist with index from PBS JOB ARRAY
TASK=$(sed -n "${PBS_ARRAY_INDEX}p" $PBS_O_WORKDIR/tasklist)

# copy input file and executable to scratch
cp $PBS_O_WORKDIR/$TASK input ; cp $PBS_O_WORKDIR/myprog.x .

# execute the calculation
./myprog.x < input > output

# copy output file to submit directory
cp output $PBS_O_WORKDIR/$TASK.out
```

In this example, the submit directory holds the 900 input files, executable myprog.x and the jobscript file. As input for each run, we take the filename of input file from created tasklist file. We copy the input file to scratch /scratch/work/user/*USER*/PBS_JOBID, execute the myprog.x and copy the output file back to the submit directory, under the \$TASK.out name. The myprog.x runs on one node only and must use threads to run in parallel. Be aware, that if the myprog.x is **not multithreaded**, then all the **jobs are run as single thread programs in sequential** manner. Due to allocation of the whole node, the **accounted time is equal to the usage of whole node**, while using only 1/24 of the node!

If huge number of parallel multicore (in means of multinode multithread, e. g. MPI enabled) jobs is needed to run, then a job array approach should also be used. The main difference compared to previous example using one node is that the local scratch should not be used (as it's not shared between nodes) and MPI or other technique for parallel multinode run has to be used properly.

Submit the job array

To submit the job array, use the qsub -J command. The 900 jobs of the example above may be submitted like this:

```
$ qsub -N JOBNAME -J 1-900 jobscript
506493[] .isrv5
```

In this example, we submit a job array of 900 subjobs. Each subjob will run on full node and is assumed to take less than 2 hours (please note the #PBS directives in the beginning of the jobscript file, don't forget to set your valid PROJECT_ID and desired queue).

Sometimes for testing purposes, you may need to submit only one-element array. This is not allowed by PBSPro, but there's a workaround:

```
$ qsub -N JOBNAME -J 9-10:2 jobscript
```

This will only choose the lower index (9 in this example) for submitting/running your job.

Manage the job array

Check status of the job array by the qstat command.

```
$ qstat -a 506493[].isrv5
```

isrv5:

Job ID	Username	Queue	Jobname	SessID	NDS	TSK	Req'd Memory	Req'd Time	Elap S	Time
12345[].dm2	user2	qprod	xx	13516	1	24	--	00:50	B	00:02

The status B means that some subjobs are already running.

Check status of the first 100 subjobs by the qstat command.

```
$ qstat -a 12345[1-100].isrv5
```

isrv5:

Job ID	Username	Queue	Jobname	SessID	NDS	TSK	Req'd Memory	Req'd Time	Elap S	Time
12345[1].isrv5	user2	qprod	xx	13516	1	24	--	00:50	R	00:02
12345[2].isrv5	user2	qprod	xx	13516	1	24	--	00:50	R	00:02
12345[3].isrv5	user2	qprod	xx	13516	1	24	--	00:50	R	00:01
12345[4].isrv5	user2	qprod	xx	13516	1	24	--	00:50	Q	--
.
,
12345[100].isrv5	user2	qprod	xx	13516	1	24	--	00:50	Q	--

Delete the entire job array. Running subjobs will be killed, queueing subjobs will be deleted.

```
$ qdel 12345[].isrv5
```

Deleting large job arrays may take a while.

Display status information for all user's jobs, job arrays, and subjobs.

```
$ qstat -u $USER -t
```

Display status information for all user's subjobs.

```
$ qstat -u $USER -tJ
```

Read more on job arrays in the PBSPro Users guide.

GNU parallel

!!! Note "Note" Use GNU parallel to run many single core tasks on one node.

GNU parallel is a shell tool for executing jobs in parallel using one or more computers. A job can be a single command or a small script that has to be run for each of the lines in the input. GNU parallel is most useful in running single core jobs via the queue system on Anselm.

For more information and examples see the parallel man page:

```
$ module add parallel
$ man parallel
```

GNU parallel jobscript

The GNU parallel shell executes multiple instances of the jobscript using all cores on the node. The instances execute different work, controlled by the `$PARALLEL_SEQ` variable.

Example:

Assume we have 101 input files with name beginning with “file” (e. g. file001, ..., file101). Assume we would like to use each of these input files with program executable myprog.x, each as a separate single core job. We call these single core jobs tasks.

First, we create a tasklist file, listing all tasks - all input files in our example:

```
$ find . -name 'file*' > tasklist
```

Then we create jobscript:

```
#!/bin/bash
#PBS -A PROJECT_ID
#PBS -q qprod
#PBS -l select=1:ncpus=24,walltime=02:00:00

[ -z "$PARALLEL_SEQ" ] &&
{ module add parallel ; exec parallel -a $PBS_O_WORKDIR/tasklist $0 ; }

# change to local scratch directory
SCR=/scratch/work/user/$USER/$PBS_JOBID/$PARALLEL_SEQ
mkdir -p $SCR ; cd $SCR || exit

# get individual task from tasklist
TASK=$1

# copy input file and executable to scratch
cp $PBS_O_WORKDIR/$TASK input

# execute the calculation
cat input > output

# copy output file to submit directory
cp output $PBS_O_WORKDIR/$TASK.out
```

In this example, tasks from tasklist are executed via the GNU parallel. The jobscript executes multiple instances of itself in parallel, on all cores of the node. Once an instance of jobscript is finished, new instance starts until all entries in tasklist are processed. Currently processed entry of the joblist may be retrieved via `$1` variable. Variable `$TASK` expands to one of the input filenames from tasklist. We copy the input file to local scratch, execute the myprog.x and copy the output file back to the submit directory, under the `$TASK.out` name.

Submit the job

To submit the job, use the `qsub` command. The 101 tasks' job of the example above may be submitted like this:

```
$ qsub -N JOBNAME jobscript
12345.dm2
```

In this example, we submit a job of 101 tasks. 24 input files will be processed in parallel. The 101 tasks on 24 cores are assumed to complete in less than 2 hours.

Please note the #PBS directives in the beginning of the jobscript file, don't forget to set your valid PROJECT_ID and desired queue.

Job arrays and GNU parallel

!!! Note “Note” Combine the Job arrays and GNU parallel for best throughput of single core jobs

While job arrays are able to utilize all available computational nodes, the GNU parallel can be used to efficiently run multiple single-core jobs on single node. The two approaches may be combined to utilize all available (current and future) resources to execute single core jobs.

!!! Note “Note” Every subjob in an array runs GNU parallel to utilize all cores on the node

GNU parallel, shared jobscript

Combined approach, very similar to job arrays, can be taken. Job array is submitted to the queuing system. The subjobs run GNU parallel. The GNU parallel shell executes multiple instances of the jobscript using all cores on the node. The instances execute different work, controlled by the \$PBS_JOB_ARRAY and \$PARALLEL_SEQ variables.

Example:

Assume we have 992 input files with name beginning with “file” (e. g. file001, ..., file992). Assume we would like to use each of these input files with program executable myprog.x, each as a separate single core job. We call these single core jobs tasks.

First, we create a tasklist file, listing all tasks - all input files in our example:

```
$ find . -name 'file*' > tasklist
```

Next we create a file, controlling how many tasks will be executed in one subjob

```
$ seq 32 > numtasks
```

Then we create jobscript:

```
#!/bin/bash
#PBS -A PROJECT_ID
#PBS -q qprod
#PBS -l select=1:ncpus=24,walltime=02:00:00

[ -z "$PARALLEL_SEQ" ] &&
{ module add parallel ; exec parallel -a $PBS_O_WORKDIR/numtasks $0 ; }

# change to local scratch directory
SCR=/scratch/work/user/$USER/$PBS_JOBID/$PARALLEL_SEQ
mkdir -p $SCR ; cd $SCR || exit

# get individual task from tasklist with index from PBS JOB ARRAY and index from Parallel
IDX=$(( $PBS_ARRAY_INDEX + $PARALLEL_SEQ - 1 ))
TASK=$(sed -n "${IDX}p" $PBS_O_WORKDIR/tasklist)
```

```

[ -z "$TASK" ] && exit

# copy input file and executable to scratch
cp $PBS_O_WORKDIR/$TASK input

# execute the calculation
cat input > output

# copy output file to submit directory
cp output $PBS_O_WORKDIR/$TASK.out

```

In this example, the jobscript executes in multiple instances in parallel, on all cores of a computing node. Variable \$TASK expands to one of the input filenames from tasklist. We copy the input file to local scratch, execute the myprog.x and copy the output file back to the submit directory, under the \$TASK.out name. The numtasks file controls how many tasks will be run per subjob. Once an task is finished, new task starts, until the number of tasks in numtasks file is reached.

!!! Note “Note” Select subjob walltime and number of tasks per subjob carefully

When deciding this values, think about following guiding rules :

1. Let $n=N/24$. Inequality $(n+1) * T < W$ should hold. The N is number of tasks per subjob, T is expected single task walltime and W is subjob walltime. Short subjob walltime improves scheduling and job throughput.
2. Number of tasks should be modulo 24.
3. These rules are valid only when all tasks have similar task walltimes T.

Submit the job array

To submit the job array, use the qsub -J command. The 992 tasks’ job of the example above may be submitted like this:

```

$ qsub -N JOBNAME -J 1-992:32 jobscript
12345[] .dm2

```

In this example, we submit a job array of 31 subjobs. Note the -J 1-992:48, this must be the same as the number sent to numtasks file. Each subjob will run on full node and process 24 input files in parallel, 48 in total per subjob. Every subjob is assumed to complete in less than 2 hours.

Please note the #PBS directives in the beginning of the jobscript file, dont’ forget to set your valid PROJECT_ID and desired queue.

Examples

Download the examples in capacity.zip, illustrating the above listed ways to run huge number of jobs. We recommend to try out the examples, before using this for running production jobs.

Unzip the archive in an empty directory on Anselm and follow the instructions in the README file

```

$ unzip capacity.zip
$ cd capacity
$ cat README

```


Resources Allocation Policy

Resources Allocation Policy

The resources are allocated to the job in a fairshare fashion, subject to constraints set by the queue and resources available to the Project. The Fairshare at Anselm ensures that individual users may consume approximately equal amount of resources per week. Detailed information in the Job scheduling section. The resources are accessible via several queues for queueing the jobs. The queues provide prioritized and exclusive access to the computational resources. Following table provides the queue partitioning overview:

queue	active project	project resources	nodes min ncpus*	priority authorization walltime	—			
— qexe	Express queue	no	none required	32 nodes, max 8 per user	24 >150 no 1 / 1h qprod			
Production queue	yes	> 0	>1006 nodes, max 86 per job	24 0 no 24 / 48h qlong	Long queue			
yes	> 0	256 nodes, max 40 per job, only non-accelerated nodes allowed	24 0 no	72 / 144h	qmpp	Massive parallel queue		
yes	> 0	1006 nodes	24 0 yes	2 / 4h qfat	UV2000 queue	yes		
> 0	1 (uv1)	8 0 yes	24 / 48h qfree	Free resource queue	yes	none required	752 nodes, max	
86 per job	24	-1024	no	12 / 12h qviz	Visualization queue	yes	none required	2 (with NVIDIA
Quadro K5000)	4	150	no	1 / 2h				

!!! Note “Note” **The qfree queue is not free of charge.** Normal accounting applies. However, it allows for utilization of free resources, once a Project exhausted all its allocated computational resources. This does not apply for Directors Discretion’s projects (DD projects) by default. Usage of qfree after exhaustion of DD projects computational resources is allowed after request for this queue.

- **qexp**, the Express queue: This queue is dedicated for testing and running very small jobs. It is not required to specify a project to enter the qexp. There are 2 nodes always reserved for this queue (w/o accelerator), maximum 8 nodes are available via the qexp for a particular user. The nodes may be allocated on per core basis. No special authorization is required to use it. The maximum runtime in qexp is 1 hour.
- **qprod**, the Production queue: This queue is intended for normal production runs. It is required that active project with nonzero remaining resources is specified to enter the qprod. All nodes may be accessed via the qprod queue, however only 86 per job. Full nodes, 24 cores per node are allocated. The queue runs with medium priority and no special authorization is required to use it. The maximum runtime in qprod is 48 hours.
- **qlong**, the Long queue: This queue is intended for long production runs. It is required that active project with nonzero remaining resources is specified to enter the qlong. Only 336 nodes without acceleration may be accessed via the qlong queue. Full nodes, 24 cores per node are allocated. The queue runs with medium priority and no special authorization is required to use it. The maximum runtime in qlong is 144 hours (three times of the standard qprod time - 3 * 48 h)
- **qmpp**, the massively parallel queue. This queue is intended for massively parallel runs. It is required that active project with nonzero remaining resources is specified to enter the qmpp. All nodes may be accessed via the qmpp queue. Full nodes, 24 cores per node are allocated. The queue runs with medium priority and no special authorization is required to use it. The maximum runtime in qmpp is 4 hours. An PI needs explicitly ask support for authorization to enter the queue for all users associated to her/his Project.
- **qfat**, the UV2000 queue. This queue is dedicated to access the fat SGI UV2000 SMP machine. The machine (uv1) has 112 Intel IvyBridge cores at 3.3GHz and 3.25TB RAM. An PI needs explicitly ask support for authorization to enter the queue for all users associated to her/his Project.
- **qfree**, the Free resource queue: The queue qfree is intended for utilization of free resources,

after a Project exhausted all its allocated computational resources (Does not apply to DD projects by default. DD projects have to request for permission on qfree after exhaustion of computational resources.). It is required that active project is specified to enter the queue, however no remaining resources are required. Consumed resources will be accounted to the Project. Only 178 nodes without accelerator may be accessed from this queue. Full nodes, 24 cores per node are allocated. The queue runs with very low priority and no special authorization is required to use it. The maximum runtime in qfree is 12 hours.

- **qviz**, the Visualization queue: Intended for pre-/post-processing using OpenGL accelerated graphics. Currently when accessing the node, each user gets 4 cores of a CPU allocated, thus approximately 73 GB of RAM and 1/7 of the GPU capacity (default “chunk”). If more GPU power or RAM is required, it is recommended to allocate more chunks (with 4 cores each) up to one whole node per user, so that all 28 cores, 512 GB RAM and whole GPU is exclusive. This is currently also the maximum allowed allocation per one user. One hour of work is allocated by default, the user may ask for 2 hours maximum.

!!! Note “Note” To access node with Xeon Phi co-processor user needs to specify that in job submission select statement.

Notes

The job wall clock time defaults to **half the maximum time**, see table above. Longer wall time limits can be set manually, see examples.

Jobs that exceed the reserved wall clock time (Req’d Time) get killed automatically. Wall clock time limit can be changed for queuing jobs (state Q) using the qalter command, however can not be changed for a running job (state R).

Salomon users may check current queue configuration at <https://extranet.it4i.cz/rsweb/salomon/queues>.

Queue status

!!! Note “Note” Check the status of jobs, queues and compute nodes at <https://extranet.it4i.cz/rsweb/salomon>

Display the queue status on Salomon:

```
$ qstat -q
```

The PBS allocation overview may be obtained also using the rspbs command.

```
$ rspbs
```

```
Usage: rspbs [options]
```

Options:

--version	show program's version number and exit
-h, --help	show this help message and exit
--get-server-details	Print server
--get-queues	Print queues
--get-queues-details	Print queues details
--get-reservations	Print reservations
--get-reservations-details	Print reservations details
--get-nodes	Print nodes of PBS complex
--get-nodeset	Print nodeset of PBS complex
--get-nodes-details	Print nodes details

Cluster usage



User: Project: Queue:

Figure 1: RSWEB Salomon

```

--get-jobs          Print jobs
--get-jobs-details  Print jobs details
--get-jobs-check-params
                    Print jobid, job state, session_id, user, nodes
--get-users        Print users of jobs
--get-allocated-nodes
                    Print allocated nodes of jobs
--get-allocated-nodeset
                    Print allocated nodeset of jobs
--get-node-users    Print node users
--get-node-jobs     Print node jobs
--get-node-ncpus    Print number of ncpus per node
--get-node-allocated-ncpus
                    Print number of allocated ncpus per node
--get-node-qlist    Print node qlist
--get-node-ibswitch Print node ibswitch
--get-user-nodes    Print user nodes
--get-user-nodeset  Print user nodeset
--get-user-jobs     Print user jobs
--get-user-jobc     Print number of jobs per user
--get-user-nodect   Print number of allocated nodes per user
--get-user-ncpus    Print number of allocated ncpus per user
--get-qlist-nodes   Print qlist nodes
--get-qlist-nodeset Print qlist nodeset
--get-ibswitch-nodes Print ibswitch nodes
--get-ibswitch-nodeset
                    Print ibswitch nodeset
--summary          Print summary
--get-node-ncpu-chart
                    Obsolete. Print chart of allocated ncpus per node
--server=SERVER    Use given PBS server
--state=STATE      Only for given job state
--jobid=JOBID      Only for given job ID
--user=USER        Only for given user
--node=NODE        Only for given node
--nodestate=NODESTATE
                    Only for given node state (affects only --get-node*
                    --get-qlist-* --get-ibswitch-* actions)
--incl-finished    Include finished jobs

```

Resources Accounting Policy

The Core-Hour

The resources that are currently subject to accounting are the core-hours. The core-hours are accounted on the wall clock basis. The accounting runs whenever the computational cores are allocated or blocked via the PBS Pro workload manager (the qsub command), regardless of whether the cores are actually used for any calculation. 1 core-hour is defined as 1 processor core allocated for 1 hour of wall clock time. Allocating a full node (24 cores) for 1 hour accounts to 24 core-hours. See example in the Job submission and execution section.

Check consumed resources

The **it4ifree** command is a part of `it4i.portal.clients` package, located here: <https://pypi.python.org/pypi/it4i.portal.clients>

User may check at any time, how many core-hours have been consumed by himself/herself and his/her projects. The command is available on clusters' login nodes.

```
$ it4ifree
```

```
Password:
```

PID	Total	Used	...by me	Free
OPEN-0-0	1500000	400644	225265	1099356
DD-13-1	10000	2606	2606	7394

Job submission and execution

Job Submission

When allocating computational resources for the job, please specify

1. suitable queue for your job (default is qprod)
2. number of computational nodes required
3. number of cores per node required
4. maximum wall time allocated to your calculation, note that jobs exceeding maximum wall time will be killed
5. Project ID
6. Jobscript or interactive switch

!!! Note “Note” Use the **qsub** command to submit your job to a queue for allocation of the computational resources.

Submit the job using the qsub command:

```
$ qsub -A Project_ID -q queue -l select=x:ncpus=y,walltime=[[hh:]mm:]ss[.ms] jobscript
```

The qsub submits the job into the queue, in another words the qsub command creates a request to the PBS Job manager for allocation of specified resources. The resources will be allocated when available, subject to above described policies and constraints. **After the resources are allocated the jobscript or interactive shell is executed on first of the allocated nodes.**

!!! Note “Note” PBS statement nodes (qsub -l nodes=nodespec) is not supported on Salomon cluster.

Job Submission Examples

```
$ qsub -A OPEN-0-0 -q qprod -l select=64:ncpus=24,walltime=03:00:00 ./myjob
```

In this example, we allocate 64 nodes, 24 cores per node, for 3 hours. We allocate these resources via the qprod queue, consumed resources will be accounted to the Project identified by Project ID OPEN-0-0. Jobscript myjob will be executed on the first node in the allocation.

```
$ qsub -q qexp -l select=4:ncpus=24 -I
```

In this example, we allocate 4 nodes, 24 cores per node, for 1 hour. We allocate these resources via the qexp queue. The resources will be available interactively

```
$ qsub -A OPEN-0-0 -q qlong -l select=10:ncpus=24 ./myjob
```

In this example, we allocate 10 nodes, 24 cores per node, for 72 hours. We allocate these resources via the qlong queue. Jobscript myjob will be executed on the first node in the allocation.

```
$ qsub -A OPEN-0-0 -q qfree -l select=10:ncpus=24 ./myjob
```

In this example, we allocate 10 nodes, 24 cores per node, for 12 hours. We allocate these resources via the qfree queue. It is not required that the project OPEN-0-0 has any available resources left. Consumed resources are still accounted for. Jobscript myjob will be executed on the first node in the allocation.

Intel Xeon Phi co-processors

To allocate a node with Xeon Phi co-processor, user needs to specify that in select statement. Currently only allocation of whole nodes with both Phi cards as the smallest chunk is supported. Standard PBSPro approach through attributes “accelerator”, “naccelerators” and “accelerator_model” is used. The “accelerator_model” can be omitted, since on Salomon only one type of accelerator type/model is available.

The absence of specialized queue for accessing the nodes with cards means, that the Phi cards can be utilized in any queue, including qexp for testing/experiments, qlong for longer jobs, qfree after the project resources have been spent, etc. The Phi cards are thus also available to PRACE users. There’s no need to ask for permission to utilize the Phi cards in project proposals.

```
$ qsub -A OPEN-0-0 -I -q qprod -l select=1:ncpus=24:accelerator=True:naccelerators=2:acc
```

In this example, we allocate 1 node, with 24 cores, with 2 Xeon Phi 7120p cards, running batch job ./myjob. The default time for qprod is used, e. g. 24 hours.

```
$ qsub -A OPEN-0-0 -I -q qlong -l select=4:ncpus=24:accelerator=True:naccelerators=2 -l v
```

In this example, we allocate 4 nodes, with 24 cores per node (totalling 96 cores), with 2 Xeon Phi 7120p cards per node (totalling 8 Phi cards), running interactive job for 56 hours. The accelerator model name was omitted.

UV2000 SMP

!!! Note “Note” 14 NUMA nodes available on UV2000 Per NUMA node allocation. Jobs are isolated by cpusets.

The UV2000 (node uv1) offers 3328GB of RAM and 112 cores, distributed in 14 NUMA nodes. A NUMA node packs 8 cores and approx. 236GB RAM. In the PBS the UV2000 provides 14 chunks, a chunk per NUMA node (see Resource allocation policy). The jobs on UV2000 are isolated from each other by cpusets, so that a job by one user may not utilize CPU or memory allocated to a job by other user. Always, full chunks are allocated, a job may only use resources of the NUMA nodes allocated to itself.

```
$ qsub -A OPEN-0-0 -q qfat -l select=14 ./myjob
```

In this example, we allocate all 14 NUMA nodes (corresponds to 14 chunks), 112 cores of the SGI UV2000 node for 72 hours. Jobscript myjob will be executed on the node uv1.

```
$ qsub -A OPEN-0-0 -q qfat -l select=1:mem=2000GB ./myjob
```

In this example, we allocate 2000GB of memory on the UV2000 for 72 hours. By requesting 2000GB of memory, 10 chunks are allocated. Jobscript myjob will be executed on the node uv1.

Useful tricks

All qsub options may be saved directly into the jobscript. In such a case, no options to qsub are needed.

```
$ qsub ./myjob
```

By default, the PBS batch system sends an e-mail only when the job is aborted. Disabling mail events completely can be done like this:

```
$ qsub -m n
```

Advanced job placement

Placement by name

Specific nodes may be allocated via the PBS

```
qsub -A OPEN-0-0 -q qprod -l select=1:ncpus=24:host=r24u35n680+1:ncpus=24:host=r24u36n681
```

Or using short names

```
qsub -A OPEN-0-0 -q qprod -l select=1:ncpus=24:host=cns680+1:ncpus=24:host=cns681 -I
```

In this example, we allocate nodes r24u35n680 and r24u36n681, all 24 cores per node, for 24 hours. Consumed resources will be accounted to the Project identified by Project ID OPEN-0-0. The resources will be available interactively.

Placement by |Hypercube|dimension|

Nodes may be selected via the PBS resource attribute ehc_[1-7]d .

Hypercube	dimension
1D	ehc_1d
2D	ehc_2d
3D	ehc_3d
4D	ehc_4d
5D	ehc_5d
6D	ehc_6d
7D	ehc_7d

```
$ qsub -A OPEN-0-0 -q qprod -l select=4:ncpus=24 -l place=group=ehc_1d -I
```

In this example, we allocate 4 nodes, 24 cores, selecting only the nodes with hypercube dimension 1.

Placement by IB switch

Groups of computational nodes are connected to chassis integrated Infiniband switches. These switches form the leaf switch layer of the Infiniband network . Nodes sharing the leaf switch can communicate most efficiently. Sharing the same switch prevents hops in the network and provides for unbiased, most efficient network communication.

There are at most 9 nodes sharing the same Infiniband switch.

Infiniband switch list:

```
$ qmgr -c "print node @a" | grep switch
set node r4i1n11 resources_available.switch = r4i1s0sw1
set node r2i0n0 resources_available.switch = r2i0s0sw1
set node r2i0n1 resources_available.switch = r2i0s0sw1
...
```

List of all nodes per Infiniband switch:

```
$ qmgr -c "print node @a" | grep r36sw3
set node r36u31n964 resources_available.switch = r36sw3
```



```

set node r36u32n965 resources_available.switch = r36sw3
set node r36u33n966 resources_available.switch = r36sw3
set node r36u34n967 resources_available.switch = r36sw3
set node r36u35n968 resources_available.switch = r36sw3
set node r36u36n969 resources_available.switch = r36sw3
set node r37u32n970 resources_available.switch = r36sw3
set node r37u33n971 resources_available.switch = r36sw3
set node r37u34n972 resources_available.switch = r36sw3

```

Nodes sharing the same switch may be selected via the PBS resource attribute switch.

We recommend allocating compute nodes of a single switch when best possible computational network performance is required to run the job efficiently:

```
$ qsub -A OPEN-0-0 -q qprod -l select=9:ncpus=24:switch=r4i1s0sw1 ./myjob
```

In this example, we request all the 9 nodes sharing the r4i1s0sw1 switch for 24 hours.

```
$ qsub -A OPEN-0-0 -q qprod -l select=9:ncpus=24 -l place=group=switch ./myjob
```

In this example, we request 9 nodes placed on the same switch using node grouping placement for 24 hours.

HTML commented section #1 (turbo boost is to be implemented)

Job Management

!!! Note “Note” Check status of your jobs using the **qstat** and **check-pbs-jobs** commands

```

$ qstat -a
$ qstat -a -u username
$ qstat -an -u username
$ qstat -f 12345.isrv5

```

Example:

```
$ qstat -a
```

```
srv11:
```

Job ID	Username	Queue	Jobname	SessID	NDS	TSK	Memory	Req'd Time	Req'd S	Elap Time
16287.isrv5	user1	qlong	job1	6183	4	64	--	144:0	R	38:25
16468.isrv5	user1	qlong	job2	8060	4	64	--	144:0	R	17:44
16547.isrv5	user2	qprod	job3x	13516	2	32	--	48:00	R	00:58

In this example user1 and user2 are running jobs named job1, job2 and job3x. The jobs job1 and job2 are using 4 nodes, 16 cores per node each. The job1 already runs for 38 hours and 25 minutes, job2 for 17 hours 44 minutes. The job1 already consumed $64 \times 38.41 = 2458.6$ core hours. The job3x already consumed $0.9632 = 30.93$ core hours. These consumed core hours will be accounted on the respective project accounts, regardless of whether the allocated cores were actually used for computations.

Check status of your jobs using check-pbs-jobs command. Check presence of user's PBS jobs' processes on execution hosts. Display load, processes. Display job standard and error output. Continuously display (tail -f) job standard or error output.

```
$ check-pbs-jobs --check-all
$ check-pbs-jobs --print-load --print-processes
$ check-pbs-jobs --print-job-out --print-job-err
$ check-pbs-jobs --jobid JOBID --check-all --print-all
$ check-pbs-jobs --jobid JOBID --tailf-job-out
```

Examples:

```
$ check-pbs-jobs --check-all
JOB 35141.dm2, session_id 71995, user user2, nodes r3i6n2,r3i6n3
Check session id: OK
Check processes
r3i6n2: OK
r3i6n3: No process
```

In this example we see that job 35141.dm2 currently runs no process on allocated node r3i6n2, which may indicate an execution error.

```
$ check-pbs-jobs --print-load --print-processes
JOB 35141.dm2, session_id 71995, user user2, nodes r3i6n2,r3i6n3
Print load
r3i6n2: LOAD: 16.01, 16.01, 16.00
r3i6n3: LOAD: 0.01, 0.00, 0.01
Print processes
      %CPU CMD
r3i6n2: 0.0 -bash
r3i6n2: 0.0 /bin/bash /var/spool/PBS/mom_priv/jobs/35141.dm2.SC
r3i6n2: 99.7 run-task
...
```

In this example we see that job 35141.dm2 currently runs process run-task on node r3i6n2, using one thread only, while node r3i6n3 is empty, which may indicate an execution error.

```
$ check-pbs-jobs --jobid 35141.dm2 --print-job-out
JOB 35141.dm2, session_id 71995, user user2, nodes r3i6n2,r3i6n3
Print job standard output:
===== Job start =====
Started at      : Fri Aug 30 02:47:53 CEST 2013
Script name     : script
Run loop 1
Run loop 2
Run loop 3
```

In this example, we see actual output (some iteration loops) of the job 35141.dm2

!!! Note “Note” Manage your queued or running jobs, using the **qhold**, **qrls**, **qdel**, **qsig** or **qalter** commands

You may release your allocation at any time, using qdel command

```
$ qdel 12345.isrv5
```

You may kill a running job by force, using qsig command

```
$ qsig -s 9 12345.isrv5
```

Learn more by reading the pbs man page

```
$ man pbs_professional
```

Job Execution

Jobscript

!!! Note “Note” Prepare the jobscript to run batch jobs in the PBS queue system

The Jobscript is a user made script, controlling sequence of commands for executing the calculation. It is often written in bash, other scripts may be used as well. The jobscript is supplied to PBS **qsub** command as an argument and executed by the PBS Professional workload manager.

!!! Note “Note” The jobscript or interactive shell is executed on first of the allocated nodes.

```
$ qsub -q qexp -l select=4:ncpus=24 -N Name0 ./myjob
$ qstat -n -u username
```

isrv5:

Job ID	Username	Queue	Jobname	SessID	NDS	TSK	Memory	Req'd Time	Req'd S	Elap Time
15209.isrv5	username	qexp	Name0	5530	4	96	--	01:00	R	00:00
r21u01n577/0*24+r21u02n578/0*24+r21u03n579/0*24+r21u04n580/0*24										

In this example, the nodes r21u01n577, r21u02n578, r21u03n579, r21u04n580 were allocated for 1 hour via the qexp queue. The jobscript myjob will be executed on the node r21u01n577, while the nodes r21u02n578, r21u03n579, r21u04n580 are available for use as well.

!!! Note “Note” The jobscript or interactive shell is by default executed in home directory

```
$ qsub -q qexp -l select=4:ncpus=24 -I
qsub: waiting for job 15210.isrv5 to start
qsub: job 15210.isrv5 ready
```

```
$ pwd
/home/username
```

In this example, 4 nodes were allocated interactively for 1 hour via the qexp queue. The interactive shell is executed in the home directory.

!!! Note “Note” All nodes within the allocation may be accessed via ssh. Unallocated nodes are not accessible to user.

The allocated nodes are accessible via ssh from login nodes. The nodes may access each other via ssh as well.

Calculations on allocated nodes may be executed remotely via the MPI, ssh, pdsh or clush. You may find out which nodes belong to the allocation by reading the \$PBS_NODEFILE file

```
qsub -q qexp -l select=2:ncpus=24 -I
qsub: waiting for job 15210.isrv5 to start
qsub: job 15210.isrv5 ready
```

```
$ pwd
/home/username
```

```
$ sort -u $PBS_NODEFILE
r2i5n6.ib0.smc.salomon.it4i.cz
r4i6n13.ib0.smc.salomon.it4i.cz
r4i7n0.ib0.smc.salomon.it4i.cz
```

```
r4i7n2.ib0.smc.salomon.it4i.cz
```

```
$ pdsh -w r2i5n6,r4i6n13,r4i7n[0,2] hostname
r4i6n13: r4i6n13
r2i5n6: r2i5n6
r4i7n2: r4i7n2
r4i7n0: r4i7n0
```

In this example, the hostname program is executed via pdsh from the interactive shell. The execution runs on all four allocated nodes. The same result would be achieved if the pdsh is called from any of the allocated nodes or from the login nodes.

Example Jobscript for MPI Calculation

!!! Note “Note” Production jobs must use the /scratch directory for I/O

The recommended way to run production jobs is to change to /scratch directory early in the jobscript, copy all inputs to /scratch, execute the calculations and copy outputs to home directory.

```
#!/bin/bash

# change to scratch directory, exit on failure
SCRDIR=/scratch/work/user/$USER/myjob
mkdir -p $SCRDIR
cd $SCRDIR || exit

# copy input file to scratch
cp $PBS_O_WORKDIR/input .
cp $PBS_O_WORKDIR/mympiprogram.x .

# load the mpi module
module load OpenMPI

# execute the calculation
mpiexec -pernode ./mympiprogram.x

# copy output file to home
cp output $PBS_O_WORKDIR/.

#exit
exit
```

In this example, some directory on the /home holds the input file input and executable mympiprogram.x. We create a directory myjob on the /scratch filesystem, copy input and executable files from the /home directory where the qsub was invoked (\$PBS_O_WORKDIR) to /scratch, execute the MPI program mympiprogram.x and copy the output file back to the /home directory. The mympiprogram.x is executed as one process per node, on all allocated nodes.

!!! Note “Note” Consider preloading inputs and executables onto shared scratch before the calculation starts.

In some cases, it may be impractical to copy the inputs to scratch and outputs to home. This is especially true when very large input and output files are expected, or when the files should be reused by a subsequent calculation. In such a case, it is users responsibility to preload the

input files on shared /scratch before the job submission and retrieve the outputs manually, after all calculations are finished.

!!! Note “Note” Store the qsub options within the jobscript. Use **mpiprocs** and **ompthreads** qsub options to control the MPI job execution.

Example jobscript for an MPI job with preloaded inputs and executables, options for qsub are stored within the script :

```
#!/bin/bash
#PBS -q qprod
#PBS -N MYJOB
#PBS -l select=100:ncpus=24:mpiprocs=1:ompthreads=24
#PBS -A OPEN-0-0

# change to scratch directory, exit on failure
SCRDIR=/scratch/work/user/$USER/myjob
cd $SCRDIR || exit

# load the mpi module
module load OpenMPI

# execute the calculation
mpiexec ./mymprog.x

#exit
exit
```

In this example, input and executable files are assumed preloaded manually in /scratch/\$USER/myjob directory. Note the **mpiprocs** and **ompthreads** qsub options, controlling behavior of the MPI execution. The mympprog.x is executed as one process per node, on all 100 allocated nodes. If mympprog.x implements OpenMP threads, it will run 24 threads per node.

HTML commented section #2 (examples need to be reworked)

Example Jobscript for Single Node Calculation

!!! Note “Note” Local scratch directory is often useful for single node jobs. Local scratch will be deleted immediately after the job ends. Be very careful, use of RAM disk filesystem is at the expense of operational memory.

Example jobscript for single node calculation, using local scratch on the node:

```
#!/bin/bash

# change to local scratch directory
cd /lscratch/$PBS_JOBID || exit

# copy input file to scratch
cp $PBS_O_WORKDIR/input .
cp $PBS_O_WORKDIR/myprog.x .

# execute the calculation
./myprog.x
```

```
# copy output file to home
cp output $PBS_O_WORKDIR/.

#exit
exit
```

In this example, some directory on the home holds the input file input and executable myprog.x. We copy input and executable files from the home directory where the qsub was invoked ($PBS_{O_w}ORKDIR$) to *localscratch/lscratch*/PBS_JOBID, execute the myprog.x and copy the output file back to the /home directory. The myprog.x runs on one node only and may use threads.

Job scheduling

Job execution priority

Scheduler gives each job an execution priority and then uses this job execution priority to select which job(s) to run.

Job execution priority is determined by these job properties (in order of importance):

1. queue priority
2. fairshare priority
3. eligible time

Queue priority

Queue priority is priority of queue where job is queued before execution.

Queue priority has the biggest impact on job execution priority. Execution priority of jobs in higher priority queues is always greater than execution priority of jobs in lower priority queues. Other properties of job used for determining job execution priority (fairshare priority, eligible time) cannot compete with queue priority.

Queue priorities can be seen at <https://extranet.it4i.cz/rsweb/salomon/queues>

Fairshare priority

Fairshare priority is priority calculated on recent usage of resources. Fairshare priority is calculated per project, all members of project share same fairshare priority. Projects with higher recent usage have lower fairshare priority than projects with lower or none recent usage.

Fairshare priority is used for ranking jobs with equal queue priority.

Fairshare priority is calculated as

$$MAX_FAIRSHARE * (1 - \frac{usage_{Project}}{usage_{Total}})$$

Figure 1:

where MAX_FAIRSHARE has value 1E6, usage_{Project} is cumulated usage by all members of selected project, usage_{Total} is total usage by all users, by all projects.

Usage counts allocated corehours (ncpus*walltime). Usage is decayed, or cut in half periodically, at the interval 168 hours (one week). Jobs queued in queue qexp are not calculated to project's usage.

!!! Note “Note” Calculated usage and fairshare priority can be seen at <https://extranet.it4i.cz/rsweb/salomon/projects>.

Calculated fairshare priority can be also seen as Resource_List.fairshare attribute of a job.

Eligible time

Eligible time is amount (in seconds) of eligible time job accrued while waiting to run. Jobs with higher eligible time gains higher priority.

Eligible time has the least impact on execution priority. Eligible time is used for sorting jobs with equal queue priority and fairshare priority. It is very, very difficult for eligible time to compete with fairshare priority.

Eligible time can be seen as `eligible_time` attribute of job.

Formula

Job execution priority (job sort formula) is calculated as:

$$1000 * \text{queue_priority} + \frac{\text{fairshare_priority}}{1000} + \frac{\text{eligible_time}}{864000}$$

Figure 2:

Job backfilling

The scheduler uses job backfilling.

Backfilling means fitting smaller jobs around the higher-priority jobs that the scheduler is going to run next, in such a way that the higher-priority jobs are not delayed. Backfilling allows us to keep resources from becoming idle when the top job (job with the highest execution priority) cannot run.

The scheduler makes a list of jobs to run in order of execution priority. Scheduler looks for smaller jobs that can fit into the usage gaps around the highest-priority jobs in the list. The scheduler looks in the prioritized list of jobs and chooses the highest-priority smaller jobs that fit. Filler jobs are run only if they will not delay the start time of top jobs.

It means, that jobs with lower execution priority can be run before jobs with higher execution priority.

!!! Note “Note” It is **very beneficial to specify the walltime** when submitting jobs.

Specifying more accurate walltime enables better scheduling, better execution times and better resource usage. Jobs with suitable (small) walltime could be backfilled - and overtake job(s) with higher priority.

Job placement

Job placement can be controlled by flags during submission.

CESNET Data Storage

Introduction

Do not use shared filesystems at IT4Innovations as a backup for large amount of data or long-term archiving purposes.

!!! Note “Note” [../img/The IT4Innovations](#) does not provide storage capacity for data archiving. Academic staff and students of research institutions in the Czech Republic can use CESNET Storage service [\[1\]](#).

The CESNET Storage service can be used for research purposes, mainly by academic staff and students of research institutions in the Czech Republic.

User of data storage CESNET (DU) association can become organizations or an individual person who is either in the current employment relationship (employees) or the current study relationship (students) to a legal entity (organization) that meets the “Principles for access to CESNET Large infrastructure (Access Policy)”.

User may only use data storage CESNET for data transfer and storage which are associated with activities in science, research, development, the spread of education, culture and prosperity. In detail see “Acceptable Use Policy CESNET Large Infrastructure (Acceptable Use Policy, AUP)”.

The service is documented at <https://du.cesnet.cz/wiki/doku.php/en/start> [\[1\]](#). For special requirements please contact directly CESNET Storage Department via e-mail [du-support\(at\)cesnet.cz](mailto:du-support@cesnet.cz).

The procedure to obtain the CESNET access is quick and trouble-free.

(source <https://du.cesnet.cz/> [\[1\]](#))

CESNET storage access

Understanding Cesnet storage

!!! Note “Note” It is very important to understand the Cesnet storage before uploading data. Please read <https://du.cesnet.cz/en/navody/home-migrace-plzen/start> [\[1\]](#) first.

Once registered for CESNET Storage, you may access the storage [\[1\]](#) in number of ways. We recommend the SSHFS and RSYNC methods.

SSHFS Access

!!! Note “Note” SSHFS: The storage will be mounted like a local hard drive

The SSHFS provides a very convenient way to access the CESNET Storage. The storage will be mounted onto a local directory, exposing the vast CESNET Storage as if it was a local removable harddrive. Files can be then copied in and out in a usual fashion.

First, create the mountpoint

```
$ mkdir cesnet
```

Mount the storage. Note that you can choose among the [ssh.du1.cesnet.cz](#) (Plzen), [ssh.du2.cesnet.cz](#) (Jihlava), [ssh.du3.cesnet.cz](#) (Brno) Mount tier1_home (**only 5120M !**):

```
$ sshfs username@ssh.du1.cesnet.cz:. cesnet/
```

For easy future access from Anselm, install your public key

```
$ cp .ssh/id_rsa.pub cesnet/.ssh/authorized_keys
```

Mount tier1_cache_tape for the Storage VO:

```
$ sshfs username@ssh.du1.cesnet.cz:/cache_tape/VO_storage/home/username cesnet/
```

View the archive, copy the files and directories in and out

```
$ ls cesnet/
$ cp -a mydir cesnet/.
$ cp cesnet/myfile .
```

Once done, please remember to unmount the storage


```
$ fusermount -u cesnet
```

Rsync access

!!! Note “Note” Rsync provides delta transfer for best performance, can resume interrupted transfers

Rsync is a fast and extraordinarily versatile file copying tool. It is famous for its delta-transfer algorithm, which reduces the amount of data sent over the network by sending only the differences between the source files and the existing files in the destination. Rsync is widely used for backups and mirroring and as an improved copy command for everyday use.

Rsync finds files that need to be transferred using a “quick check” algorithm (by default) that looks for files that have changed in size or in last-modified time. Any changes in the other preserved attributes (as requested by options) are made on the destination file directly when the quick check indicates that the file’s data does not need to be updated.

More about Rsync at https://du.cesnet.cz/en/navody/rsync/start#pro_bezne_uzivatele 

Transfer large files to/from Cesnet storage, assuming membership in the Storage VO

```
$ rsync --progress datafile username@ssh.du1.cesnet.cz:VO_storage-cache_tape/.
$ rsync --progress username@ssh.du1.cesnet.cz:VO_storage-cache_tape/datafile .
```

Transfer large directories to/from Cesnet storage, assuming membership in the Storage VO

```
$ rsync --progress -av datafolder username@ssh.du1.cesnet.cz:VO_storage-cache_tape/.
$ rsync --progress -av username@ssh.du1.cesnet.cz:VO_storage-cache_tape/datafolder .
```

Transfer rates of about 28MB/s can be expected.

Storage

Introduction

There are two main shared file systems on Salomon cluster, the HOME and SCRATCH.

All login and compute nodes may access same data on shared filesystems. Compute nodes are also equipped with local (non-shared) scratch, ramdisk and tmp filesystems.

Policy (in a nutshell)

!!! Note “Note” Use [for your most valuable data and programs. Use WORK for your large project files. Use TEMP for large scratch data.

Do not use for [archiving](storage/#archiving)!

Archiving

Please don't use shared filesystems as a backup for large amount of data or long-term archiving mean. The academic staff and students of research institutions in the Czech Republic can use CESNET storage service, which is available via SSHFS.

Shared Filesystems

Salomon computer provides two main shared filesystems, the HOME filesystem and the SCRATCH filesystem. The SCRATCH filesystem is partitioned to WORK and TEMP workspaces. The HOME filesystem is realized as a tiered NFS disk storage. The SCRATCH filesystem is realized as a parallel Lustre filesystem. Both shared file systems are accessible via the Infiniband network. Extended ACLs are provided on both HOME/SCRATCH filesystems for the purpose of sharing data with other users using fine-grained control.

HOME filesystem

The HOME filesystem is realized as a Tiered filesystem, exported via NFS. The first tier has capacity 100TB, second tier has capacity 400TB. The filesystem is available on all login and computational nodes. The Home filesystem hosts the HOME workspace.

SCRATCH filesystem

The architecture of Lustre on Salomon is composed of two metadata servers (MDS) and six data/object storage servers (OSS). Accessible capacity is 1.69 PB, shared among all users. The SCRATCH filesystem hosts the WORK and TEMP workspaces.

Configuration of the SCRATCH Lustre storage

- SCRATCH Lustre object storage
 - Disk array SFA12KX
 - 540 4TB SAS 7.2krpm disks
 - 54 OSTs of 10 disks in RAID6 (8+2)
 - 15 hot-spare disks

- 4x 400GB SSD cache
- SCRATCH Lustre metadata storage
 - Disk array EF3015
 - 12 600GB SAS 15krpm disks

Understanding the Lustre Filesystems

(source <http://www.nas.nasa.gov>)

A user file on the Lustre filesystem can be divided into multiple chunks (stripes) and stored across a subset of the object storage targets (OSTs) (disks). The stripes are distributed among the OSTs in a round-robin fashion to ensure load balancing.

When a client (a compute node from your job) needs to create or access a file, the client queries the metadata server (MDS) and the metadata target (MDT) for the layout and location of the file's stripes. Once the file is opened and the client obtains the striping information, the MDS is no longer involved in the file I/O process. The client interacts directly with the object storage servers (OSSes) and OSTs to perform I/O operations such as locking, disk allocation, storage, and retrieval.

If multiple clients try to read and write the same part of a file at the same time, the Lustre distributed lock manager enforces coherency so that all clients see consistent results.

There is default stripe configuration for Salomon Lustre filesystems. However, users can set the following stripe parameters for their own directories or files to get optimum I/O performance:

1. `stripe_size`: the size of the chunk in bytes; specify with k, m, or g to use units of KB, MB, or GB, respectively; the size must be an even multiple of 65,536 bytes; default is 1MB for all Salomon Lustre filesystems
2. `stripe_count` the number of OSTs to stripe across; default is 1 for Salomon Lustre filesystems one can specify -1 to use all OSTs in the filesystem.
3. `stripe_offset` The index of the OST where the first stripe is to be placed; default is -1 which results in random selection; using a non-default value is NOT recommended.

!!! Note “Note” Setting stripe size and stripe count correctly for your needs may significantly impact the I/O performance you experience.

Use the `lfs getstripe` for getting the stripe parameters. Use the `lfs setstripe` command for setting the stripe parameters to get optimal I/O performance. The correct stripe setting depends on your needs and file access patterns.

```
$ lfs getstripe dir|filename
$ lfs setstripe -s stripe_size -c stripe_count -o stripe_offset dir|filename
```

Example:

```
$ lfs getstripe /scratch/work/user/username
/scratch/work/user/username
stripe_count: 1 stripe_size: 1048576 stripe_offset: -1

$ lfs setstripe -c -1 /scratch/work/user/username/
$ lfs getstripe /scratch/work/user/username/
/scratch/work/user/username/
stripe_count: -1 stripe_size: 1048576 stripe_offset: -1
```

In this example, we view current stripe setting of the `/scratch/username/` directory. The stripe count is changed to all OSTs, and verified. All files written to this directory will be striped over all

(54) OSTs

Use `lfs check OSTs` to see the number and status of active OSTs for each filesystem on Salomon. Learn more by reading the man page

```
$ lfs check osts
$ man lfs
```

Hints on Lustre Stripping

!!! Note “Note” Increase the `stripe_count` for parallel I/O to the same file.

When multiple processes are writing blocks of data to the same file in parallel, the I/O performance for large files will improve when the `stripe_count` is set to a larger value. The stripe count sets the number of OSTs the file will be written to. By default, the stripe count is set to 1. While this default setting provides for efficient access of metadata (for example to support the `ls -l` command), large files should use stripe counts of greater than 1. This will increase the aggregate I/O bandwidth by using multiple OSTs in parallel instead of just one. A rule of thumb is to use a stripe count approximately equal to the number of gigabytes in the file.

Another good practice is to make the stripe count be an integral factor of the number of processes performing the write in parallel, so that you achieve load balance among the OSTs. For example, set the stripe count to 16 instead of 15 when you have 64 processes performing the writes.

!!! Note “Note” Using a large stripe size can improve performance when accessing very large files

Large stripe size allows each client to have exclusive access to its own part of a file. However, it can be counterproductive in some cases if it does not match your I/O pattern. The choice of stripe size has no effect on a single-stripe file.

Read more on http://wiki.lustre.org/manual/LustreManual20_HTML/ManagingStripingFreeSpace.html

Disk usage and quota commands

User quotas on the Lustre file systems (SCRATCH) can be checked and reviewed using following command:

```
$ lfs quota dir
```

Example for Lustre SCRATCH directory:

```
$ lfs quota /scratch
```

```
Disk quotas for user user001 (uid 1234):
```

Filesystem	kbytes	quota	limit	grace	files	quota	limit	grace	
/scratch	8	0	100000000000	-	3	0	0	-	

```
Disk quotas for group user001 (gid 1234):
```

Filesystem	kbytes	quota	limit	grace	files	quota	limit	grace
/scratch	8	0	0	-	3	0	0	-

In this example, we view current quota size limit of 100TB and 8KB currently used by user001.

HOME directory is mounted via NFS, so a different command must be used to obtain quota information:

```
$ quota
```

Example output:

```
$ quota
Disk quotas for user vop999 (uid 1025):
    Filesystem blocks quota limit grace files quota limit grace
home-nfs-ib.salomon.it4i.cz:/home
                28      0 2500000000          10      0 500000
```

To have a better understanding of where the space is exactly used, you can use following command to find out.

```
$ du -hs dir
```

Example for your HOME directory:

```
$ cd /home
$ du -hs * .[a-zA-z0-9]* | grep -E "[0-9]*G|[0-9]*M" | sort -hr
258M      cuda-samples
15M       .cache
13M       .mozilla
5,5M     .eclipse
2,7M     .idb_13.0_linux_intel64_app
```

This will list all directories which are having MegaBytes or GigaBytes of consumed space in your actual (in this example HOME) directory. List is sorted in descending order from largest to smallest files/directories.

To have a better understanding of previous commands, you can read manpages.

```
$ man lfs
```

```
$ man du
```

Extended Access Control List (ACL)

Extended ACLs provide another security mechanism beside the standard POSIX ACLs which are defined by three entries (for owner/group/others). Extended ACLs have more than the three basic entries. In addition, they also contain a mask entry and may contain any number of named user and named group entries.

ACLs on a Lustre file system work exactly like ACLs on any Linux file system. They are manipulated with the standard tools in the standard manner. Below, we create a directory and allow a specific user access.

```
[vop999@login1.salomon ~]$ umask 027
[vop999@login1.salomon ~]$ mkdir test
[vop999@login1.salomon ~]$ ls -ld test
drwxr-x--- 2 vop999 vop999 4096 Nov  5 14:17 test
[vop999@login1.salomon ~]$ getfacl test
# file: test
# owner: vop999
# group: vop999
user::rwx
group::r-x
other::---

[vop999@login1.salomon ~]$ setfacl -m user:johnsm:rwx test
[vop999@login1.salomon ~]$ ls -ld test
drwxrwx---+ 2 vop999 vop999 4096 Nov  5 14:17 test
```


```
[vop999@login1.salomon ~]$ getfacl test
# file: test
# owner: vop999
# group: vop999
user::rwx
user:johnsm:rwx
group::r-x
mask::rwx
other::---
```

Default ACL mechanism can be used to replace setuid/setgid permissions on directories. Setting a default ACL on a directory (-d flag to setfacl) will cause the ACL permissions to be inherited by any newly created file or subdirectory within the directory. Refer to this page for more information on Linux ACL:

http://www.vanemery.com/Linux/ACL/POSIX_ACL_on_Linux.html 

Shared Workspaces

HOME

Users home directories /home/username reside on HOME filesystem. Accessible capacity is 0.5PB, shared among all users. Individual users are restricted by filesystem usage quotas, set to 250GB per user. If 250GB should prove as insufficient for particular user, please contact support , the quota may be lifted upon request.

!!! Note “Note” The HOME filesystem is intended for preparation, evaluation, processing and storage of data generated by active Projects.

The HOME should not be used to archive data of past Projects or other unrelated data.

The files on HOME will not be deleted until end of the users lifecycle.

The workspace is backed up, such that it can be restored in case of catastrophic failure resulting in significant data loss. This backup however is not intended to restore old versions of user data or to restore (accidentaly) deleted files.

HOME workspace	
Accesspoint	/home/username
Capacity	0.5PB
Throughput	6GB/s
User quota	250GB
Protocol	NFS, 2-Tier

WORK

The WORK workspace resides on SCRATCH filesystem. Users may create subdirectories and files in directories /scratch/work/user/username and /scratch/work/project/projectid. The /scratch/work/user/username is private to user, much like the home directory. The /scratch/work/project/projectid is accessible to all users involved in project projectid.

!!! Note “Note” The WORK workspace is intended to store users project data as well as for high performance access to input and output files. All project data should be removed once the project

is finished. The data on the WORK workspace are not backed up.

Files on the WORK filesystem are ****persistent**** (not automatically deleted) throughout duration of the job.

The WORK workspace is hosted on SCRATCH filesystem. The SCRATCH is realized as Lustre parallel filesystem and is available from all login and computational nodes. Default stripe size is 1MB, stripe count is 1. There are 54 OSTs dedicated for the SCRATCH filesystem.

!!! Note “Note” Setting stripe size and stripe count correctly for your needs may significantly impact the I/O performance you experience.

WORK workspace	
Accesspoints	/scratch/work/user/username, /scratch/work/user/projectid
Capacity	1.6P
Throughput	30GB/s
User quota	100TB
Default stripe size	1MB
Default stripe count	1
Number of OSTs	54
Protocol	Lustre

TEMP

The TEMP workspace resides on SCRATCH filesystem. The TEMP workspace accesspoint is /scratch/temp. Users may freely create subdirectories and files on the workspace. Accessible capacity is 1.6P, shared among all users on TEMP and WORK. Individual users are restricted by filesystem usage quotas, set to 100TB per user. The purpose of this quota is to prevent runaway programs from filling the entire filesystem and deny service to other users. >If 100TB should prove as insufficient for particular user, please contact support [\[4\]](#), the quota may be lifted upon request.

!!! Note “Note” The TEMP workspace is intended for temporary scratch data generated during the calculation as well as for high performance access to input and output files. All I/O intensive jobs must use the TEMP workspace as their working directory.

Users are advised to save the necessary data from the TEMP workspace to HOME or WORK after the calculation is finished.

Files on the TEMP filesystem that are ****not accessed for more than 90 days**** will be automatically deleted.

The TEMP workspace is hosted on SCRATCH filesystem. The SCRATCH is realized as Lustre parallel filesystem and is available from all login and computational nodes. Default stripe size is 1MB, stripe count is 1. There are 54 OSTs dedicated for the SCRATCH filesystem.

!!! Note “Note” Setting stripe size and stripe count correctly for your needs may significantly impact the I/O performance you experience.

TEMP workspace	
Accesspoint	/scratch/temp
Capacity	1.6P
Throughput	30GB/s
User quota	100TB
Default stripe size	1MB
Default stripe count	1
Number of OSTs	54
Protocol	Lustre

RAM disk

Every computational node is equipped with filesystem realized in memory, so called RAM disk.

!!! Note “Note” Use RAM disk in case you need really fast access to your data of limited size during your calculation. Be very careful, use of RAM disk filesystem is at the expense of operational memory.

The local RAM disk is mounted as /ramdisk and is accessible to user at /ramdisk/\$PBS_JOBID directory.

The local RAM disk filesystem is intended for temporary scratch data generated during the calculation as well as for high performance access to input and output files. Size of RAM disk filesystem is limited. Be very careful, use of RAM disk filesystem is at the expense of operational memory. It is not recommended to allocate large amount of memory and use large amount of data in RAM disk filesystem at the same time.

!!! Note “Note” The local RAM disk directory /ramdisk/\$PBS_JOBID will be deleted immediately after the calculation end. Users should take care to save the output data from within the jobscript.

RAM disk	
Mountpoint	/ramdisk
Accesspoint	/ramdisk/\$PBS_JOBID
Capacity	20 GB
Throughput	1.5 GB/s write, over 5 GB/s read, sin- gle thread, over 10 GB/s write, over 50 GB/s read, 16 threads
User quota	none

Summary

Mount Point	Usage
/home	home directory
/scratch	work project files
/scratch1	temporary data
/ramdisk	temporary data, node local

PRACE User Support

Intro

PRACE users coming to Salomon as to TIER-1 system offered through the DECI calls are in general treated as standard users and so most of the general documentation applies to them as well. This section shows the main differences for quicker orientation, but often uses references to the original documentation. PRACE users who don't undergo the full procedure (including signing the IT4I AuP on top of the PRACE AuP) will not have a password and thus access to some services intended for regular users. This can lower their comfort, but otherwise they should be able to use the TIER-1 system as intended. Please see the Obtaining Login Credentials section, if the same level of access is required.

All general PRACE User Documentation [\[1\]](#) should be read before continuing reading the local documentation here.

Help and Support

If you have any troubles, need information, request support or want to install additional software, please use PRACE Helpdesk [\[2\]](#).

Information about the local services are provided in the introduction of general user documentation. Please keep in mind, that standard PRACE accounts don't have a password to access the web interface of the local (IT4Innovations) request tracker and thus a new ticket should be created by sending an e-mail to support[at]it4i.cz.

Obtaining Login Credentials

In general PRACE users already have a PRACE account setup through their HOMESITE (institution from their country) as a result of rewarded PRACE project proposal. This includes signed PRACE AuP, generated and registered certificates, etc.

If there's a special need a PRACE user can get a standard (local) account at IT4Innovations. To get an account on the Salomon cluster, the user needs to obtain the login credentials. The procedure is the same as for general users of the cluster, so please see the corresponding section of the general documentation here.

Accessing the cluster




Access with GSI-SSH

For all PRACE users the method for interactive access (login) and data transfer based on grid services from Globus Toolkit (GSI SSH and GridFTP) is supported.

The user will need a valid certificate and to be present in the PRACE LDAP (please contact your HOME SITE or the primary investigator of your project for LDAP account creation).

Most of the information needed by PRACE users accessing the Salomon TIER-1 system can be found here:

- General user's FAQ [\[3\]](#)
- Certificates FAQ [\[4\]](#)

- Interactive access using GSISSH 
- Data transfer with GridFTP 
- Data transfer with gtransfer 

Before you start to use any of the services don't forget to create a proxy certificate from your certificate:

```
$ grid-proxy-init
```

To check whether your proxy certificate is still valid (by default it's valid 12 hours), use:

```
$ grid-proxy-info
```

To access Salomon cluster, two login nodes running GSI SSH service are available. The service is available from public Internet as well as from the internal PRACE network (accessible only from other PRACE partners).

Access from PRACE network:

It is recommended to use the single DNS name salomon-prace.it4i.cz which is distributed between the two login nodes. If needed, user can login directly to one of the login nodes. The addresses are:

Login address	Port
salomon-prace.it4i.cz	2222
login1-prace.salomon.it4i.cz	2222
login2-prace.salomon.it4i.cz	2222
login3-prace.salomon.it4i.cz	2222
login4-prace.salomon.it4i.cz	2222

```
$ gsissh -p 2222 salomon-prace.it4i.cz
```

When logging from other PRACE system, the prace_service script can be used:

```
$ gsissh `prace_service -i -s salomon`
```

Access from public Internet:

It is recommended to use the single DNS name salomon.it4i.cz which is distributed between the two login nodes. If needed, user can login directly to one of the login nodes. The addresses are:

Login address	Port
salomon.it4i.cz	2222
login1.salomon.it4i.cz	2222
login2-prace.salomon.it4i.cz	2222
login3-prace.salomon.it4i.cz	2222
login4-prace.salomon.it4i.cz	2222

```
$ gsissh -p 2222 salomon.it4i.cz
```

When logging from other PRACE system, the prace_service script can be used:

```
$ gsissh `prace_service -e -s salomon`
```

Although the preferred and recommended file transfer mechanism is using GridFTP, the GSI SSH implementation on Salomon supports also SCP, so for small files transfer gsiscp can be used:

```
$ gsiscp -P 2222 _LOCAL_PATH_TO_YOUR_FILE_ salomon.it4i.cz:_SALOMON_PATH_TO_YOUR_FILE_
```

```
$ gsiscp -P 2222 salomon.it4i.cz:_SALOMON_PATH_TO_YOUR_FILE_ _LOCAL_PATH_TO_YOUR_FILE_
```

```
$ gsiscp -P 2222 _LOCAL_PATH_TO_YOUR_FILE_ salomon-prace.it4i.cz:_SALOMON_PATH_TO_YOU
```

```
$ gsiscp -P 2222 salomon-prace.it4i.cz:_SALOMON_PATH_TO_YOUR_FILE_ _LOCAL_PATH_TO_YOU
```

Access to X11 applications (VNC)

If the user needs to run X11 based graphical application and does not have a X11 server, the applications can be run using VNC service. If the user is using regular SSH based access, please see the section in general documentation.

If the user uses GSI SSH based access, then the procedure is similar to the SSH based access (look here), only the port forwarding must be done using GSI SSH:

```
$ gsissh -p 2222 salomon.it4i.cz -L 5961:localhost:5961
```

Access with SSH

After successful obtainment of login credentials for the local IT4Innovations account, the PRACE users can access the cluster as regular users using SSH. For more information please see the section in general documentation.

File transfers

PRACE users can use the same transfer mechanisms as regular users (if they've undergone the full registration procedure). For information about this, please see the section in the general documentation.

Apart from the standard mechanisms, for PRACE users to transfer data to/from Salomon cluster, a GridFTP server running Globus Toolkit GridFTP service is available. The service is available from public Internet as well as from the internal PRACE network (accessible only from other PRACE partners).

There's one control server and three backend servers for striping and/or backup in case one of them would fail.

Access from PRACE network:

Login address	Port
gridftp-prace.salomon.it4i.cz	2812
lgw1-prace.salomon.it4i.cz	2813
lgw2-prace.salomon.it4i.cz	2813
lgw3-prace.salomon.it4i.cz	2813

Copy files **to** Salomon by running the following commands on your local machine:

```
$ globus-url-copy file://_LOCAL_PATH_TO_YOUR_FILE_ gsiftp://gridftp-prace.salomon.it4
```

Or by using `prace_service` script:

```
$ globus-url-copy file://_LOCAL_PATH_TO_YOUR_FILE_ gsiftp://`prace_service -i -f salo
```

Copy files **from** Salomon:

```
$ globus-url-copy gsiftp://gridftp-prace.salomon.it4i.cz:2812/home/prace/_YOUR_ACCOUNT_ON_SALOMON
```

Or by using prace_service script:

```
$ globus-url-copy gsiftp://`prace_service` -i -f salomon`/home/prace/_YOUR_ACCOUNT_ON_SALOMON
```

Access from public Internet:

Login address	Port
gridftp.salomon.it4i.cz	2812
lgw1.salomon.it4i.cz	2813
lgw2.salomon.it4i.cz	2813
lgw3.salomon.it4i.cz	2813

Copy files **to** Salomon by running the following commands on your local machine:

```
$ globus-url-copy file://_LOCAL_PATH_TO_YOUR_FILE_ gsiftp://gridftp.salomon.it4i.cz:2812/home/prace/_YOUR_ACCOUNT_ON_SALOMON
```

Or by using prace_service script:

```
$ globus-url-copy file://_LOCAL_PATH_TO_YOUR_FILE_ gsiftp://`prace_service` -e -f salomon`/home/prace/_YOUR_ACCOUNT_ON_SALOMON
```

Copy files **from** Salomon:

```
$ globus-url-copy gsiftp://gridftp.salomon.it4i.cz:2812/home/prace/_YOUR_ACCOUNT_ON_SALOMON
```

Or by using prace_service script:

```
$ globus-url-copy gsiftp://`prace_service` -e -f salomon`/home/prace/_YOUR_ACCOUNT_ON_SALOMON
```

Generally both shared file systems are available through GridFTP:

File	
sys-	
tem	
mount	
point	Filesystem

/home	Lustre
-------	--------

/scratch	Lustre
----------	--------

More information about the shared file systems is available [here](#).

Please note, that for PRACE users a “prace” directory is used also on the SCRATCH file system.

Data type	Default path
large project files	/scratch/work/user/prace/login/
large scratch/temporary data	/scratch/temp/

Usage of the cluster

There are some limitations for PRACE user when using the cluster. By default PRACE users aren't allowed to access special queues in the PBS Pro to have high priority or exclusive access to some special equipment like accelerated nodes and high memory (fat) nodes. There may be also restrictions obtaining a working license for the commercial software installed on the cluster, mostly because of the license agreement or because of insufficient amount of licenses.

For production runs always use scratch file systems. The available file systems are described here.

Software, Modules and PRACE Common Production Environment

All system wide installed software on the cluster is made available to the users via the modules. The information about the environment and modules usage is in this section of general documentation.

PRACE users can use the “prace” module to use the PRACE Common Production Environment [\[4\]](#).

```
$ module load prace
```

Resource Allocation and Job Execution

General information about the resource allocation, job queuing and job execution is in this section of general documentation.

For PRACE users, the default production run queue is “qprace”. PRACE users can also use two other queues “qexp” and “qfree”.

	Active
queue	project

qexp	no
Ex-	
press	
queue	

qprace	yes
Pro-	
duc-	
tion	
queue	

qfree	yes
Free	
re-	
source	
queue	

qprace, the PRACE This queue is intended for normal production runs. It is required that active project with nonzero remaining resources is specified to enter the qprace. The queue runs with medium priority and no special authorization is required to use it. The maximum runtime in qprace is 48 hours. If the job needs longer time, it must use checkpoint/restart functionality.

Accounting & Quota

The resources that are currently subject to accounting are the core hours. The core hours are accounted on the wall clock basis. The accounting runs whenever the computational cores are allocated or blocked via the PBS Pro workload manager (the qsub command), regardless of whether the cores are actually used for any calculation. See example in the general documentation.

PRACE users should check their project accounting using the PRACE Accounting Tool (DART) [\[1\]](#).

Users who have undergone the full local registration procedure (including signing the IT4Innovations Acceptable Use Policy) and who have received local password may check at any time, how many core-hours have been consumed by themselves and their projects using the command “it4ifree”. Please note that you need to know your user password to use the command and that the displayed core hours are “system core hours” which differ from PRACE “standardized core hours”.

!!! Note “Note” The **it4ifree** command is a part of it4i.portal.clients package, located here: <https://pypi.python.org/pypi/it4i.portal.clients> [\[2\]](#)

```
$ it4ifree
Password:
  PID      Total    Used    ...by me Free
-----
  OPEN-0-0 1500000 400644    225265 1099356
  DD-13-1   10000    2606      2606    7394
```

By default file system quota is applied. To check the current status of the quota (separate for HOME and SCRATCH) use

```
$ quota
$ lfs quota -u USER_LOGIN /scratch
```

If the quota is insufficient, please contact the support and request an increase.

Hardware Overview

Introduction

The Salomon cluster consists of 1008 computational nodes of which 576 are regular compute nodes and 432 accelerated nodes. Each node is a powerful x86-64 computer, equipped with 24 cores (two twelve-core Intel Xeon processors) and 128GB RAM. The nodes are interlinked by high speed InfiniBand and Ethernet networks. All nodes share 0.5PB /home NFS disk storage to store the user files. Users may use a DDN Lustre shared storage with capacity of 1.69 PB which is available for the scratch project data. The user access to the Salomon cluster is provided by four login nodes.

More about schematic representation of the Salomon cluster compute nodes IB topology.



Figure 1: Salomon

The parameters are summarized in the following tables:

General information

In general	
Primary purpose	High Performance Computing
Architecture of compute nodes	x86-64
Operating system	CentOS 6.7 Linux
Compute nodes	
Totally	1008
Processor	2x Intel Xeon E5-2680v3, 2.5GHz, 12cores
RAM	128GB, 5.3GB per core, DDR4@2133 MHz

In general	
Local disk drive	no
Compute network / Topology	InfiniBand FDR56 / 7D Enhanced hypercube
w/o accelerator	576
MIC accelerated	432
In total	
Total theoretical peak performance (Rpeak)	2011 Tflop/s
Total amount of RAM	129.024 TB

Compute nodes

Node	Count
w/o ac- cel- era- tor	576
MIC ac- cel- er- ated	432

For more details please refer to the Compute nodes.

Remote visualization nodes

For remote visualization two nodes with NICE DCV software are available each configured:

Node	Count
visualization	2

SGI UV 2000

For large memory computations a special SMP/NUMA SGI UV 2000 server is available:

Node	Count
UV2000	1



Figure 2:

PBS Professional® 12.0

Reference Guide



PBS Works™

PBS Works is a division of  Altair

Altair PBS Professional 12 Reference Guide, updated: 1/25/13

Copyright © 2003-2012 Altair Engineering, Inc. All rights reserved.

PBS™, PBS Works™, PBS GridWorks®, PBS Professional®, PBS Analytics™, PBS Catalyst™, e-Compute™, and e-Render™ are trademarks of Altair Engineering, Inc. and are protected under U.S. and international laws and treaties. All other marks are the property of their respective owners.

ALTAIR ENGINEERING INC. Proprietary and Confidential. Contains Trade Secret Information. Not for use or disclosure outside ALTAIR and its licensed clients. Information contained herein shall not be decompiled, disassembled, duplicated or disclosed in whole or in part for any purpose. Usage of the software is only as explicitly permitted in the end user software license agreement.

Copyright notice does not imply publication.

For documentation and the PBS Works forums, go to:

Web: www.pbsworks.com

For more information, contact Altair at:

Email: pbssales@altair.com

Technical Support

Location	Telephone	e-mail
North America	+1 248 614 2425	pbssupport@altair.com
China	+86 (0)21 6117 1666	es@altair.com.cn
France	+33 (0)1 4133 0992	francesupport@altair.com
Germany	+49 (0)7031 6208 22	hwsupport@altair.de
India	+91 80 66 29 4500	pbs-support@india.altair.com
Italy	+39 0832 315573 +39 800 905595	support@altairengineering.it
Japan	+81 3 5396 2881	pbs@altairjp.co.jp
Korea	+82 31 728 8600	support@altair.co.kr
Scandinavia	+46 (0)46 286 2050	support@altair.se
UK	+44 (0)1926 468 600	pbssupport@uk.altair.com

This document is proprietary information of Altair Engineering, Inc.

Table of Contents

About PBS Documentation	vii
1 Terminology	1
2 PBS Commands	25
2.1 Windows Requirements	25
2.2 Requirements for Commands	25
2.3 mpiexec	29
2.4 nqs2pbs	31
2.5 pbs-report.	33
2.6 pbs_account.	42
2.7 pbs_attach	45
2.8 pbs_datservice	46
2.9 pbs_ds_password.	47
2.10 pbs_hostn.	49
2.11 pbs_idled	50
2.12 pbs	52
2.13 pbs_interactive.	53
2.14 pbs_lamboot	54
2.15 pbs_migrate_users	55
2.16 pbs_mkdirs	56
2.17 pbs_mom	58
2.18 pbs_mom_globus.	64
2.19 pbs_mpihp	65
2.20 pbs_mpilam	66
2.21 pbs_mpirun	67
2.22 pbs_password	69
2.23 pbs_probe	71
2.24 pbs_python	73
2.25 pbs_rdel	76

Table of Contents

2.26	pbs_renew	77
2.27	pbs_rstat	78
2.28	pbs_rsub	79
2.29	pbs_sched	88
2.30	pbs_server	90
2.31	pbs_tclsh	95
2.32	pbs_tmrsh	96
2.33	pbs_topologyinfo	98
2.34	pbs_wish	99
2.35	pbsdsh	100
2.36	pbsfs	102
2.37	pbsnodes	105
2.38	pbsrun	107
2.39	pbsrun_unwrap	124
2.40	pbsrun_wrap	125
2.41	printjob	127
2.42	qalter	128
2.43	qdel	143
2.44	qdisable	146
2.45	qenable	147
2.46	qhold	149
2.47	qmgr	151
2.48	qmove	171
2.49	qmsg	173
2.50	qorder	175
2.51	qrerun	176
2.52	qrts	178
2.53	qrun	180
2.54	qselect	183
2.55	qsig	191
2.56	qstart	193
2.57	qstat	194
2.58	qstop	208
2.59	qsub	210
2.60	qterm	229
2.61	tracejob	232
2.62	xpbs	235
2.63	xpbsmon	250
3	MOM Parameters	267
3.1	Syntax of MOM Configuration File	267

Table of Contents

3.2	Contents of MOM Configuration File.	268
4	Scheduler Parameters	281
4.1	Format of Scheduler's Configuration File.	281
4.2	Configuration Parameters	282
5	Resources	297
5.1	Resource Data Types.	297
5.2	Advice on Using Resources	297
5.3	Custom Resource Formats.	299
5.4	Built-in Resources	299
5.5	Custom Cray Resources	306
5.6	Specifying Architectures	307
6	Attributes	309
6.1	When Attribute Changes Take Effect	309
6.2	How To Set Attributes.	309
6.3	Viewing Attribute Values	311
6.4	Attribute Table Format	312
6.5	Caveats	313
6.6	Server Attributes	314
6.7	Scheduler Attributes	340
6.8	Reservation Attributes.	341
6.9	Queue Attributes	354
6.10	Vnode Attributes	366
6.11	Job Attributes	375
6.12	Hook Attributes	400
7	Formats	403
7.1	List of Formats	403
8	States	411
8.1	Job States.	411
8.2	Job Array States.	414
8.3	Subjob States.	415
8.4	Server States	415
8.5	Vnode States	416
8.6	Reservation States	419
9	Accounting Log	423

Table of Contents

9.1	Log Entry Format	423
9.2	Record Types.....	424
10	Example Configurations	435
10.1	Single Vnode System	436
10.2	Separate Server and Execution Host.....	437
10.3	Multiple Execution Hosts	438
10.4	Complex Multi-level Route Queues	440
10.5	External Software License Management.....	443
10.6	Multiple User ACL Example	444
11	Run Limit Error Messages	445
11.1	Run Limit Error Messages	445
12	Error Codes	449
13	Request Codes	459
14	PBS Environment Variables	463
14.1	PBS Environment Variables	463
15	File Listing	467
16	Log Messages	489
Appendix A: License Agreement		495
Index		505

About PBS Documentation

Where to Keep the Documentation

To make cross-references work, put all of the PBS guides in the same directory.

What is PBS Professional?

PBS is a workload management system that provides a unified batch queuing and job management interface to a set of computing resources.

The PBS Professional Documentation

The documentation for PBS Professional includes the following:

PBS Professional Administrator's Guide:

Provides the PBS administrator with the information required to configure and manage PBS Professional (PBS).

PBS Professional Quick Start Guide:

Provides a quick overview of PBS Professional installation and license file generation.

PBS Professional Installation & Upgrade Guide:

Contains information on installing and upgrading PBS Professional.

PBS Professional User's Guide:

Covers user commands and how to submit, monitor, track, delete, and manipulate jobs.

PBS Professional Programmer's Guide:

Discusses the PBS application programming interface (API).

PBS Professional Reference Guide:

Contains PBS reference material.

PBS Manual Pages:

Describe PBS commands, resources, attributes, APIs

Ordering Software and Publications

To order additional copies of this manual and other PBS publications, or to purchase additional software licenses, contact your Altair sales representative. Contact information is included on the copyright page of this book.

Document Conventions

PBS documentation uses the following typographic conventions:

abbreviation

The shortest acceptable abbreviation of a command or subcommand is underlined.

`command`

Commands such as `qmgr` and `scp`

input

Command-line instructions

`manpage (x)`

File and path names. Manual page references include the section number in parentheses appended to the manual page name.

formats

Formats

Attributes

Attributes, parameters, objects, variable names, resources, types

Values

Keywords, instances, states, values, labels

Definitions

Terms being defined

Output

Output or example code

File contents

Chapter 1

Terminology

This chapter describes the terms used in PBS Professional documentation.

Accept an action (Hooks)

A hook *accepts* an action when the hook allows the action to take place.

Access control list, ACL

An *ACL*, or *Access Control List*, is a list of users, groups, or hosts from which users or groups may be attempting to gain access. This list defines who or what is allowed or denied access to parts of PBS such as the Server, queues, or reservations. A Server ACL applies to access to the Server, and therefore all of PBS. A queue's ACL applies only to that particular queue. A reservation's ACL applies only to that particular reservation. See [section 9.3.4, “ACLs”, on page 643](#).

Access to a queue

Applies to users, groups, and hosts. Being able to submit jobs to the queue, move jobs into the queue, being able to perform operations on jobs in the queue, and being able to get the status of the queue.

Access to a reservation

Applies to users, groups, and hosts. Being able to place jobs in the reservation, whether by submitting jobs to the reservation or moving jobs into the reservation. It also means being able to delete the reservation, and being able to operate on the jobs in the reservation.

Access to the server

Applies to users, groups, and hosts. Being able to run PBS commands to submit jobs and perform operations on them such as altering, selecting, and querying status. It also means being able to get the status of the Server and queues.

Account

An *account* is an arbitrary character string, which may have meaning to one or more hosts in the batch system. Frequently, an account is used as a grouping for charging for the use of resources.

Action (Hooks)

A PBS operation or state transition. The actions that hooks can affect are submitting a job, altering a job, running a job, making a reservation, and moving a job to another queue.

Active (Failover)

A server daemon is active when it is managing user requests and communicating with the scheduler and MOMs.

Active Directory (Windows)

Active Directory is an implementation of LDAP directory services by Microsoft to use in Windows environments. It is a directory service used to store information about the network resources (e.g. user accounts and groups) across a domain.

Admin (Windows)

A user logged in from an account that is either:

1. A member of a group having full control over the local computer and the domain controller
2. Allowed to make domain and schema changes to the Active Directory.

Administrator, PBS Administrator

A person who administers PBS, performing functions such as downloading, installing, upgrading, configuring, or managing PBS. A PBS administrator must have an account with Manager privilege and an account with root privilege. Administrator is distinguished from “site administrator”, although often these are the same person.

The term *PBS Administrator* is used in PBS documentation to mean the above, rather than to indicate a user role recognized by PBS Professional.

Administrators (Windows)

A group that has built-in capabilities that give its members full control over the local system, or the domain controller host itself.

Advance reservation

A reservation for a specific set of resources for a specified start time and duration in the future. Advance reservations are created by users to reserve resources for jobs. The reservation is available only to the creator of the reservation and any users or groups specified by the creator.

AOE, Application operating environment

The environment on a vnode. This may be one that results from provisioning that vnode, or one that is already in place

API

PBS provides an *Application Programming Interface*, or *API*, which is used by the commands to communicate with the Server. This API is described in the PBS Professional Programmer's Guide. A site may make use of the API to implement new commands if so desired.

Application Checkpoint

The application performs its own checkpointing when it receives the appropriate signal etc.

Array job

See ["Job array"](#).

Attribute

An *attribute* is a data item belonging to an object. The attribute's value affects the behavior of or provides information about the object. A job's owner can set the attributes of a job, and the administrator can set attributes of queues and vnodes.

Backfilling

A scheduling policy where

1. High-priority jobs are scheduled for execution
 2. Lower-priority jobs are run if the following conditions are true:
 - Resources (that cannot be used by the high-priority jobs) are available
 - The lower-priority jobs will not delay the higher-priority jobs
- Lower-priority jobs selected for execution are those next in priority order that will fit in the available resources.

Batch, Batch processing

Allowing jobs to be run outside of the interactive login environment.

Borrowing vnode

The vnode where a shared vnode resource is available, but not managed.

Built-in resource

A resource that is defined in PBS Professional as shipped. Examples of built-in resources are `ncpus`, which tracks the number of CPUs, and `mem`, which tracks memory. See [section 5.4.1, "Built-in and Custom Resources" on page 287 in the PBS Professional Administrator's Guide](#).

Checkpoint/Restart

Allows jobs to be checkpointed and restarted. Uses OS-provided or third-party checkpoint/restart facility.

Checkpoint and Abort, `checkpoint_abort`

The checkpoint script or tool writes a restart file, then PBS kills and requeues the job. The job resumes from the start file when it is executed again.

Chunk

A set of resources allocated as a unit to a job. Specified inside a selection directive. All parts of a chunk come from the same host. In a typical MPI (Message-Passing Interface) job, there is one chunk per MPI process.

Cluster

A relatively homogeneous set of systems that are used as if they are a single machine.

Commands

PBS supplies both command line programs that are POSIX 1003.2d conforming and a graphical interface. These are used to submit, monitor, modify, and delete jobs. These client commands can be installed on any system type supported by PBS and do not require the local presence of any of the other components of PBS.

There are three classifications of commands: user commands (which any authorized user can use), Operator commands, and Manager (or administrator) commands. Operator and Manager commands require specific access privileges.

Complex

A PBS complex consists of the machines running one primary Server+Scheduler (plus, optionally, a secondary backup Server+Scheduler) and all the machines on which the MOMs (attached to this Server+Scheduler) are running. A complex can be a heterogeneous mix of system architectures, and can include one or more clusters.

Consumable resource

A consumable resource is a resource that is reduced or taken up by being used. Examples of consumable resources are memory or CPUs. See [section 5.4.3, "Consumable and Non-consumable Resources" on page 288 in the PBS Professional Administrator's Guide](#).

CPU

Has two meanings, one from a hardware viewpoint, and one from a software viewpoint:

1. A core. The part of a processor that carries out computational tasks. Some systems present virtual cores, for example in hyperthreading.
2. Resource required to execute a program thread. PBS schedules jobs according, in part, to the number of threads, giving each thread a core on which to execute. The resource used by PBS to track CPUs is called “*ncpus*”. The number of CPUs available for use defaults to the number of cores reported by the OS. When a job requests one CPU, it is requesting one core on which to run.

Custom resource

A resource that is not defined in PBS as shipped. Custom resources are created by the PBS administrator or by PBS for some systems. See [section 5.4.1, "Built-in and Custom Resources" on page 287 in the PBS Professional Administrator's Guide](#).

Degraded reservation

An advance reservation for which one or more associated vnodes are unavailable.

A standing reservation for which one or more vnodes associated with any occurrence are unavailable.

Delegation (Windows)

A capability provided by Active Directory that allows granular assignment of privileges to a domain account or group. So for instance, instead of adding an account to the “Account Operators” group which might give too much access, delegation allows giving the account read access only to all domain users and groups information. This is done via the Delegation wizard.

Destination, destination queue

A queue where a job is sent. A destination may be at the same PBS server or at another server.

Destination identifier

The string that names a destination queue. It is composed of two parts and has the following format:

queue@server
where

server

The name of a PBS Server

queue

The string identifying a queue on that Server.

Directive

A means by which the user specifies to PBS the value of a job submission variable such as number of CPUs, the name of the job, etc. The default start of a directive is “*#PBS*”. PBS directives either specify resource requirements or attribute values. See page [section 3.11, "Changing the Job's PBS Directive"](#), on page 59 of the [PBS Professional User's Guide](#).

Domain Admin Account (Windows)

A domain account on Windows that is a member of the “Domain Admins” group.

Domain Admins (Windows)

A global group whose members are authorized to administer the domain. By default, the Domain Admins group is a member of the Administrators group on all computers that have joined a domain, including the domain controllers.

Domain User Account (Windows)

A domain account on Windows that is a member of the Domain Users group.

Domain Users (Windows)

A global group that, by default, includes all user accounts in a domain. When you create a user account in a domain, it is added to this group automatically.

Enterprise Admins (Windows)

A group that exists only in the root domain of an Active Directory forest of domains. The group is authorized to make forest-wide changes in Active Directory, such as adding child domains.

Entity, PBS entity

A user, group, or host.

Entity share

Setting job execution and/or preemption priority according to how much of the fairshare tree is assigned to each job's owner.

Execution host

A computer which runs PBS jobs. An *execution host* is a system with a single operating system (OS) image, a unified virtual memory space, one or

more CPUs and one or more IP addresses. Systems like Linux clusters, which contain separate computational units each with their own OS, are collections of hosts. Systems such as the SGI ICE are also collections of hosts. The SGI Altix 4700 is a single execution host.

An execution host can be comprised of one or more vnodes. For example, the SGI Altix 4700, while being a single execution host, can contain multiple vnodes, where each vnode is a blade. On the SGI ICE, each blade is treated as a vnode. See ["Vnode"](#).

Execution queue

A queue from which a job can be executed.

Failover

The PBS complex can run a backup server. If the primary server fails, the secondary takes over without an interruption in service.

Fairshare

A scheduling policy that prioritizes jobs according to how much of a specified resource is being used by, and has recently been used by, job submitters. Job submitters can be organized into groups and subgroups, so that jobs can also be prioritized according to those groups' resource usage. Users and groups can each be allotted a percentage of total resource usage. See [section 4.8.18, "Using Fairshare" on page 165 in the PBS Professional Administrator's Guide](#).

File staging

File staging is the transfer of files between a specified storage location and the execution host. See ["Stage in"](#) and ["Stage out"](#).

Finished jobs

Jobs whose execution is done, for any reason:

- Jobs which finished execution successfully and exited
- Jobs terminated by PBS while running
- Jobs whose execution failed because of system or network failure
- Jobs which were deleted before they could start execution

Floating license

A unit of license dynamically allocated (checked out) when a user begins using an application on some host (when the job starts), and deallocated (checked in) when a user finishes using the application (when the job ends).

Furnishing queue

In peer scheduling, the queue from which jobs are pulled to be run at another complex

Generic group limit

A limit that applies separately to groups at the server or a queue. This is the limit for groups which have no individual limit specified. A limit for generic groups is applied to the usage across the entire group. A separate limit can be specified at the server and each queue.

Generic project limit

Applies separately to projects at the server or a queue. The limit for projects which have no individual limit specified. A limit for generic projects is applied to the usage across the entire project. A separate limit can be specified at the server and each queue.

Generic user limit

A limit that applies separately to users at the server or a queue. This is the limit for users who have no individual limit specified. A separate limit for generic users can be specified at the server and at each queue.

Global resource

A global resource is defined in a `resources_available` attribute, at the server, a queue, or a host. Global resources can be operated on via the `qmgr` command and are visible via the `qstat` and `pbsnodes` commands. See [section 5.4.5, "Global and Local Resources" on page 289 in the PBS Professional Administrator's Guide](#).

Group

A collection of system users. A user must be a member of at least one group, and can be a member of more than one group.

Group access, Access by group

Refers to access to PBS objects, such as the server, queues, and reservations. A user in the specified group is allowed access at the Server, queues, and reservations

Group ID (GID)

Unique numeric identifier assigned to each group. See ["Group"](#).

Group limit

Refers to configurable limits on resources and jobs. This is a limit applied to the total used by a group, whether the limit is a generic group limit or an individual group limit.

History jobs

Jobs which will no longer execute at this server:

- Moved jobs
- Finished jobs

Hold

A restriction which prevents a job from being executed. When a job has a hold applied to it, it is in the *Held (H)* state. See [section 2.46, “qhold”, on page 149](#).

Hook

Hooks are custom executables that can be run at specific points in the execution of PBS. They accept, reject, or modify the upcoming action. This provides job filtering, patches or workarounds, and extends the capabilities of PBS, without the need to modify source code.

Host

A machine running an operating system. A host can be made up of one or more vnodes. All vnodes of a host share the same value for `resources_available.host`.

Host access, Access by host

Refers to user access at the Server, queues, and reservations from the specified host

HPC Basic Profile (HPCBP)

Proposed standard web services specification for basic job execution capabilities defined by the OGSA High Performance Computing Profile Working Group.

HPC Basic Profile Job, HPCBP Job

Generic job that can run either on vnodes managed by PBS or on nodes managed by HPC Basic Profile Server.

HPC Basic Profile Server

Service that executes jobs from any HPC Basic Profile compliant client

HPCBP MOM

MOM that sends jobs for execution to an HPC Basic Profile Server. This MOM is a client-side implementation of the HPC Basic Profile Specification, and acts as a proxy for and interface to an HPC Basic Profile compliant server.

Idle

A server daemon is idle when it is running, but only accepting handshake messages, not performing workload management.

Indirect resource

A shared vnode resource at vnode(s) where the resource is not defined, but which share the resource.

Individual group limit

Applies separately to groups at the server or a queue. This is the limit for a group which has its own individual limit specified. An individual group limit overrides the generic group limit, but only in the same context, for example, at a particular queue. The limit is applied to the usage across the entire group. A separate limit can be specified at the server and each queue.

Individual project limit

Applies separately to projects at the server or a queue. Limit for a project which has its own individual limit specified. An individual project limit overrides the generic project limit, but only in the same context, for example, at a particular queue. The limit is applied to the usage across the entire project. A separate limit can be specified at the server and each queue.

Individual user limit

Applies separately to users at the server or a queue. This is the limit for users who have their own individual limit specified. A limit for an individual user overrides the generic user limit, but only in the same context, for example, at a particular queue. A separate limit can be specified at the server and each queue.

Installation account

The account used by the administrator when installing PBS. Not the *pbsadmin* account used by PBS.

Interactive job

A job where standard input and output are connected to the terminal from which the job was submitted.

Job or Batch job

A unit of work managed by PBS. A *job* is a related set of tasks, created and submitted by the user. The user specifies the resources required by the job, and the processes that make up the job. When the user submits a job to PBS, the user is handing off these tasks to PBS to manage. PBS then schedules the job to be run, and manages the running of the job, treating the tasks as parts of a whole. A job is usually composed of a set of directives and a shell script.

Job array

A *job array* is a container for a collection of similar jobs submitted under a single job ID. It can be submitted, queried, modified and displayed as a unit. The jobs in the collection are called subjobs. For more on job arrays, see [section , "Job Arrays", on page 235 of the PBS Professional User's Guide](#).

Job array identifier

The identifier returned upon success when submitting a job array. The format is

sequence_number[]

or

sequence_number[][*.server.domain.com*].

Note that some shells require you to enclose a job array identifier in double quotes.

Job array range

A specification for a set of subjobs within a job array. When specifying a range, indices used must be valid members of the job array's indices. Format:

sequence_number[<*first*>-<*last*>:<*step*>][*.server*][@*new server*]

first is the first index of the subjobs.

last is the last index of the subjobs.

step is the stepping factor.

Job ID, Job identifier

When a job is successfully submitted to PBS, PBS returns a unique identifier for the job. Format:

sequence_number[*.server*][@*new server*]

Job state

A job exists in one of the possible states throughout its existence within the PBS system. For example, a job can be queued, running, or exiting. See ["States" on page 411](#).

Job Submission Description Language (JSDL)

Language for describing the resource requirements of jobs.

Job-wide resource, Server resource

A server resource, also called a server-level or job-wide resource, is a resource that is available at the server. A server resource is available to be consumed or matched at the server if you set the server's `resources_available.<resource name>` attribute to the available or matching value. For example, you can define a custom resource called *FloatingLicenses* and set the server's `resources_available.FloatingLicenses` attribute to the number of available floating licenses.

A server resource is a job-wide resource. This means that a job can request this resource for the entire job, but not for individual chunks.

An example of a job-wide resource is shared scratch space, or any custom resource that is defined at the server and queue level.

Job-wide resource request

A job-wide resource request is for resource(s) at the server or queue level. This resource must be a server-level or queue-level resource. A job-wide resource is designed to be used by the entire job, and is available to the complex, not just one execution host. Job-wide resources are requested outside of a selection statement, in this form:

```
-l keyword=value[,keyword=value ...]
```

where *keyword* identifies either a consumable resource or a time-based resource such as **walltime**.

A resource request “outside of a selection statement” means that the resource request comes after “-l”, but not after “-lselect=”.

Kill a job

To terminate the execution of a job.

License Manager Daemon (**lmx-serv-altair**)

Daemon that functions as the license server.

License server

Manages licenses for PBS jobs.

License Server List Configuration

One form of redundant license server configuration. A collection of license server files, or “<port>@<host>” settings, pointing to license servers managing Altair licenses. Each server on the list is tried in turn. There could be X licenses on <server1>, Y licenses on <server2>, and Z licenses on <server3>, and the total licenses available would actually be X+Y+Z, but a request must be satisfied only by one server at a time. The first running server is the only server queried.

Limit

A maximum that can be applied in various situations:

- The maximum amount of a resource that can be consumed at any time by running jobs
- The maximum amount of a resource that can be allocated to queued jobs,
- The maximum number of jobs that can be running
- The maximum number of jobs that can be queued

Load balance

Scheduling policy wherein jobs are distributed across multiple hosts to even out the workload on each host.

Local resource

A local resource is defined in a Version 1 MOM configuration file. Local resources cannot be operated on via the `qmgr` command and are not visible via the `qstat` and `pbsnodes` commands. Local resources can be used by the scheduler. See [section 5.4.5, "Global and Local Resources" on page 289 in the PBS Professional Administrator's Guide](#).

Manager

A person who has been granted Manager privilege by being listed in the Server's `managers` attribute. A Manager is authorized to use all restricted capabilities of PBS. A PBS Manager may act upon the Server, queues, or jobs. See [section 9.2.2.3, "Manager" on page 640 in the PBS Professional Administrator's Guide](#).

Managing vnode

The vnode where a shared vnode resource is defined, and which manages the resource.

Master provisioning script, Master script (Hooks)

The script that makes up the provisioning hook.

Memory-only vnode

Represents a node board that has only memory resources (no CPUs), for example, an Altix memory-only blade.

MOM

The daemon which runs on an execution host, managing the jobs on that host. *MOM* is the informal name for the process called `pbs_mom`. One MOM runs on each execution host.

MOM runs each job when it receives a copy of the job from the Server. MOM creates a new session that is as identical to the user's login session as possible. For example under UNIX, if the user's login shell is `csh`, then MOM creates a session in which `.login` is run as well as `.cshrc`. MOM returns the job's output to the user when directed to do so by the Server.

MOM is a reverse-engineered acronym that stands for "Machine Oriented Mini-server".

Monitoring

The act of tracking and reserving system resources and enforcing usage policy. This covers both user-level and system-level monitoring as well as

monitoring running jobs. Tools are provided to aid human monitoring of the PBS system as well.

Mother Superior

Mother Superior is the MOM on the head or first host of a multihost job. Mother Superior controls the job, communicates with the Server, and controls and consolidates resource usage information. When a job is to run on more than one execution host, the job is sent to the MOM on the primary execution host, which then starts the job.

Moved jobs

Jobs which were moved to another server

Node

No longer used. See ["Execution host"](#).

Non-consumable resource

A non-consumable resource is a resource that is not reduced or taken up by being used. Examples of non-consumable resources are Boolean resources and walltime. See [section 5.4.3, "Consumable and Non-consumable Resources" on page 288 in the PBS Professional Administrator's Guide](#).

Object, PBS object

An element of PBS such as the Server, a queue, or a reservation

Occurrence of a standing reservation

An instance of the standing reservation.

An occurrence of a standing reservation behaves like an advance reservation, with the following exceptions:

- While a job can be submitted to a specific advance reservation, it can only be submitted to the standing reservation as a whole, not to a specific occurrence. You can only specify *when* the job is eligible to run. See the `qsub(1B)` man page.
- When an advance reservation ends, it and all of its jobs, running or queued, are deleted, but when an occurrence ends, only its running jobs are deleted.

Each occurrence of a standing reservation has reserved resources which satisfy the resource request, but each occurrence may have its resources drawn from a different source. A query for the resources assigned to a standing reservation will return the resources assigned to the soonest occurrence, shown in the `resv_nodes` attribute reported by `pbs_rstat`.

Operator

This term means a person who has been granted Operator privilege by being listed in the Server's **operators** attribute. An Operator can use some but not all of the restricted capabilities of PBS. See [section 9.2.2.2, "Operator" on page 639 in the PBS Professional Administrator's Guide](#).

Overall limit

Limit on the total usage. In the context of server limits, this is the limit for usage at the PBS complex. In the context of queue limits, this is the limit for usage at the queue. An overall limit is applied to the total usage at the specified location. Separate overall limits can be specified at the server and each queue.

Owner, Job owner

The user who submitted a specific job to PBS.

Parameter

A *parameter* specifies an element of the behavior of a component of PBS. For example, MOMs have parameters specifying which events to log, or what the maximum load should be. Parameters are specified by editing the component's configuration files.

PBS Entity

A user, group, or host

pbs Module

The *pbs module* is an interface to PBS and the hook environment. The interface is made up of Python objects, which have attributes and methods. You can operate on these objects using Python code.

PBS Object

An element of PBS such as the Server, a queue, or a reservation

pbsadmin (Windows)

The account that is used to execute the PBS daemons `pbs_server`, `pbs_mom`, `pbs_sched`, and `pbs_rshd` via the Service Control Manager on Windows. This must be "*pbsadmin*".

PBS_HOME

The path containing PBS files. The path under which PBS files are installed on the local system.

PBS_EXEC

The path containing PBS executables. The path under which PBS executables are installed on the local system.

PBS Professional

A workload management system consisting of a Server, a Scheduler, and any number of execution hosts each managed by a MOM. PBS accepts batch jobs from users, and schedules them on execution hosts according to the policy chosen by the site. PBS manages the jobs and their output according to site-specified policy.

Peer scheduling

A feature allowing different PBS complexes to automatically run each others' jobs. This way jobs can be dynamically load-balanced across the complexes. Each complex involved in peer scheduling is called a *peer*.

Placement set

A set of vnodes on which jobs can be run, selected so that the job will run as efficiently as possible. Placement sets are used to improve task placement (optimizing to provide a “good fit”) by exposing information on system configuration and topology. See [section 4.8.32, "Placement Sets" on page 205 in the PBS Professional Administrator's Guide](#).

Placement set series

The set of placement sets defined by a resource, where each set has the same value for the resource. If the resource takes on N values, there are N placement sets in the series. See [section 4.8.32, "Placement Sets" on page 205 in the PBS Professional Administrator's Guide](#).

Placement pool

All of the placement sets defined at a PBS object. Each queue can have its own placement pool, and the server can have its own placement pool. See [section 4.8.32, "Placement Sets" on page 205 in the PBS Professional Administrator's Guide](#).

Policy, Scheduling policy

The set of rules by which the scheduler selects jobs for execution.

POSIX

Refers to the various standards developed by the Technical Committee on Operating Systems and Application Environments of the IEEE Computer Society under standard P1003.

Preempt

Stop one or more running jobs in order to start a higher-priority job.

Preemption level

Job characteristic used to determine whether a job may preempt another or may be preempted, such as being in an express queue, starving, having an

owner who is over a soft limit, being a normal job, or having an owner who is over a fairshare allotment.

Preemption method

The method by which a job is preempted. This can be checkpointing, suspension, or requeueing.

Primary Scheduler

The PBS Professional scheduler daemon which is running during normal operation.

Primary Server

The PBS Professional server daemon which is running during normal operation.

Project

In PBS, a project is a way to group jobs independently of users and groups. A project is a tag that identifies a set of jobs. Each job's `project` attribute specifies the job's project.

Project limit

This is a limit applied to the total used by a project, whether the limit is a generic project limit or an individual project limit.

Provision

To install an OS or application, or to run a script which performs installation and/or setup

Provisioned vnode

A vnode which, through the process of provisioning, has an OS or application that was installed, or which has had a script run on it

Provisioning hook

The hook which performs the provisioning, either by calling other scripts or by running commands

Provisioning tool

A tool that performs the actual provisioning, e.g. SGI Tempo.

Pulling queue

In peer scheduling, the queue into which jobs are pulled, and from which they are run

Queue

A *queue* is a named container for jobs at a Server. There are two types of queues in PBS: routing queues and execution queues. A *routing queue* is a queue used to move jobs to other queues including those that exist on other

PBS Servers. A job must reside in an *execution queue* to be eligible to run and remains in an execution queue during the time it is running. In spite of the name, jobs in a queue need not be processed in queue order (first-come first-served or *FIFO*).

Queuing

The collecting together of work or tasks to be run on a computer. Users submit tasks or “jobs” to the resource management system where they are queued up until the system is ready to run them.

Redundant License Server Configuration

Allows licenses to continue to be available should one or more license servers fail. There are two types: 1) license server list configuration, and 2) three-server configuration.

Reject an action (Hooks)

An action is *rejected* when a hook prevents the action from taking place.

Requeue

The process of stopping a running job and putting it back into the *queued* (“Q”) state.

Rerunnable

If a running PBS job can be terminated and then restarted from the beginning without harmful side effects, the job is rerunnable. The job’s *Rerunnable* attribute must be set to *y* in order for PBS to consider a job to be rerunnable.

Reservation Degradation

PBS attempts to ensure that reservations run by finding usable vnodes when reservation vnodes become unavailable.

Resource

A *resource* can be something used by a job, such as CPUs, memory, high-speed switches, scratch space, licenses, or time, or it can be an arbitrary item defined for another purpose. PBS has built-in resources, and allows custom-defined resources. See [section , "PBS Resources" on page 281 in the PBS Professional Administrator’s Guide.](#)

Restart

A job that was stopped after being checkpointed while previously executing is executed again, starting from the point where it was checkpointed.

Restart File

The job-specific file that is written by the checkpoint script or tool. This file contains any information needed to restart the job from where it was when it was checkpointed.

Restart Script

The script that MOM runs to restart a job. This script is common to all jobs, and so must use the information in a job's restart file to restart the job.

Route a job

When PBS moves a job between queues. PBS provides a mechanism whereby a job is automatically moved from a routing queue to another queue. This is performed by PBS. The resource request for each job in a routing queue is examined, and the job is placed in a destination queue which matches the resource request. The destination queue can be an execution queue or another routing queue.

Routing queue

A queue that serves as a temporary holding place for jobs, before they are moved to another queue. Jobs cannot run from routing queues.

Scheduler

The *scheduler* is the daemon which implements the site's job scheduling policy controlling when and where each job is run. The scheduler is the process called `pbs_sched`.

Scheduling

The process of selecting which jobs to run when and where, according to a predetermined policy. Sites balance competing needs and goals on the system(s) to maximize efficient use of resources (both computer time and people time).

Scheduling policy

Scheduling policy determines when each job runs, and how much of each resource it can use. Scheduling policy consists of a system for determining the priority of each job, combined with a set of limits on how many jobs can be run, and/or how much of each resource can be used.

Schema Admins (Windows)

A group that exists only in the root domain of an Active Directory forest of domains. The group is authorized to make schema changes in Active Directory.

Secondary Scheduler

The PBS Professional scheduler daemon which takes over when the primary scheduler is not available.

Secondary Server

The PBS Professional server daemon which takes over when the primary server fails.

Sequence_number

The numeric part of a job or job array identifier, e.g. **1234**.

Server

The central PBS daemon, which does the following:

- Handles PBS commands
- Receives and creates batch jobs
- Sends jobs for execution

The server is the process called `pbs_server`.

Each PBS complex has one primary server, and if the complex is configured for failover, a secondary server.

The server contains a licensing client which communicates with the licensing server for licensing PBS jobs.

Shared resource

A vnode resource defined and managed at one vnode, but available for use at others.

Shrink-to-fit job

A job that requests the `min_walltime` resource. A shrink-to-fit job requests a running time in a specified range, where `min_walltime` is required, and `max_walltime` is not. PBS computes the actual `walltime`.

Sister

Any MOM that is not on the head or first host of a multihost job. A sister is directed by the Mother Superior. Also called a *subordinate MOM*.

Sisterhood

All of the MOMs involved in running a particular job.

Snapshot Checkpoint

The checkpoint script or tool writes a restart file, and the job continues to execute. The job resumes from this start file if the system experiences a problem during the job's subsequent execution.

Soonest occurrence of a standing reservation

The occurrence which is currently active, or if none is active, then it is the next occurrence.

Stage in

The process of moving one or more job-related files from a storage location to the execution host before running the job.

Stage out

The process of moving one or more job-related files from the execution host to a storage location after running the job.

Staging and execution directory

The *staging and execution directory* is a directory on the execution host where the following happens:

- Files are staged into this directory before execution
- The job runs in this directory
- Files are staged out from this directory after execution

A job-specific staging and execution directory can be created for each job, or PBS can use a specified directory, or a default directory. See [section 12.14.1, "The Job's Staging and Execution Directories" on page 835 in the PBS Professional Administrator's Guide](#).

Standing reservation

An advance reservation which recurs at specified times. For example, the user can reserve 8 CPUs and 10GB every Wednesday and Thursday from 5pm to 8pm, for the next three months.

State

The PBS server, vnodes, reservations, and jobs can be in various states, depending on what PBS is doing. For example the server can be *idle* or *scheduling*, vnodes can be *busy* or *free*, and jobs can be *queued* or *running*, among other states. For a complete description of states, see ["States" on page 411](#).

Strict ordering

A scheduling policy where jobs are run according to policy order. If the site-specified policy dictates a particular priority ordering for jobs, that is the order in which they are run. Strict ordering can be modified by backfilling in order to increase throughput. See ["Backfilling"](#).

Subject

A process belonging to a job run by an authorized, unprivileged user (a job submitter.)

Subjob

One of the jobs in a job array, e.g. 1234 [7], where 1234 [] is the job array itself, and 7 is the index. Queued subjobs are not individually listed

in the queue; only their job array is listed. Running subjobs are individually listed.

Subjob index

The unique index which differentiates one subjob from another. This must be a non-negative integer.

Subordinate MOM

Any MOM that is not on the head or first host of a multihost job. A subordinate MOM is directed by the Mother Superior. Also called a *sister*.

Task

A process belonging to a job. A POSIX session started by MOM on behalf of a job.

Task placement

The process of choosing a set of vnodes to allocate to a job that will both satisfy the job's resource request (select and place specifications) and satisfy the configured scheduling policy.

Three-server Configuration

One form of redundant license server configuration. Means that if any 2 of the 3 license servers are up and running (referred to as a quorum), the system is functional, with 1 server acting as master who can issue licenses. If the master goes down, then another server must take over as master. This is set up as a license file on each of the 3 redundant servers containing:

```
SERVER <server1> ... <port1>
```

```
SERVER <server2> ... <port2>
```

```
SERVER <server3> ... <port3>
```

PBS Professional can point to a license server host that has

Token

Also called “GridWorks Unit”, a unit of value which is checked out from the license server. The number of PBS tokens will be related to the number of CPUs requested by a job that is being executed.

User

Has two meanings:

1. A person who submits jobs to PBS, as differentiated from Operators, Managers and administrators. See [section 9.2.2.1, "User" on page 639 in the PBS Professional Administrator's Guide](#).
2. A system user, identified by a unique character string (the user name) and by a unique number (the user ID). Any person using the system has a username and user ID.

User access, Access by user

The specified user is allowed access at the Server, queues, and reservations .

User ID, UID

A unique numeric identifier assigned to each user.

User limit

Refers to configurable limits on resources and jobs. A limit placed on one or more users, whether generic or individual.

vchunk

The part of a chunk that is supplied by one vnode. If a chunk is broken up across multiple vnodes, each vnode supplies a vchunk.

Version 1 configuration file

MOM configuration file containing MOM configuration parameters. See [Chapter 3, "MOM Parameters", on page 267](#).

Version 2 configuration file

Vnode configuration file containing vnode attribute settings. Created using `pbs_mom -s insert` command.

Virtual processor, VP

PBS can treat a vnode as if it has more processors available than the number of physical processors. When `resources_available.ncpus` is set to a number higher than the actual number of physical processors, the vnode can be said to have virtual processors. Also called logical processors.

Vnode

A virtual node, or *vnode*, is an abstract object representing a set of resources which form a usable part of an execution host. This could be an entire host, or a nodeboard or a blade. A single host can be made up of multiple vnodes. Each vnode can be managed and scheduled independently. Each vnode in a complex must have a unique name. Vnodes can share resources, such as node-locked licenses.

Chapter 2

PBS Commands

This chapter contains a description of each PBS command. Each description includes any options to the command.

2.1 Windows Requirements

Under Windows, use double quotes when specifying arguments to PBS commands.

2.2 Requirements for Commands

Some PBS commands require root privilege or PBS Operator or Manager privilege in order to run. Some can be executed by anyone, but the output depends upon the privilege of the user.

Most PBS commands require that the Server be running; some require that MOMs be running.

The following table lists the commands, and indicates the permissions required to use each, and whether the server or MOM must be running.

Table 2-1: Permission and Daemon Requirements for Commands

Command	Permissions Required	Server Must Be Running ?	MOM Must Be Running ?
<u>"mpiexec"</u>	Any	No	No
<u>"nqs2pbs"</u>	Any	No	No
<u>"pbs"</u>	Users can only get status	No	No
<u>"pbs-report"</u>	Root	No	No
<u>"pbsdsh"</u>	Any	No	Yes
<u>"pbsfs"</u>	Any	No	No
<u>"pbsnodes"</u>	Result depends on permission	Yes	No
<u>"pbsrun"</u>	Can be run by root or PBS administrator only	No	No
<u>"pbsrun_unwrap"</u>	Can be run by root only	No	No
<u>"pbsrun_wrap"</u>	Can be run by root only	No	No
<u>"pbs_account"</u>	Admin on Windows	No	No
<u>"pbs_attach"</u>	Any	Yes	Yes
<u>"pbs_dataservice"</u>	Root on UNIX; Admin on Windows	No	No
<u>"pbs_ds_password"</u>	Root on UNIX; Admin on Windows	No	No
<u>"pbs_hostn"</u>	Any	No	No
<u>"pbs_idled"</u>	Can be run only by root or PBS administrator	No	No

Table 2-1: Permission and Daemon Requirements for Commands

Command	Permissions Required	Server Must Be Running ?	MOM Must Be Running ?
<u>"pbs_lamboot"</u>	Any	No	No
<u>"pbs_migrate_users"</u>	Any	Yes	No
<u>"pbs_mkdirs"</u>	Can be run by PBS administrator only	No	No
<u>"pbs_mom"</u>	Can be run only by root or PBS administrator	No	No
<u>"pbs_mom_globus"</u>	Can be run only by root or PBS administrator	No	No
<u>"pbs_mpihp"</u>	Any	Yes	Yes
<u>"pbs_mpilam"</u>	Any	Yes	Yes
<u>"pbs_mpirun"</u>	Any	Yes	Yes
<u>"pbs_password"</u>	Any	Yes	No
<u>"pbs_probe"</u>	Can be run only by root or PBS administrator	No	No
<u>"pbs_python"</u>	Only to call pbs.server()	No	No
<u>"pbs_rdel"</u>	Any	Yes	No
<u>"pbs_renew"</u>	Any	No	No
<u>"pbs_rstat"</u>	Any	Yes	No
<u>"pbs_rsub"</u>	Any	Yes	No
<u>"pbs_sched"</u>	Can be run only by root or PBS administrator	No	No
<u>"pbs_server"</u>	Can be run only by root or PBS administrator	No	No

Table 2-1: Permission and Daemon Requirements for Commands

Command	Permissions Required	Server Must Be Running ?	MOM Must Be Running ?
<u>"pbs_tclsh"</u>	Any	No	No
<u>"pbs_tmrsh"</u>	Any	No	Yes
<u>"pbs_topologyinfo"</u>	Root	No	No
<u>"pbs_wish"</u>	Any	No	No
<u>"printjob"</u>	Can be run only by root or PBS administrator	No	No
<u>"galter"</u>	Any	Yes	No
<u>"qdel"</u>	Any	Yes	No
<u>"qdisable"</u>	Requires Manager or Operator privilege to run	Yes	No
<u>"genable"</u>	Requires Manager or Operator privilege to run	Yes	No
<u>"qhold"</u>	Some holds can be set by root or administrator only	Yes	No
<u>"qmgr"</u>	Any	Yes	No
<u>"qmove"</u>	Any	Yes	No
<u>"qmsg"</u>	Any	Yes	No
<u>"qorder"</u>	Any	Yes	No
<u>"qrerun"</u>	Any	Yes	No
<u>"qrls"</u>	Some holds can be released by root or administrator only	Yes	No

Table 2-1: Permission and Daemon Requirements for Commands

Command	Permissions Required	Server Must Be Running ?	MOM Must Be Running ?
<u>"grun"</u>	Can be run only by Operator or Manager	Yes	No
<u>"gselect"</u>	Any	Yes	No
<u>"qsig"</u>	Any	Yes	No
<u>"gstart"</u>	Can be run only by Operator or Manager	Yes	No
<u>"gstat"</u>	Result depends on permission	Yes	No
<u>"gstop"</u>	Can be run only by Operator or Manager	Yes	No
<u>"qsub"</u>		Yes	No
<u>"qterm"</u>	Can be run only by Operator or Manager	Yes	No
<u>"tracejob"</u>	Can be run only by root or PBS administrator	No	No
<u>"xpbs"</u>		Yes	No
<u>"xpbsmon"</u>		Yes	No

2.3 mpiexec

Runs MPI programs under PBS on Linux

2.3.1 Synopsis

mpiexec

mpiexec --version

2.3.2 Description

The PBS *mpiexec* command provides the standard *mpiexec* interface on an SGI Altix, ICE or UV running supported versions of ProPack or Performance Suite. If executed on a non-Altix, non-ICE, and non-UV system, it will assume it was invoked by mistake. In this case it will use the value of `PBS_O_PATH` to search for the correct *mpiexec*. If one is found, the PBS *mpiexec* will exec it.

The PBS *mpiexec* calls the SGI `mpirun(1)`. The name of the array to use when invoking `mpirun` is user-specifiable via the `PBS_MPI_SGIARRAY` environment variable.

It is transparent to the user; MPI jobs submitted outside of PBS will run as they would normally. MPI jobs can be launched across multiple SGI machines. PBS will manage, track, and cleanly terminate multi-host MPI jobs. PBS users can run MPI jobs within specific partitions.

If the `PBS_MPI_DEBUG` environment variable's value has a nonzero length, PBS will write debugging information to standard output.

2.3.3 Usage

The PBS *mpiexec* command presents the *mpiexec* interface described in section “4.1 Portable MPI Process Startup” of the “MPI-2: Extensions to the Message-Passing Interface” document in <http://www.mpiforum.org/docs/mpi-2-0-html/node42.htm>

2.3.4 Options

`--version`

The *mpiexec* command returns its PBS version information and exits.
This option can only be used alone.

2.3.5 Requirements

Altix, ICE, or UV running a supported version of ProPack or Performance Suite.

PBS uses SGI's `mpirun(1)` command to launch MPI jobs. SGI's `mpirun` must be in the standard location.

The location of `pbs_attach()` on each node of a multinode MPI job must be the same as it is on the mother superior node.

In order to run multihost jobs, the SGI Array Services must be correctly configured. SGI systems communicating via SGI's Array Services must all use the same version of the `sgi-arraysvcs` package. SGI systems communicating via SGI's Array Services must have been configured to interoperate with each other using the default array. See SGI's `array_services(5)` man page.

2.3.6 Environment Variables

The PBS `mpiexec` script sets the `PBS_CPUSSET_DEDICATED` environment variable to assert exclusive use of the resources in the assigned cpuset.

The PBS `mpiexec` checks the `PBS_MPI_DEBUG` environment variable. If this variable has a nonzero length, debugging information is written.

If the `PBS_MPI_SGIARRAY` environment variable is present, the PBS `mpiexec` will use its value as the name of the array to use when invoking `mpirun`.

The `PBS_ENVIRONMENT` environment variable is used to determine whether `mpiexec` is being called from within a PBS job.

The PBS `mpiexec` uses the value of `PBS_O_PATH` to search for the correct `mpiexec` if it was invoked by mistake.

2.3.7 Path

PBS' `mpiexec` is located in `PBS_EXEC/bin/mpiexec`.

2.3.8 See Also

The PBS Professional Administrator's Guide

SGI man pages: SGI's `mpirun(1)`, SGI's `mpiexec_mpt(1)`, SGI's `array_services(5)`

PBS man pages: `pbs_attach(8B)`

2.4 nqs2pbs

Converts NQS job scripts to PBS format.

2.4.1 Synopsis

```
nqs2pbs nqs_script [pbs_script]
nqs2pbs --version
```

2.4.2 Description

This utility converts an existing NQS job script to work with PBS and NQS. The existing script is copied and PBS directives, #PBS , are inserted prior to each NQS directive #QSUB or #@ \$, in the original script.

Certain NQS date specification and options are not supported by PBS. A warning message will be displayed indicating the problem and the line of the script on which it occurred.

If any unrecognizable NQS directives are encountered, an error message is displayed. The new PBS script will be deleted if any errors occur.

2.4.3 Options

--version

The nqs2pbs command returns its PBS version information and exits.
This option must be used alone.

2.4.4 Operands

nqs_script

Specifies the file name of the NQS script to convert. This file is not changed.

pbs_script

If specified, it is the name of the new PBS script. If not specified, the new file name is nqs_script.new .

NOTES

Converting NQS date specifications to the PBS form may result in a warning message and an incompletely converted date. PBS does not support date specifications of “*today*”, “*tomorrow*”, or the name of the days of the week such as “*Monday*”. If any of these are encountered in a script, the PBS specification will contain only the time portion of the NQS specification, i.e. #PBS -a hhmm[.ss]. It is suggested that you specify the execution time on the qsub command line rather than in the script.

Note that PBS will interpret a time specification without a date in the following way:

- If the time specified has not yet been reached, the job will become eligible to run at that time today.
- If the specified time has already passed when the job is submitted, the job will become eligible to run at that time tomorrow.

This command does not support time zone identifiers. All times are taken as local time.

2.4.5 See Also

`qsub(1B)`

2.5 pbs-report

Prints PBS job statistics.

2.5.1 Synopsis

```
pbs-report [--age seconds[:offset]] [--account account] [--begin -b yyyyymmdd[:hhmm[ss]]]
  [--count -c] [--cpumax seconds] [--cpumin seconds] [--csv character] [--dept
  department] [--end -e yyyyymmdd[:hhmm[ss]]] [--exit -x integer] [--explainwait] [--
  group UNIX group] [--help] [--host hostname] [--inclusive] [--index key] [--man] [--
  negate option] [--package solver] [--point yyyyymmdd[:hhmm[ss]]] [--queue PBS queue]
  [--range span] [--reslist] [--sched] [--sort field] [--time option] [--user username] [--
  verbose] [--vsort field] [--waitmax seconds] [--waitmin seconds] [--wall] [--wallmax
  seconds] [--wallmin seconds]
```

```
pbs-report --version
```

2.5.2 Description

Allows the PBS Administrator to generate a report of job statistics from the PBS accounting logfiles. Options to the `pbs-report` command control how jobs are selected for reporting and how reports are generated.

The `pbs-report` command is not available on Windows.

Before first using `pbs-report`, the Administrator is advised to tune the configuration to match the local site by editing the file `PBS_EXEC/lib/pm/PBS.pm`.

2.5.2.1 Permissions

This command can be run only by root.

2.5.2.2 Selecting Jobs For Reporting

2.5.2.2.i Filtering Jobs by Dates or Times

--begin, --end, --range, --age, --point

--begin and --end

Work from hard date limits. Omitting either will cause the report to contain all data to either the beginning or the end of the accounting data.

Unbounded date reports may take several minutes to run, depending on the volume of work logged.

Jobs are listed by start time only, regardless of whether end time is specified via **--end** or **--inclusive**.

--range

A short-hand way of selecting a prior date range and will supersede **--begin** and **--end**.

--age

Allows the user to select an arbitrary period going back a specified number of seconds from the time the report is run. **--age** will silently supersede all other date options.

--point

Displays all jobs which were running at the specified point in time, and is incompatible with the other options. **--point** will produce an error if specified with any other date-related option.

2.5.2.2.ii Filtering Jobs by Attribute

--cpumax, --cpumin, --waitmax, --waitmin, --wallmax, --wallmin

A maximum value will cause any jobs with more than the specified amount to be ignored. A minimum value will cause any jobs with less than the specified amount to be ignored. All six options may be combined, though doing so will often restrict the filter such that no jobs can meet the requested criteria. Combine time filters for different time with caution.

2.5.2.2.iii Filtering Jobs by User or Department

--dept, --group, --user

--dept

Allows for integration with an LDAP server and will generate reports based on department codes as queried from that server. If no LDAP server is available, department-based filtering and sorting will not function.

--group

Allows for filtering of jobs by primary group ownership of the submitting user, as defined by the operating system on which the PBS server runs.

--user

Allows for explicit naming of users to be included.

It is possible to specify a list of values for these filters, by providing a single colon-concatenated argument or using the option multiple times, each with a single value.

2.5.2.2.iv Filtering Jobs by Job Property

--host, --exit, --package, --queue

--host

Allows for filtering of jobs based on the host on which the job was executed.

--exit

Allows for filtering of jobs based on the job exit code.

--package

Allows for filtering of jobs based on the software package used in the job. This option will only function when a package-specific custom resource is defined for the PBS server and requested by the jobs as they are submitted.

--queue

Allows for filtering of jobs based on the queue in which the job finally executed. With the exception of `--exit`, it is possible to specify a list of values for these filters, by providing a single colon-concatenated argument or using the option multiple times, each with a single value.

2.5.2.2.v Filtering Jobs by Account String

--account

This option allows the user to filter jobs based on an arbitrary, user-specified job account string. The content and format of these strings is site-defined and unrestricted; it may be used by a custom job front-end which enforces permissible account strings, which are passed to `qsub` with `qsub's -A` option.

2.5.2.2.vi Negating Filters

--negate

The **--negate** option allows for logical negation of one or more specified filters. Only the account, dept, exit, group, host, package, queue, and user filters may be negated. If a user is specified with **--user**, and the '**--negate user**' option is used, only jobs not belonging to that user will be included in the report. Multiple report filters may be negated by providing a single colon-concatenated argument or using **--negate** multiple times, each with a single value.

2.5.2.3 Generating Reports

Several report types can be generated, each indexed and sorted according to the user's needs.

--verbose

Generates a wide tabular output with detail for every job selected. It can be used to generate output for import to a spreadsheet. Verbose reports may be sorted on any field using the **--vsort** option. Default: summary report only.

--reslist

Generates a tabular output with detail on resources requested for every job selected. Resource list reports may be sorted on any field using the **--vsort** option. Default: summary report only.

--inclusive

Allows a user to require that the job's start time also falls within the date range.

Jobs are listed by start time only, regardless of whether end time is specified via **--end** or **--inclusive**.

--index

Allows specification of the field on which data in the summary should be grouped. Fields listed in the option description are mutually exclusive. The selected field will be the left-most column of the summary report output. One value may be selected as an index while another is selected for sorting. However, since index values are mutually exclusive, the only sort options which may be used (other than the index itself) are account, cpu, jobs, suspend, wait, and wall. If no sort order is selected, the index is used as the sort key for the summary.

--sort

Allows the user to specify a field on which to sort the summary report. It operates independently of the sort field for verbose reports (see **--vsort**). See the description for **--index** for how the two options interact.

--vsort

Allows the user to specify a field on which to sort the verbose report. It operates independently of the sort field for summary reports (see **--sort**).

2.5.3 Options to **pbs-report**

--age -a seconds[:offset]

Report age in seconds. If an offset is specified, the age range is taken from that offset backward in time, otherwise a zero offset is assumed. The time span is from (now - age - offset) to (now - offset). This option silently supersedes **--begin**, **--end**, and **--range**.

--account account

Limit results to those jobs with the specified account string. Multiple values may be concatenated with colons or specified with multiple instances of **--account**.

--begin -b yyyyymmdd[:hhmm[ss]]

Report begin date and optional time. Default: most recent log data. **--begin** and **--end** work from hard date limits. Omitting either will cause the report to contain all data to either the beginning or the end of the accounting data. Unbounded date reports may take several minutes to run, depending on the volume of work logged.

Jobs are listed by start time only, regardless of whether end time is specified via **--end** or **--inclusive**.

--count -c

Display a numeric count of matching jobs. Currently only valid with **--cpumax** for use in monitoring rapidly-exiting jobs.

--cpumax seconds

Filter out any jobs which have more than the specified number of CPU seconds.

--cpumin seconds

Filter out any jobs which have less than the specified number of CPU seconds.

--dept -d department

Limit results to those jobs whose owners are in the indicated department . Default: any. This option only works in conjunction with an LDAP server which supplies department codes. See also the **--group** option. Multiple values may be concatenated with colons or specified with multiple instances of **--dept**.

--end -e yyyyymmdd[:hhmm[ss]]

Report end date and optional time. Default: most recent log data. **--begin** and **--end** work from hard date limits. Omitting either will cause the report to contain all data to either the beginning or the end of the accounting data. Unbounded date reports may take several minutes to run, depending on the volume of work logged.

Jobs are listed by start time only, regardless of whether end time is specified via **--end** or **--inclusive**.

--exit -x integer

Limit results to jobs with the specified exit status. Default: any.

--explainwait

Print a reason for why jobs had to wait before running.

--group -g group

Limit results to the specified group name. Group is defined by the operating system. Multiple values may be concatenated with colons or specified with multiple instances of **--group**.

--help -h

Prints a brief help message and exits.

--host -m execution host

Limit results to the specified execution host. Multiple values may be concatenated with colons or specified with multiple instances of **--host**.

--inclusive key

Limit results to jobs which had start times in the range.

Jobs are listed by start time only, regardless of whether end time is specified via **--end** or **--inclusive**.

--index -i key

Field on which to index the summary report. Default: *user*. Valid values include: *date*, *dept*, *host*, *package*, *queue*, *user*.

--man

Prints the manual page and exits.

--negate -n option

Logically negate the selected options; print all records except those that match the values for the selected criteria. Default: unset. Valid values: *account*, *dept*, *exit*, *group*, *host*, *package*, *queue*, *user*. Defaults cannot be negated, only options explicitly specified are negated. Multiple values may be concatenated with colons or specified with multiple instances of **--negate**.

--package -p package

Limit results to the specified software package. Multiple values may be concatenated with colons or specified with multiple instances of **--package**. Valid values are can be seen by running a report with the **--index** package option. This option keys on custom resources requested at job submission time. Sites not using such custom resources will have all jobs reported under the catchall None package with this option.

--point yyyyymmdd[:hhmm[ss]]

Print a report of all jobs which were actively running at the point in time specified. This option cannot be used with any other date or age option.

--queue -q queue

Limit results to the specified queue. Multiple values may be concatenated with colons or specified with multiple instances of **--queue**. Note that if specific queues are defined via the **@QUEUES** line in **PBS.pm**, then only those queues will be displayed. Leaving that parameter blank allows all queues to be displayed.

--range -r period

Time period used is period before now. For example, if the period given is “week”, **pbs-report** looks at all jobs which have finished and which were running any time from a week ago to now. Default: all. Valid values for period are *today*, *week*, *month*, *quarter*, and *year*. This option silently supersedes **--begin** and **--end**, and is superseded by **--age**.

--reslist

Include resource requests for all matching jobs. This option is mutually exclusive with **--verbose**.

--sched -t

Generate a brief statistical analysis of Scheduler cycle times. No other data on jobs is reported.

--sort -s field

Field by which to sort reports. Default: user. Valid values are *cpu*, *date*, *dept*, *host*, *jobs*, *package*, *queue*, *suspend* (aka muda), *wait*, and *wall*.

--time option

Valid values: “*full*”, “*partial*”. Used to indicate how time should be accounted. The default of “*full*” means that entire job’s CPU and wall time is counted in the report if the job ended during the report’s date range. With the “*partial*” option, only CPU and wall time during the report’s date range are counted.

By default, time is credited at the point when the job ended. This can be changed using the **--inclusive** option. For a job which ended a few seconds

after the report range begins, this can cause significant overlap, which may boost results. During a sufficiently large time frame, this overlap effect is negligible and may be ignored. This value for `--time` should be used when generating monthly usage reports. With “partial”, any CPU or wall time accumulated prior to the beginning of the report is ignored. “*partial*” is intended to allow for more accurate calculation of overall cluster efficiency during short time spans during which a significant overlap effect can skew results. See `--inclusive`.

--user -u username

Limit results to the specified username. Multiple values may be concatenated with colons or specified with multiple instances of `--user`.

--verbose -v

Include attributes for reported jobs. Subjobs are shown, but not job arrays. Default: no attributes.

--version

The `pbs-report` command returns its PBS version information and exits. This option can only be used alone.

--vsort field

Field by which to sort the verbose output section reports. Default: `jobid`. Valid values are *cpu*, *date*, *exit*, *host*, *jobid*, *jobname*, *mem*, *name*, *package*, *queue*, *scratch*, *suspend*, *user*, *vmem*, *wall*, *wait*. If neither `--verbose` nor `--reslist` is specified, `--vsort` is silently ignored. The scratch sort option is available only for resource reports (`--reslist`).

--waitmax seconds

Filter out any jobs which have more than the specified wait time in seconds.

--waitmin seconds

Filter out any jobs which have less than the specified wait time in seconds.

--wallmax seconds

Filter out any jobs which have more than the specified wall time in seconds.

--wallmin seconds

Filter out any jobs which have less than the specified wall time in seconds.

--wall -w

Use the `walltime` resource attribute rather than wall time calculated by subtracting the job start time from end time. The `walltime` resource attribute does not accumulate when a job is suspended for any reason, and thus may not accurately reflect the local interpretation of wall time.

2.5.4 Examples

“How much in the way of resources did every job this month waiting more than 10 minutes request?”

pbs-report --range month --waitmin 600 --reslist

This information might be valuable to determine if some simple resource additions (e.g. more memory or more disk) might increase overall throughput of the cluster.

2.5.4.1 Statistical Analysis

At the bottom of the summary statistics, prior to the job set summary, is a statistical breakdown of the values in each column. Example:

	# of	Total	Total		Average
Date	jobs	CPU Time	Wall Time	Efcy.	Wait Time
-----	-----	-----	-----	-----	-----
TOTAL	1900	10482613	17636290	0.594	1270
Minimum	4	4715	13276	0.054	221
Maximum	162	1399894	2370006	1.782	49284
Mean	76	419304	705451	0.645	2943
Deviation	41	369271	616196	0.408	9606
Median	80	242685	436724	0.556	465

This summary should be read in column format. The values each represent a statistical data point in the column. For instance, while the minimum number of jobs run in one day was 4 and the maximum 162, these values do not correlate to the 4715 and 1399894 CPU seconds listed as minima and maxima.

In the Job Set Summary section, the values should be read in rows, as shown here:

	Standard				
	Minimum	Maximum	Mean	Deviation	Median
	-----	-----	----	-----	-----
CPU time	0	18730	343	812	0
Wall time	0	208190	8496	19711	93
Wait time	0	266822	4129	9018	3

These values represent aggregate statistical analysis for the entire set of jobs included in the report. The values in the prior summary represent values over the set of totals based on the summary index (e.g. Maximum and Minimum are the maximum and minimum totals for a given day/user/department, rather than an individual job. The job set summary represents an analysis of all individual jobs.

2.5.4.2 Cluster Monitoring

The `--count` and `--cpumax` functions are intended to allow an administrator to periodically run this script to monitor for jobs which are exiting rapidly, representing a potential global error condition causing all jobs to fail. It is most useful in conjunction with `--age`, which allows a report to span an arbitrary number of seconds backward in time from the current moment. A typical set of options would be “`--count --cpumax 30 --age 21600`”, which would show a total number of jobs which consumed less than 30 seconds of CPU time within the last six hours.

2.5.5 Standard Error

The `pbs-report` command will write a diagnostic message to standard error for each error occurrence.

2.5.6 Exit Status

Zero upon successful processing of all operands.

Greater than zero if the `pbs-report` command fails to process any operand.

2.5.7 See Also

The PBS Professional Administrator's Guide, `pbs_server(8B)`, `pbs_sched(8B)`, `pbs_mom(8B)`

2.6 pbs_account

Manage PBS service account

2.6.1 Synopsis

```
pbs_account [-a PBS service account name] [-c] [-s] [-p password] [--reg service_path] [--unreg service_path] [-o output_path] [--ci]
```

2.6.2 Description

The `pbs_account` command is used to manage the PBS service account. It is used to create the account, set or validate the account password, add privileges to the account, and register or unregister the account with the SCM.

2.6.3 Permissions

This command can be run by administrators only.

2.6.4 Platforms

This command is available on Windows only.

2.6.5 Options

- a <account name>**
Specifies account name.
- c [<password>]**
 - If specified account does not exist, creates the account with the password.
 - If specified account exists, validates password against it.

If password is not specified, user is prompted for password.
- o <output path>**
Prints `stdout` and `stderr` messages in specified output path
- p [<password >]**
Updates the PBS service account password. If no password is specified, the user is prompted for a password.
- s**
Adds necessary privileges to the PBS service account. Grants the "Create Token Object", "Replace Process Level Token", "Log On as a Service", and "Act as Part of the Operating System" privileges to PBS service account.

```
--reg-server <path to server>    [<password>]
--reg-mom   <path to MOM>        [<password>]
--reg-sched <path to scheduler>  [<password>]
--reg-rshd  <path to rshd>       [<password>]
        Registers the PBS service with the SCM, instructing it to run the services
        under the PBS service account and supplied password. <path> must be in
        double quotes.

--unreg-server <path to server>
--unreg-mom   <path to MOM>
--unreg-sched <path to scheduler>
--unreg-rshd  <path to rshd>
        Unregisters the PBS service with the SCM. <path> must be in double
        quotes.

--ci
        Prints actions taken by pbs_account while creating PBS service account
        when -c option is used

<no options>
        Prints name of PBS service account, if it exists. Exit value is 0.
```

2.6.6 Examples

- To create the PBS service account:
pbs_account -c -s -p password
- To change the PBS service account:
pbs_account [--reg service_path] [- a PBS service account name]
- To register the server, scheduler, MOM, and rshd services:
**pbs_account --reg "\Program Files\PBS Pro\exec\sbin\pbs_server.exe" [-p
 <password>]**
**pbs_account --reg "\Program Files\PBS Pro\exec\sbin\pbs_mom.exe" [-p
 <password>]**
**pbs_account --reg "\Program Files\PBS Pro\exec\sbin\pbs_sched.exe" [-p
 <password>]**
**pbs_account --reg "\Program Files\PBS Pro\exec\sbin\pbs_rshd.exe" [-p
 <password>]**

2.7 pbs_attach

Attaches a session ID to a PBS job

2.7.1 Synopsis

```
pbs_attach [-j jobid] [-m port] -p pid
pbs_attach [-j jobid] [-m port] [-P] [-s] cmd [arg ...]
pbs_attach --version
```

2.7.2 Description

The `pbs_attach` command associates the processes in a session with a PBS job by attaching the session ID to the job. This allows PBS MOM to monitor and control those processes.

MOM uses process IDs to determine session IDs, which are put into MOM's task list (attached to the job.) All process IDs in a session are then associated with the job.

When a command `cmd` is given as an operand, the `pbs_attach` process becomes the parent process of `cmd`, and the session ID of `pbs_attach` is attached to the job.

The `-p` option cannot be used with the `-P` or `-s` options or the `cmd` operand.

This command is not supported under Windows.

2.7.3 Options to pbs_attach

- `-j jobid`
The job ID to which the session ID is to be attached. If `jobid` is not specified, a best effort will be made to determine the job to which to attach the session.
- `-m port`
The port at which to contact MOM. Default: value of `PBS_MANAGER_SERVICE_PORT` from `pbs.conf`
- `-p pid`
Process ID whose session ID will be attached to the job. Default: process ID of `pbs_attach`.
- `-P`
Attach sessions of both `pbs_attach` and the parent of `pbs_attach` to job. When used with `-s` option, this means the sessions of the new `fork()` ed

-
- `pbs_attach` and its parent, which is `pbs_attach`, are attached to the job.
- s** Starts a new session by `fork ()`-ing `pbs_attach`. The session ID of the new `pbs_attach` is attached to the job.
- version** The `pbs_attach` command returns its PBS version information and exits. This option can only be used alone.

2.7.4 Operands

- cmd** Name of command whose process ID is to be associated with the job.

2.7.5 Exit Status

- 0** Success
- 1** Any error following successful command line processing. A message is printed to standard error.

If *cmd* is specified, `pbs_attach` waits for *cmd* to exit, then exits with the exit value of *cmd*.

If *cmd* is not specified, `pbs_attach` exits after attaching the session ID(s) to the job.

2.7.6 See Also

The PBS Professional Administrator's Guide, `pbs_mom(8B)`, `pbs_tmsh(8B)`, `tm(3)`

2.8 pbs_datservice

Start, stop, or check the status of PBS data service

2.8.1 Synopsis

pbs_datservice [start|stop|status]

2.8.2 Description

The `pbs_dataservice` command starts, stops or gets the status of the PBS data service.

2.8.3 Permission

On UNIX, root privilege is required to use this command. On Windows, Admin privilege is required.

2.8.4 Arguments

<code>start</code>	Starts the PBS data service.
<code>stop</code>	Stops the PBS data service. Can be used only when PBS server is not running.
<code>status</code>	Displays the status of the PBS data service, as follows: <ul style="list-style-type: none"> Data service running PBS Data Service running Data service not running PBS Data Service not running

2.8.5 Exit Status

Zero for success

Non-zero for failure

2.9 pbs_ds_password

Sets or changes data service user account or its password

2.9.1 Synopsis

pbs_ds_password [-r] [-C username]

2.9.2 Description

You can use this command to change the user account or account password for the data service. Blank passwords are not allowed.

2.9.3 Permissions

On UNIX, root privilege is required to use this command. On Windows, Admin privilege is required.

2.9.4 Restrictions

Do not run this command if failover is configured. It is important not to inadvertently start two separate instances of the data service on two machines, thus potentially corrupting the database.

2.9.5 Options to `pbs_ds_password`

-C <username>

Changes user account for data service to specified account. Specified user account must already exist.

On UNIX-based systems, the specified user account must not be root.

On Windows, the specified user account must match the PBS service account (which can be any user account.)

This option cannot be used while the data service is running.

Can be used with the `-r` option to automatically generate a password for the new account.

-r

Generates a random password. The data service is updated with the new password.

Can be used with the `-C` option.

<no option>

Asks the user to enter a new password twice. Entries must match. Updates data service with new password.

2.9.6 Exit Status

Zero on success.

Non-zero on failure.

2.10 pbs_hostn

Reports hostname and network address(es)

2.10.1 Synopsis

pbs_hostn [-v] hostname

pbs_hostn --version

2.10.2 Description

The `pbs_hostn` command takes a hostname, and reports the results of both `gethostbyname(3)` and `gethostbyaddr(3)` system calls. Both forward and reverse lookup of hostname and network addresses need to succeed in order for PBS to authenticate a host.

Running this command can assist in troubleshooting problems related to incorrect or non-standard network configuration, especially within clusters.

2.10.3 Options

-v

Turns on verbose mode.

--version

The `pbs_hostn` command returns its PBS version information and exits. This option can only be used alone.

2.10.4 Operands

The `pbs_hostn` command accepts a hostname operand either in short name form, or in fully qualified domain name (FQDN) form.

2.10.5 Standard Error

The `pbs_hostn` command will write a diagnostic message to standard error for each error occurrence.

2.10.6 Exit Status

Zero upon successful processing of all the operands presented to the `pbs_hostn` command.

Greater than zero if the `pbs_hostn` command fails to process any operand.

2.10.7 See Also

The PBS Professional Administrator's Guide and the following manual page:
`pbs_server(8B)`

2.11 `pbs_idled`

PBS daemon to watch the console and inform `pbs_mom` of idle time

2.11.1 UNIX/Linux Synopsis

```
pbs_idled [-w wait_time] [-f idle_file] [-D display] [-r reconnect_delay]  
pbs_idled --version
```

2.11.2 Windows Synopsis

```
pbs_idled [start | stop]  
pbs_idled --version
```

2.11.3 UNIX/Linux Description

On UNIX/Linux, the `pbs_idled` program sits and watches an X windows display and communicates the idle time of the display back to PBS. If the mouse is moved or a key is touched, PBS is informed that the node is busy.

This program should be run out of the system-wide Xsession file. It should be run in the background before the window manager is run. If this program is run outside of the Xsession, it will need to be able to make a connection to the X display. See the xhost or xauth man pages for a description of X security.

2.11.4 Windows Description

On Windows, `pbs_idled` reads its polling interval from a file called `idle_poll_time` which is created by MoM. The process monitors keyboard, mouse, and console activity, and updates a file called `idle_touch` when it finds user activity. The `idle_touch` file is created by MoM.

2.11.5 UNIX/Linux Options to `pbs_idled`

- `-w <wait_time>`
Granularity between when the daemon checks for events or pointer movement.
- `-f <idle_file>`
Update file times on `<idle_file>`. PBS will not monitor any other than the default.
- `-D <display>`
The display to connect to and monitor.
- `-r <reconnect_delay>`
The amount of time to try and reconnect to the X display if the previous attempt was unsuccessful.
- `--version`
The `pbs_idled` command returns its PBS version information and exits. This option can only be used alone.

2.11.6 Windows Options to `pbs_idled`

- `start`
Starts the `pbs_idled` process.
- `stop`
Stops the `pbs_idled` process.
- `--version`
The `pbs_idled` process returns its PBS version information and exits. This option can only be used alone.

2.11.7 See Also

The PBS Professional Administrator's Guide and the following manual pages:

`pbs_mom(8B)`, `xhost(1)`, `xauth(1)`

2.12 pbs

Start, stop, restart, or get the PIDs of PBS daemons

2.12.1 Synopsis

pbs [start|stop|restart|status]

2.12.2 Description

The `pbs` command starts, stops or restarts all PBS daemons on the local machine. Does not affect other hosts. It also reports the PIDs of all daemons when given the status argument.

2.12.2.1 Caveats

This command operates only on daemons that are marked as active in `pbs.conf`. For example, if `PBS_START_MOM` is set to `0` in the local `pbs.conf`, this command will not operate on `pbs_mom`, and will not start, stop, or restart `pbs_mom`.

2.12.2.2 Privilege

PBS Manager privilege is required to use this command.

2.12.3 Arguments

`start`

Each daemon on the local machine is started. PBS reports the number and type of licenses available, as well as the name of the license server. Any running jobs are killed.

`stop`

Each daemon on the local machine is stopped, and its PID is reported.

restart

All daemons on the local machine are stopped, then they are restarted. PBS reports the name of the license server and the number and type of licenses available.

status

PBS reports the PID of each daemon on the local machine.

2.12.4 See Also

The PBS Professional Administrator's Guide, `pbs_mom(8B)`, `pbs_server(8B)`, `pbs_sched(3)`

2.13 pbs_interactive

Register, unregister, or get the version of PBS_INTERACTIVE service

2.13.1 Synopsis

pbs_interactive [R | U]

pbs_interactive --version

2.13.2 Description

The `pbs_interactive` process is used to register or unregister the Windows PBS_INTERACTIVE service. The service must be registered manually; the installer does not register it.

On Windows, the PBS_INTERACTIVE service itself monitors logging in and out by users, starts a `pbs_idled` process for each user logging in, and stops the `pbs_idled` process of each user logging out.

2.13.2.1 Privilege

Administrator privilege is required to use this command.

2.13.3 Arguments

R

Registers the PBS_INTERACTIVE service.

U

Unregisters the PBS_INTERACTIVE service.

--version

The `pbs_interactive` command returns its PBS version information and exits. This option can only be used alone.

2.14 pbs_lamboot

PBS front end to LAM's `lamboot` program

2.14.1 Synopsis

pbs_lamboot

pbs_lamboot --version

2.14.2 Description

The PBS command `pbs_lamboot` replaces the standard `lamboot` command in a PBS LAM MPI job, for starting LAM software on each of the PBS execution hosts running Linux 2.4 or higher.

Usage is the same as for LAM's `lamboot`. All arguments except for `bhost` are passed directly to `lamboot`. PBS will issue a warning saying that the `bhost` argument is ignored by PBS since input is taken automatically from `$PBS_NODEFILE`. The `pbs_lamboot` program will not redundantly consult the `$PBS_NODEFILE` if it has been instructed to boot the nodes using the `tm` module. This instruction happens when an argument is passed to `pbs_lamboot` containing “-ssi boot tm” or when the `LAM_MPI_SSI_boot` environment variable exists with the value *tm*.

2.14.3 Options

`--version`

The `pbs_lamboot` command returns its PBS version information and exits. This option can only be used alone.

2.14.4 Operands

The operands for `pbs_lamboot` are the same as for `lamboot`.

2.14.5 Environment Variables and Path

The `PATH` on remote machines must contain `PBS_EXEC/bin`.

2.14.6 See Also

The PBS Professional Administrator's Guide, `lamboot(1)`, `tm(3)`

2.15 `pbs_migrate_users`

Transfers per-user or per-server passwords between PBS servers during a migration upgrade

2.15.1 Synopsis

`pbs_migrate_users old_server new_server`

`pbs_migrate_users --version`

2.15.2 Description

The `pbs_migrate_users` command is used to transfer the per-user or per-server password of a PBS user from one server to another during a migration upgrade.

Users' passwords on the old server are not deleted.

Available on Windows and supported Linux x86 and x86_64 platforms only.

2.15.3 Options

`--version`

The `pbs_migrate_users` command returns its PBS version information and exits. This option can only be used alone.

2.15.4 Operands

The format of *old_server* and *new_server* is

hostname[:port_number]

2.15.5 Exit Status

0

Success

-1

Writing out passwords to files failed.

-2

Communication failure between *old_server* and *new_server*.

-3

`Single_signon_password_enable` not set in either *old_server* or *new_server*

-4

User running `pbs_migrate_users` not authorized to migrate users.

2.15.6 See Also

`pbs_password(8B)`

2.16 pbs_mkdirs

Create, or fix the permissions of, the directories and files used by PBS

2.16.1 Synopsis

pbs_mkdirs

pbs_mkdirs [mom | sched | server]

2.16.2 Description

Runs on Windows only. If the directories and files used by PBS exist, the `pbs_mkdirs` command fixes their permissions. If the directories and/or files do not exist, the `pbs_mkdirs` command creates them, with the correct permissions. The `pbs_mkdirs` command always examines the following directories and files:

- `pbs.conf`
- `PBS_EXEC`
- `PBS_HOME/spool`
- `PBS_HOME/undelivered`
- `PBS_HOME/pbs_environment`

2.16.3 Options

mom

The `pbs_mkdirs` command examines the following additional items:

- `PBS_HOME/mom_priv`
- `PBS_HOME/mom_logs`

sched

The `pbs_mkdirs` command examines the following additional items:

- `PBS_HOME/sched_priv`
- `PBS_HOME/sched_logs`

server

The `pbs_mkdirs` command examines the following additional items:

- `PBS_HOME/server_priv`
- `PBS_HOME/server_logs`

(no options)

The `pbs_mkdirs` command examines all of the files and directories specified for each of the `mom`, `server`, and `sched` options.

2.16.4 See Also

The PBS Professional Administrator's Guide, `pbs_probe(8B)`

2.17 `pbs_mom`

The PBS job monitoring and execution daemon

2.17.1 Synopsis

```
pbs_mom [-a alarm_timeout] [-C checkpoint_directory] [-c config_file] [-d home_directory]
        [-L logfile] [-M TCP_port] [-n nice_val] [-N] [-p|-r] [-R UDP_port] [-S server_port] [-s script_options]
```

```
pbs_mom --version
```

2.17.2 Description

The `pbs_mom` command starts the PBS job monitoring and execution daemon, called *MOM*. A special HPC Basic Profile MOM manages jobs for an HPC Basic Profile Server. This MOM is called the HPCBP MOM; the standard MOM is called the MOM. See HPCBP MOM below.

The standard MOM starts jobs on the execution host, monitors and reports resource usage, enforces resource usage limits, and notifies the server when the job is finished. The MOM also runs any prologue scripts before the job runs, and runs any epilogue scripts after the job runs.

The MOM performs any communication with job tasks and with other MOMs. The MOM on the first vnode on which a job is running manages communication with the MOMs on the remaining vnodes on which the job runs.

The MOM manages one or more vnodes. PBS may treat a host such as an Altix as a set of virtual nodes, in which case one MOM would manage all of the host's vnodes. See the PBS Professional Administrator's Guide.

The MOM's log file is in `PBS_HOME/mom_logs`. The MOM writes an error message in its log file when it encounters any error. If it cannot write to its log file, it writes to standard error. The MOM will write events to its log file. The MOM writes its PBS version and build information to the logfile whenever it starts up or the logfile is rolled to a new file.

The executable for `pbs_mom` is in `PBS_EXEC/sbin`, and can be run only by root.

2.17.2.1 HPCBP MOM

The HPCBP MOM acts as an intermediary between the PBS complex and the HPCBP Server. The HPCBP MOM does the following:

- Converts between formats used by PBS and HPCBP for information going both to and from the HPC Basic Profile Server.
- Takes job requests from the PBS complex, converts them to JSDL, and submits these converted jobs to the HPC Basic Profile Server.
- Retrieves job and node status information from the HPCBP Server and hands this status information to the PBS Server.

One HPCBP MOM is required for each HPCBP Server.

The HPCBP MOM is created by setting the HPCBP-specific vnode attributes on the vnode that will host the HPCBP MOM. These are the attributes prefixed by “`hpcbp_`”. See the `pbs_node_attributes(7B)` man page.

Many of the configuration variables for `pbs_mom` behave differently in the HPCBP MOM. See the PBS Professional Administrator’s Guide.

2.17.2.2 Cpusets

A cpusetted machine can have a *boot cpuset* defined by the administrator. A boot cpuset contains one or more CPUs and memory boards and is used to restrict the default placement of system processes, including login. If defined, the boot cpuset will contain CPU 0.

Run parallel jobs exclusively within a cpuset for repeatability of performance. SGI states, “Using cpusets on an Altix system improves cached locality and memory access times and can substantially improve an application’s performance and runtime repeatability.”

The `CPUSET_CPU_EXCLUSIVE` flag will prevent CPU 0 from being used by the MOM in the creation of job cpusets. This flag is set by default, so this is the default behavior.

To find out which cpuset is assigned to a running job, use `qstat -f` to see the cpuset field in the job’s `altid` attribute.

2.17.2.2.i Altix Running Supported Versions of ProPack or Performance Suite

The cpusets created for jobs are marked `cpu-exclusive`.

MOM does not use any CPU which was in use at startup.

A PBS job can run across multiple Altixes that run supported versions of ProPack or Performance Suite.

PBS can run using SGI's MPI (MPT) over InfiniBand. See the PBS Professional Administrator's Guide.

2.17.2.3 Effect on Jobs of Starting MOM

When MOM is started or restarted, her default behavior is to leave any running processes running, but to tell the PBS server to requeue the jobs she manages. MOM tracks the process ID of jobs across restarts.

In order to have all jobs killed and requeued, use the `r` option when starting or restarting MOM.

In order to leave any running processes running, and not to requeue any jobs, use the `p` option when starting or restarting MOM.

2.17.3 Options to `pbs_mom`

`-a alarm_timeout`

Number of seconds before alarm timeout. Whenever a resource request is processed, an alarm is set for the given amount of time. If the request has not completed before `alarm_timeout`, the OS generates an alarm signal and sends it to MOM.

Format: Integer

Default: *10 seconds*

`-C checkpoint_directory`

Specifies the path to the directory where MOM creates job-specific subdirectories used to hold each job's restart files. MOM passes this path to checkpoint and restart scripts. Overrides other checkpoint path specification methods. Any directory specified with the `-C` option must be owned, readable, writable, and executable by root only (*rwX,---,---*, or *0700*), to protect the security of the restart files. See the `-d` option to `pbs_mom` and [section 10.3.6.5, "Specifying Checkpoint Path" on page 720 in the PBS Professional Administrator's Guide](#).

Format: *String*

Default: `PBS_HOME/checkpoint`

`-c config_file`

MOM will read this alternate default configuration file upon starting. If this is a relative file name it will be relative to `PBS_HOME/mom_priv`. If the specified file cannot be opened, `pbs_mom` will abort. See the `-d` option.

MOM's normal operation, when the `-c` option is not given, is to attempt to open the default configuration file `PBS_HOME/mom_priv/config`. If this file is not present, `pbs_mom` will log the fact and continue.

-d home_directory

Specifies the path of the directory to be used in place of `PBS_HOME` by `pbs_mom`. The default directory is given by `$PBS_HOME`.

Format: String

-L logfile

Specifies an absolute path and filename for the log file. The default is a file named for the current date in `PBS_HOME/mom_logs/`. See the `-d` option.

Format: string.

-M TCP_port

Specifies the number of the TCP port on which MOM will listen for server requests and instructions.

Format: integer port number.

Default: *15002*.

-n nice_val

Specifies the priority for the `pbs_mom` daemon.

Format: integer.

-N

Specifies that when starting, MOM should not detach from the current session.

-p

Specifies that when starting, MOM should allow any running jobs to continue running, and not have them requeued. This option can be used for single-host jobs only; multi-host jobs cannot be preserved. Cannot be used with the `-r` option. MOM is not the parent of these jobs.

Altix running ProPack or Performance Suite

The `cpuset`-enabled `pbs_mom` will, if given the `-p` flag, use the existing CPU and memory allocations for the `/PBSPRO` cpusets. The default behavior is to remove these cpusets. Should this fail, MOM will exit, asking to be restarted with the `-p` flag.

-r

Specifies that when starting, MOM should requeue any rerunnable jobs and kill any non-rerunnable jobs that she was tracking, and mark the jobs as terminated. Cannot be used with the `-p` option. MOM is not the parent of these jobs.

It is not recommended to use the `-r` option after a reboot, because process IDs of new, legitimate tasks may match those MOM was previously tracking. If they match and MOM is started with the `-r` option, MOM will kill the new tasks.

-R UDP_port

Specifies the number of the UDP port on which MOM will listen for pings, resource information requests, communication from other MOMs, etc.

Format: integer port number

Default: *15003*

-S server_port

Specifies the number of the TCP port on which `pbs_mom` initially contact the server.

Format: integer port number

Default: *15001*

-s script_options

This option provides an interface that allows the administrator to add, delete, and display MOM's configuration files. See CONFIGURATION FILES. `script_options` are used this way:

-s insert <scriptname> <inputfile>

Reads `inputfile` and inserts its contents in a new site-defined `pbs_mom` configuration file with the filename `scriptname`. If a site-defined configuration file with the name `scriptname` already exists, the operation fails, a diagnostic is presented, and `pbs_mom` exits with a nonzero status. Scripts whose names begin with the prefix "*PBS*" are reserved. An attempt to add a script whose name begins with "*PBS*" will fail. `pbs_mom` will print a diagnostic message and exit with a nonzero status. Example:

```
pbs_mom -s insert <scriptname> <inputfile>
```

-s remove <scriptname>

The configuration file named `scriptname` is removed if it exists. If the given name does not exist or if an attempt is made to remove a script with the reserved "*PBS*" prefix, the operation fails, a diagnostic is presented, and `pbs_mom` exits with a nonzero status. Example:

```
pbs_mom -s remove <scriptname>
```

-s show <scriptname>

Causes the contents of the named script to be printed to standard output. If scriptname does not exist, the operation fails, a diagnostic is presented, and pbs_mom exits with a nonzero status. Example:

```
pbs_mom -s show <scriptname>
```

-s list

Causes pbs_mom to list the set of PBS-prefixed and site-defined configuration files in the order in which they will be executed. Example:

```
pbs_mom -s list
```

WINDOWS:

Under Windows, the -N option must be used, so that pbs_mom will start up as a standalone program. For example:

```
pbs_mom -N -s insert <scriptname> <inputfile>
```

or

```
pbs_mom -N -s list
```

--version

The pbs_mom command returns its PBS version information and exits. This option can only be used alone.

2.17.4 Files and Directories

`$PBS_HOME/mom_priv`

Default directory for default configuration files.

`$PBS_HOME/mom_priv/config`

MOM's default configuration file.

`$PBS_HOME/mom_logs`

Default directory for log files written by MOM.

`$PBS_HOME/mom_priv/prologue`

File containing administrative script to be run before job execution.

`$PBS_HOME/mom_priv/epilogue`

File containing administrative script to be run after job execution.

2.17.5 Signal Handling

pbs_mom handles the following signals:

SIGHUP

The `pbs_mom` daemon will reread its configuration files, close and reopen the log file, and reinitialize resource structures.

SIGALRM

MOM writes a log file entry. See the `-a alarm_timeout` option.

SIGINT

The `pbs_mom` daemon exits, leaving all running jobs still running. See the `-p` option.

SIGKILL

This signal is not caught. The `pbs_mom` daemon exits immediately.

SIGTERM, SIGXCPU, SIGXFSZ, SIGCPULIM, SIGSHUTDN

The `pbs_mom` daemon terminates all running children and exits.

SIGPIPE, SIGUSR1, SIGUSR2, SIGINFO

These are ignored.

All other signals have their default behavior installed.

2.17.6 Exit Status

- Greater than zero if the `pbs_mom` daemon fails to start, if the `-s insert` option is used with an existing scriptname, or if the administrator attempts to add a script whose name begins with “PBS”.
- Greater than zero if the administrator attempts to use the `-s remove` option on a nonexistent configuration file, or on a configuration file whose name begins with “PBS”.
- Greater than zero if the administrator attempts to use the `-s show` option on a nonexistent script.

2.17.7 See Also

The PBS Professional Administrator’s Guide, `pbs_server(8B)`, `pbs_sched(8B)`, `gstat(1B)`, SGI’s Altix documentation

2.18 pbs_mom_globus

PBS no longer supports Globus. The Globus functionality has been **removed** from PBS.

2.19 pbs_mpihp

Runs an MPI application in a PBS job with HP MPI

2.19.1 Synopsis

```
pbs_mpihp [-np #] [-h host] [other HP mpirun options] program [args]
```

```
pbs_mpihp [HP mpirun options] -f appfile [-- [<extra_args>]]
```

```
pbs_mpihp --version
```

2.19.2 Description

The PBS command `pbs_mpihp` replaces the standard `mpirun` command in a PBS HP MPI job, for executing programs.

`pbs_mpihp` is a front end to the HP MPI version of `mpirun`. It is for PBS jobs running under Linux 2.4 and higher. `pbs_mpihp` has the same usage as `mpirun`. When `pbs_mpihp` is invoked from a PBS job, it will process the command line arguments, then call standard HP `mpirun` to actually start the MPI ranks. The ranks created will be mapped onto cpus on the nodes allocated to the PBS job. The environment variable `MPI_REMSH` will be set to `$PBS_EXEC/bin/pbs_tmsh`. This will cause the processes that are created to become part of the PBS job.

The path to standard HP `mpirun` is found by checking to see if a link exists with the name `PBS_EXEC/etc/pbs_mpihp`. If this link exists, it will point to standard HP `mpirun`. If it does not exist, a call to `mpirun -version` will be made to determine if it is HP `mpirun`. If so, the call will be made to “`mpirun`” without an absolute path. If HP `mpirun` cannot be found, an error will be output, all temp files will be cleaned up and the script will exit with value 127.

If `pbs_mpihp` is invoked from outside a PBS job, it will pass all of its arguments directly to standard HP `mpirun` without further processing.

The first form above allows one executable to be specified. The second form above uses an appfile to list multiple executables. The format is described in the HP `mpirun` man page. If this form is used from inside a PBS job, the file will be read to determine what executables are to be run and how many processes will be started for each.

When HP MPI is wrapped with `pbs_mpihp`, “`rsh`” is the default used to start the mpids. If you wish to use “`ssh`” or something else, be sure to set the following in `$PBS_HOME/pbs_environment`:

```
PBS_RSHCOMMAND=ssh
```

or put the following in the job script:

```
export PBS_RSHCOMMAND=<rsh_cmd>
```

Executing `pbs_mpihp` with the `-client` option is not supported under PBS.

2.19.3 Usage

Usage is the same as for HP `mpirun`.

2.19.4 Options to `pbs_mpihp`

All options except the following are passed directly to HP `mpirun` with no modification.

- `-client`
Not supported.
- `-np number`
Specifies the number of processes to run on the PBS nodes.
- `-h host`
Ignored by `pbs_mpihp`.
- `-l user`
Ignored by `pbs_mpihp`.
- `-f appfile`
The specified appfile is read by `pbs_mpihp`.
- `--version`
The `pbs_mpihp` command returns its PBS version information and exits.
This option can only be used alone.

2.19.5 See Also

The PBS Professional Administrator's Guide

`mpirun(1)`

2.20 `pbs_mpilam`

Runs MPI programs under PBS with LAM MPI

2.20.1 Synopsis

pbs_mpilam [options]

pbs_mpilam --version

2.20.2 Description

The PBS command `pbs_mpilam` replaces the standard `mpirun` command in a PBS LAM MPI job, for executing programs under Linux 2.4 or higher.

Usage is the same as for LAM `mpirun`. All options are passed directly to `mpirun`. If used to run a single program, PBS tracks resource usage and controls all user processes spawned by the program. If used to run multiple programs as specified in an application file (no `<where>` argument and no `-np/-c` option), then PBS does not manage the spawned user processes of each program.

If the `where` argument is not specified, then `pbs_mpilam` will try to run the user's program on all available CPUs using the `C` keyword.

2.20.3 Options to `pbs_mpilam`

(options)

The `pbs_mpilam` command uses the same options as `mpirun`.

`--version`

The `pbs_mpilam` command returns its PBS version information and exits. This option can only be used alone.

2.20.4 Path

The `PATH` on remote machines must contain `PBS_EXEC/bin`.

2.20.5 See Also

The PBS Professional Administrator's Guide

`mpirun(1)`

2.21 `pbs_mpirun`

Runs MPI programs under PBS with MPICH

2.21.1 Synopsis

pbs_mpirun [options]

pbs_mpirun --version

2.21.2 Description

The PBS command `pbs_mpirun` replaces the standard `mpirun` command in a PBS MPICH job using P4 running under Linux 2.4 and higher. Usage is the same as for `mpirun`, except for the `-machinefile` option. All other options are passed directly to `mpirun`.

On Windows, this command cannot be used to start job processes or track a job's resource usage.

2.21.3 Options to `pbs_mpirun`

(options)

The options to `pbs_mpirun` are the same as for `mpirun`, except for the `-machinefile` option. This is generated by `pbs_mpirun`. The user should not attempt to specify `-machinefile`.

The value for `-machinefile` is a temporary file created from `PBS_NODEFILE` in the format:

`hostname-1[:number of processors]`

`hostname-2[:number of processors]`

`hostname-n[:number of processors]`

where if the number of processors is not specified, it is 1. An attempt by the user to specify the `-machinefile` option will result in a warning saying "Warning, `-machinefile` value replaced by PBS".

The default value for the `-np` option is the number of entries in `PBS_NODEFILE`.

`--version`

The `pbs_mpirun` command returns its PBS version information and exits. This option can only be used alone.

2.21.4 Environment Variables

`pbs_mpirun` modifies `P4_RSHCOMMAND` and `PBS_RSHCOMMAND`. Users should not edit these. `pbs_mpirun` copies the value of `P4_RSHCOMMAND` into `PBS_RSHCOMMAND`.

2.21.5 Path

The `PATH` on remote machines must contain `PBS_EXEC/bin`.

2.21.6 See Also

The PBS Professional Administrator's Guide

`mpirun(1)`

2.22 `pbs_password`

Sets or updates password of a PBS user

2.22.1 Synopsis

`pbs_password [-r] [-s server] [-d] [user]`

`pbs_password --version`

2.22.2 Description

The `pbs_password` command is used to set or update the password of a PBS user. The user does not have to have any jobs on the system.

When no options are given to `pbs_password`, the password credential on the default PBS server for the current user, i.e. the user who executes the command, is updated to the prompted password. Any user jobs previously held due to an invalid password are not released.

Available on Windows and supported Linux x86 and x86_64 platforms only.

The `pbs_password` command has no effect on running jobs. Queued jobs use the new password.

The `pbs_password` command does not change the user's password on the current host, only the password that is cached in PBS.

Note that `pbs_password` encrypts the password obtained from the user before sending it to the PBS Server.

2.22.3 Options to `pbs_password`

(no options)

The user is prompted for a new password. The password credential on the default PBS server for the current user, i.e. the user who executes the command, is updated to the prompted password. Any user jobs previously held due to an invalid password are not released.

`-r`

Any user jobs previously held due to an invalid password are released.

`-s server`

Allows user to specify server where password will be changed.

`-d`

Deletes the password.

`user`

The password credential of user `user` is updated to the prompted password. If user is not the current user, this action is only allowed if one of the following is true:

- The current user is root or admin.
- User `user` has given the current user explicit access via the `ruserok()` mechanism, i.e. the hostname of the machine from which the current user is logged in appears in the server's `hosts.equiv` file, or the current user has an entry in user's `HOMEDIR/.rhosts` file.

`--version`

The `pbs_password` command returns its PBS version information and exits. This option can only be used alone.

2.22.4 Exit Status

Table 2-2: Exit Status

Exit Status	Meaning
0	Success
-1	single_signon_password_enable not set
-2	Password of user on server failed to be created/updated
-3	Failed to release jobs held due to bad password owned by user on server
-4	Failed to delete password of user on server
-5	Current user not authorized to change password of user

2.22.5 See Also

`qhold(1B)`, `qrls(1B)`, `qselect(1B)`, `ruserok()`

2.23 pbs_probe

Reports PBS diagnostic information

2.23.1 Synopsis

pbs_probe [-f | -v]

pbs_probe --version

2.23.2 Description

The `pbs_probe` command reports post-installation information that is useful for PBS diagnostics. Aside from the direct information that is supplied on the command line, `pbs_probe` uses as the source for basic information the file `/etc/pbs.conf` and the values of any of the following environment variable that may be set in the environment in which `pbs_probe` is run: `PBS_CONF_FILE`, `PBS_HOME`, `PBS_EXEC`, `PBS_START_SERVER`, `PBS_START_MOM`, and `PBS_START_SCHED`.

In order to execute `pbs_probe`, the user must have PBS Operator or Manager privilege.

Used without options, the `pbs_probe` runs in “report” mode. In this mode `pbs_probe` reports on any errors in the PBS infrastructure files that it detects. The problems are categorized, and a list of the problem messages placed in each category are output. Those categories which are empty do not show in the output.

2.23.3 Options to `pbs_probe`

-f

Run in “fix” mode. In this mode `pbs_probe` will examine each of the relevant infrastructure files and, where possible, fix any errors that it detects, and print a message of what got changed. If it is unable to fix a problem, it will simply print a message regarding what was detected.

-v

Run in “verbose” mode. If the verbose option is turned on, `pbs_probe` will also output a complete list of the infrastructure files that it checked.

--version

The `pbs_probe` command returns its PBS version information and exits. This option can only be used alone.

2.23.4 Standard Error

The `pbs_probe` command will write a diagnostic message to standard error for each error occurrence.

2.23.5 Files

`/etc/pbs.conf` `/etc/init.d/pbs`

2.23.6 See Also

The PBS Professional Administrator’s Guide and the following manual pages:
`pbs_server(8B)`, `pbs_sched(8B)`, `pbs_mom(8B)`.

2.24 pbs_python

Python interpreter for debugging a hook script from the command line

2.24.1 Synopsis

```
pbs_python --hook [-e <log_event_mask>] [-i <input_file>] [-L <log_dir>] [-l <log_file>]
[-o <output_file>] [-r <resourcedef_file>] [<python_script>]
```

```
pbs_python --version
```

2.24.2 Description

The PBS Python interpreter is a wrapper for Python. You can use the PBS Python interpreter for debugging a hook script at the command line by specifying the `--hook` option. You can use the `--hook` option to get access to the PBS Python module and perform interactive or non-interactive debugging.

Without the `--hook` option, you can use `pbs_python` to run the Python interpreter that is distributed with PBS Professional.

2.24.3 Options to pbs_python

`--hook`

This option is a switch. When you use this option, you can use the PBS Python module (via `"import pbs"`), and the other options described here are available. When you use this option, you cannot use the standard Python options. This option is useful for debugging.

When you do not use this option, you cannot use the other options listed here, but you can use the standard Python options.

`-i <input_file>`

Text file containing data to populate `pbs.event()` objects. Each line specifies an attribute value or a resource value. Syntax of each input line is one of the following:

```
<object_name>.<attribute_name>=<attribute_value>
```

```
<object_name>.<resource_list>[<resource_name>]=<resource_value>
```

Where

`<object_name>` is a PBS object name which can refer to its sub-objects.

Examples: `"pbs.event()", "pbs.event().job", "pbs.event().vnode_list["<vnode name>"]"`.

Example input file:

```
pbs.event().hook_name=proto
pbs.event().hook_type=site
pbs.event().type=queuejob
pbs.event().requestor=user1
pbs.event().requestor_host=host1
pbs.event().hook_alarm=40
pbs.event().job.id=72
pbs.event().job.Job_Name=job1
pbs.event().job.Resource_List[ncpus]=5
pbs.event().job.Resource_List[mem]=6mb
pbs.event().vnode_list["host1"].resources_available["ncpus"] = 5
pbs.event().vnode_list["host1"].resources_available["mem"] =
    300gb
```

Available only when `--hook` option is used.

`-e <log_event_mask>`

Sets the mask that determines which event types are logged by `pbs_python`. To see only debug messages, set the value to `0xd80`. To see all messages, set the value to `0xffff`. The `pbs_python` interpreter uses the same set of mask values that are used for the `$logevent <mask>` entry in the `pbs_mom` configuration file. See [section 2.17, “pbs mom”, on page 58](#). Available only when `--hook` option is used.

`-L <log_dir>`

Directory holding the log file where `pbs.logmsg()` and `pbs.logjobmsg()` write their output. Default is current working directory where `pbs_python` is executed. Available only when `--hook` option is used.

`-l <log_file>`

Log file where `pbs.logmsg()` and `pbs.logjobmsg()` write their output. Default file name is current date in `yyyymmdd` format. Available only when `--hook` option is used.

-o <output_file>

The output file contains the changes made after executing the hook script, such as the attributes and resources set in any `pbs.event()` jobs and reservations, whether to accept or reject an action, and any `pbs.reject()` messages.

Example output file:

```
pbs.event().job.Job_Name=job2
pbs.event().job.Resource_List[file]=60gb
pbs.event().job.Resource_List[ncpus]=5
pbs.event().job.Resource_List[mem]=20gb
pbs.event().job.Account_Name=account2
pbs.event().reject=True
pbs.event().reject_msg=No way!
```

Without this option, output goes to `stdout`. Available only when `--hook` option is used.

-r <resourcedef>

File/path name containing a resource definition specifying a custom resource whose Python type is `pbs.resource`. Format:

```
<resource_name>type=<typename> [flag=<value>]
```

This file has the same format as the `PBS_HOME/server_priv/resourcedef` file. Available only when `--hook` option is used.

--version

The `pbs_python` command prints its version information and exits. This option can only be used alone.

2.24.4 Arguments

<python_script>

The hook script to execute. We recommend importing the PBS Python module at the start of the script:

```
import pbs
```

If you do not specify *<python_script>*, you can perform interactive debugging. If you type the following:

```
% pbs_python --hook -i hook.input
```

The interpreter displays a prompt:

```
>>
```


You can type your Python lines at the prompt:

```
>>import pbs
>> e=pbs.event().job
>> print e.id
<job-id>
...
```

2.25 pbs_rdel

Deletes a PBS advance or standing reservation

2.25.1 Synopsis

pbs_rdel reservation_identifier[,reservation_identifier...]

pbs_rdel --version

2.25.2 Description

The `pbs_rdel` command deletes reservations in the order in which their reservation identifiers are presented to the command.

A reservation may be deleted by its owner, the PBS Operator, or the PBS Manager.

2.25.3 Options

`--version`

The `pbs_rdel` command returns its PBS version information and exits.
This option can only be used alone.

2.25.4 Operands

The `pbs_rdel` command accepts one or more `reservation_identifier` operands.

For an advance reservation this has the form:

[R]sequence_number[.server_name][@remote_server]

For a standing reservation this has the form:

[S]sequence_number[.server_name][@remote_server]

@remote_server is used to specify a reservation at a server other than the default server.

2.25.5 Exit Status

Zero upon success.

Greater than zero upon failure to process any operand.

2.25.6 See Also

The PBS Professional User's Guide, the PBS Professional Administrator's Guide, `pbs_rsub(1B)`, `pbs_rstat(1B)`, `pbs_resv_attributes(7B)`

2.26 pbs_renew

Renews Kerberos credential

2.26.1 Synopsis

pbs_renew [-d] program [arg(s)]

pbs_renew --version

2.26.2 Description

The `pbs_renew` command is used internally by PBS when a job has a Kerberos credential. The program is run as a child process with any arguments passed to the command line of program. The `pbs_renew` process runs periodically to renew any Kerberos credential. It will wait for the child process to return, clean up any Kerberos credential and exit when the child process is done.

2.26.3 Options

-d

Debug messages are printed to stderr.

--version

The `pbs_renew` command returns its PBS version information and exits.
This option can only be used alone.

2.26.4 See Also

The PBS Professional Administrator's Guide, `qsub` (1B)

2.27 pbs_rstat

Shows status of PBS advance or standing reservations

2.27.1 Synopsis

pbs_rstat [-F][-B][-S] [reservation_id...]

pbs_rstat --version

2.27.2 Description

The `pbs_rstat` command is used to show the status of all reservations on the PBS Server. Denied reservations are not displayed.

This command has three different output formats: brief (B), short (S), and full (F). This command can be used with any level of PBS privilege.

See the `pbs_resv_attributes`(7B) man page for information about reservation attributes.

2.27.3 Options to pbs_rstat**-B Brief**

Displays each reservation identifier only.

-S Short

Displays a table showing the name, queue, owner, state, start time, duration, and end time of each reservation.

-F Full

Displays all reservation attributes that are not set to the default value. Users without manager or operator privilege cannot print custom resources which were created to be invisible to users.

--version

The `pbs_rstat` command returns its PBS version information and exits. This option can only be used alone.

2.27.4 Output

See [section 8.6, “Reservation States”, on page 419](#).

2.27.5 Operands

The `pbs_rstat` command accepts one or more `reservation_identifier` operands.

2.27.5.1 Reservations at the default server

For an advance reservation, the `reservation_identifier` has the form:

[R]sequence_number[.server_name]

For a standing reservation, the `reservation_identifier` has the form:

[S]sequence_number[.server_name]

2.27.5.2 Reservations at a server other than the default server:

Specify the remote server’s name using `@remote_server`.

For an advance reservation:

[R]sequence_number[.server_name][@remote_server]

For a standing reservation:

[S]sequence_number[.server_name][@remote_server]

2.27.6 See Also

The PBS Professional User's Guide, the PBS Professional Administrator's Guide, `pbs_rsub(1B)`, `pbs_rdel(1B)`, `pbs_resv_attributes(7B)`

2.28 pbs_rsub

Creates a PBS advance or standing reservation

2.28.1 Synopsis

```
pbs_rsub [-D duration] [-E end_time] [-g group_list] [-G auth_group_list] [-H
auth_host_list] [-I seconds] [-m mail_points] [-M mail_list] [-N reservation_name] [-q
destination] [-r recurrence_rule] [-R start_time] [-u user_list] [-U auth_user_list] [-W
attribute_value_list] -l resource_request [-l placement]
```

```
pbs_rsub --version
```

2.28.2 Description

The `pbs_rsub` command is used to create an advance or standing reservation. An advance reservation reserves specific resources for the requested time period, and a standing reservation reserves specific resources for recurring time periods. When a reservation is created, it has an associated queue.

After the reservation is requested, it is either confirmed or denied. Once the reservation has been confirmed, authorized users submit jobs to the reservation's queue via `qsub` and `qmove`.

A confirmed reservation will accept jobs at any time. The jobs in its queue can run only during the reservation period, whether during a single advance reservation or during the occurrences of a standing reservation.

When an advance reservation ends, all of its jobs are deleted, whether running or queued. When an occurrence of a standing reservation ends, only its running jobs are deleted; those jobs still in the queue are not deleted.

To get information about a reservation, use the `pbs_rstat` command.

To delete a reservation, use the `pbs_rdel` command. Do not use the `qdel` command.

The behavior of the `pbs_rsub` command may be affected by any site hooks. Site hooks can modify the reservation's attributes.

2.28.3 Requirements

When using `pbs_rsub` to request a reservation, the user must specify two of the following options: `-R`, `-E`, and `-D`. The resource request `-l walltime` can be used instead of the `-D` option.

2.28.4 Options to **pbs_rsub**

-D duration

Specifies reservation duration. If the start time and end time are the only times specified, this duration time is calculated.

Format: *Duration*

Default: none

-E end_time

Specifies the reservation end time. If start time and duration are the only times specified, the end time value is calculated.

Format: *Datetime*.

Default: none

-g group_list

The **group_list** is a comma-separated list of group names. The server uses entries on this list, along with an ordered set of rules, to associate a group name with the reservation.

Format: *group@hostname[,group@hostname ...]*

-G auth_group_list

Comma-separated list of names of groups who can or cannot submit jobs to this reservation. Group names are interpreted in the context of the server's host, not the context of the host from which the job is submitted.

This list becomes the **acl_groups** list for the reservation's queue. More specific entries should be listed before more general, because the list is read left-to-right, and the first match determines access.

If both **Authorized_Users** and **Authorized_Groups** are set, a user must belong to both in order to be able to submit jobs to this reservation. If the reservation creator specifies this list, the creator's group is not automatically added to the list.

Refer to the **Authorized_Groups** reservation attribute on the **pbs_resv_attributes(7B)** man page.

Format: *[+|-]group_name[, [+|-]group_name ...]*

Default: All groups are authorized to submit jobs.

-H auth_host_list

Comma-separated list of hosts from which jobs can and cannot be submitted to this reservation. This list becomes the **acl_hosts** list for the reservation's queue. More specific entries should be listed before more general, because the list is read left-to-right, and the first match determines access.

If the reservation creator specifies this list, the creator's host is not automatically added to the list.

See the `Authorized_Hosts` reservation attribute on the `pbs_resv_attributes(7B)` man page.

Format: `[+|-]hostname[+|-]hostname ...]`

Default: All hosts are authorized to submit jobs.

-l block_time

Specifies interactive mode. The `pbs_rsub` command will block, up to `block_time` seconds, while waiting for the reservation request to be confirmed or denied.

If `block_time` is positive, and the reservation isn't confirmed or denied in the specified time, the ID string for the reservation is returned with the status "UNCONFIRMED".

If `block_time` is negative, and the scheduler doesn't confirm or deny the reservation in the specified time, the reservation is deleted.

Format: *Integer*.

Default: *Not interactive*.

-l placement

The placement specifies how vnodes are reserved. The place statement has this form:

`-l place=[arrangement][: sharing][: grouping]`

where

arrangement is one of *free* | *pack* | *scatter* | *vscatter*

sharing is one of *excl* | *share* | *exclhost*

grouping can have only one instance of *group=resource*

and where

free:

Use any vnode(s) for reservation.

pack:

All chunks are taken from one host.

scatter:

Only one chunk with any MPI processes will be taken from a host. A chunk with no MPI processes may be taken from the same node as another chunk.

vscatter:

Only one chunk is used in any vnode.

excl:

Only this reservation uses the vnodes chosen.

exclhost:

The entire host is allocated to the reservation.

share:

This reservation can share the chosen vnodes.

group=resource:

Chunks will be grouped according to a resource. All nodes in the group must have a common value for the resource, which can be either the built-in resource host or a site-defined node-level resource.

Note that nodes can have sharing attributes that override reservation placement requests.

See the `pbs_node_attributes(7B)` man page.

-l resource_request

The `resource_request` specifies the resources required for the reservation. These resources will be used for the limits on the queue that is dynamically created for the reservation. The aggregate amount of resources for currently running jobs from this queue will not exceed these resource limits. Jobs in the queue that request more of a resource than the queue limit for that resource are not allowed to run. Also, the queue inherits the value of any resource limit set on the server, and these are used for the job if the reservation request itself is silent about that resource. A non-privileged user cannot submit a reservation requesting a custom resource which has been created to be invisible or read-only for users.

Resources are requested by using the `-l` option, either in chunks inside of selection statements, or in job-wide requests using `resource_name=value` pairs. The selection statement is of the form:

`-l select=[N:]chunk[+[N:]chunk ...]`

where *N* specifies how many of that chunk, and a chunk is of the form:

`resource_name=value[:resource_name=value ...]`

Job-wide `resource_name=value` requests are of the form:

`-l resource_name=value[,resource_name=value ...]`

-m mail_points

Specifies the set of events that cause mail to be sent to the list of users specified in the `-M mail_list` option.

Format: string consisting of 1) any combination of “a”, “b”, “c” or “e”, or 2) the single character “n”.

Table 2-3: Suboptions to -m Option

Character	Meaning
a	Notify if the reservation is terminated for whatever reason
b	Notify when the reservation period begins
c	Notify when the reservation is confirmed
e	Notify when the reservation period ends
n	Send no mail. Cannot be used with any of a, b, c or e.

Default: “ac”.

-M mail_list

The list of users to whom mail is sent whenever the reservation transitions to one of the states specified in the -m mail_points option.

Format: *user[@hostname][,user[@hostname]...]*

Default: Reservation owner.

-N reservation_name

This specifies a name for the reservation.

Format: String up to 15 characters in length. It must consist of printable, non-white space characters with the first character alphabetic.

Default: None.

-q destination

Specifies the destination server at which to create the reservation.

Default: The default server is used if this option is not selected.

-r recurrence_rule

Specifies rule for recurrence of standing reservations. Rule must conform to iCalendar syntax, and is specified using a subset of parameters from RFC 2445.

Valid syntax for the recurrence_rule takes one of two forms:

FREQ= freq_spec; COUNT= count_spec; interval_spec

or

FREQ=*freq_spec*; *UNTIL*=*until_spec*; *interval_spec*

where

freq_spec

Frequency with which the standing reservation repeats. Valid values are:

WEEKLY|DAILY|HOURLY

count_spec

The exact number of occurrences. Number up to 4 digits in length.

Integer.

interval_spec

Specifies interval.

Format is one or both of:

BYDAY = *MO|TU|WE|TH|FR|SA|SU*

or

BYHOUR = *0|1|2|...|23*

When using both, separate them with a semicolon.

Elements specified in the recurrence rule override those specified in the arguments to the -R and -E options. For example, the BYHOUR specification overrides the hourly part of the -R option. For example, -R 0730 -E 0830 . . . BYHOUR=9 results in a reservation that starts at 9:30 and runs for 1 hour.

until_spec

Occurrences will start up to but not after date and time specified. Format:

YYYYMMDD[THHMMSS]

Note that the year-month-day section is separated from the hour-minute-second section by a capital T.

Requirements:

- The recurrence rule must be on one unbroken line and must be enclosed in double quotes.
- A start and end date must be used when specifying a recurrence rule. See the R and E options.
- The PBS_TZID environment variable must be set at the submission host. The format for PBS_TZID is a timezone location. Examples: America/Los_Angeles, America/Detroit, Europe/

Berlin, Asia/Calcutta. See the PBS Professional User's Guide.

Examples of Standing Reservations

For a reservation that runs every day from 8am to 10am, for a total of 10 occurrences:

```
pbs_rsub -R 0800 -E 1000 -r "FREQ=DAILY;COUNT=10"
```

Every weekday from 6am to 6pm until December 10 2008

```
pbs_rsub -R 0600 -E 1800 -r "FREQ=WEEKLY; BYDAY=MO,TU,WE,TH,FR;  
UNTIL=20081210"
```

Every week from 3pm to 5pm on Monday, Wednesday, and Friday, for 9 occurrences, i.e., for three weeks:

```
pbs_rsub -R 1500 -E 1700 -r "FREQ=WEEKLY;BYDAY=MO,WE,FR;  
COUNT=3"
```

-R start_time

Specifies reservation starting time. If the reservation's end time and duration are the only times specified, this start time is calculated.

If the day, DD, is not specified, it defaults to today if the time hhmm is in the future. Otherwise, the day is set to tomorrow. For example, if you submit a reservation with the specification -R 1110 at 11:15 a.m., it is interpreted as being for 11:10am tomorrow. If the month portion, MM, is not specified, it defaults to the current month, provided that the specified day DD, is in the future. Otherwise, the month is set to next month. Similar rules apply to the two other optional, left-side components.

Format: *Datetime*

-u user_list

Comma-separated list of user names. Not used. Refer to the User_List reservation attribute on the pbs_resv_attributes(7B) man page.

Format: *user[@host]/,user[@host] ...*

Default: None.

-U auth_user_list

Comma-separated list of users who are and are not allowed to submit jobs to this reservation. This list becomes the acl_users attribute for the reservation's queue. More specific entries should be listed before more general, because the list is read left-to-right, and the first match determines access.

If both Authorized_Users and Authorized_Groups are set, a user must belong to both in order to be able to submit jobs to this reservation. The

reservation creator's username is automatically added to this list, whether or not the reservation creator specifies this list.

Refer to the `Authorized_Users` reservation attribute on the `pbs_resv_attributes(7B)` man page.

Format: `[+|-]user@host[, [+|-]user@host...]`

Default: Job owner only.

-W attribute_value_list

This allows you to define other attributes for the reservation.

Supported attributes:

qmove=jobid

Converts a normal job designated by *jobid* into a reservation job that will run as soon as possible. Creates the reservation with its queue and moves the job into the reservation's queue. Uses the resources requested by the job to create the reservation.

When the reservation is created, it inherits its resources from the job, not from the resources requested through the `pbs_rsub` command.

If the `qmove` option is used and the reservation is not confirmed within the timeout period, the reservation is deleted. The default timeout period is 10 seconds. There is no option for this kind of reservation to be unconfirmed.

To specify the timeout, give a negative value for the `-I` option. For example, to specify a timeout of 300 seconds:

```
pbs_rsub -Wqmove=<job ID> -I -300
```

The `-R` and `-E` options to `pbs_rsub` are disabled when using the `qmove=jobid` attribute.

Some shells require that you enclose a job array ID in double quotes.

Timeout must be specified with a negative number.

--version

The `pbs_rsub` command returns its PBS version information and exits. This option can only be used alone.

2.28.5 Output

The `pbs_rsub` command returns the reservation name.

For an advance reservation, this has the form

RNNNN.server

where *NNNN* is a unique integer. The associated queue's name is the prefix, *RNNNN*.

For a standing reservation, this has the form

SNNNN.server

where *NNNN* is a unique integer. The associated queue's name is the prefix, *SNNNN*.

2.28.6 See Also

The PBS Professional User's Guide, the PBS Professional Administrator's Guide, `pbs_resv_attributes(7B)`, `pbs_rdel(1B)`, `pbs_rstat(1B)`, `qmove(1B)`, `qsub(1B)`

2.29 pbs_sched

Runs the PBS scheduler

2.29.1 Synopsis

```
pbs_sched [-a alarm] [-c clientsfile] [-d home] [-L logfile] [-n] [-N] [-p file] [-R port] [-S
port]
```

```
pbs_sched --version
```

2.29.2 Description

`pbs_sched` is the PBS scheduling daemon. It schedules PBS jobs.

`pbs_sched` must be executed with root permission.

2.29.3 Options to `pbs_sched`

-a alarm

Deprecated. Will overwrite value of `sched_cycle_length` scheduler attribute. Time in seconds to wait for a scheduling cycle to finish.

Format: Time, in seconds.

-c clientsfile

Add clients to the scheduler's list of known clients. The `clientsfile` contains single-line entries of the form

```
$clienthost <hostname>
```

Each hostname is added to the list of hosts allowed to connect to the scheduler. If `clientsfile` cannot be opened, the scheduler aborts. Path can be absolute or relative. If relative, it is relative to `PBS_HOME/sched_priv/`.

-d home

The directory in which the scheduler will run.

Default: `PBS_HOME/sched_priv`.

-L logfile

The absolute path and filename of the log file. The scheduler writes its PBS version and build information to the logfile whenever it starts up or the logfile is rolled to a new file.

See the `-d` option.

Default: The scheduler will open a file named for the current date in the `PBS_HOME/sched_logs` directory.

-n

This will tell the scheduler to not restart itself if it receives a `sigsegv` or a `sigbus`. The scheduler will by default restart itself if it receives either of these two signals. The scheduler will not restart itself if it receives either one within five minutes of starting.

-N

Instructs the scheduler not to detach itself from the current session.

-p file

Any output which is written to standard out or standard error will be written to this file. The pathname can be absolute or relative, in which case it will be relative to `PBS_HOME/sched_priv`.

See the `-d` option.

Default: `PBS_HOME/sched_priv/sched_out`.

-R port

The port for MOM to use. If this option is not given, the port number is taken from `PBS_MANAGER_SERVICE_PORT`, in `pbs.conf`.

Default: `15003`.

-S port

The port for the scheduler to use. If this option is not given, the default port for the PBS scheduler is taken from

`PBS_SCHEDULER_SERVICE_PORT`, in `pbs.conf`.

Default: `15004`.

--version

The `pbs_sched` command returns its PBS version information and exits. This option can only be used alone.

2.29.4 Signal Handling**SIGHUP**

The scheduler will close and reopen its log file and reread the config file if one exists.

SIGALRM

If the scheduler exceeds the time limit, the alarm will cause the scheduler to attempt to core dump and restart itself.

SIGINT and SIGTERM

Will result in an orderly shutdown of the scheduler.

All other signals have the default action installed.

2.29.5 Exit Status

Zero upon normal termination.

2.29.6 See Also

The PBS Professional Administrator's Guide, `pbs_server(8B)`, `pbs_mom(8B)`

2.30 pbs_server

Starts a PBS batch server

2.30.1 Synopsis

```
pbs_server [-a active] [-A acctfile] [-C] [-d config_path] [-e mask] [-F seconds] [-L logfile]
           [-M mom_port] [-N] [-p port] [-R momRPP_port] [-S scheduler_port] [-t type]
```

```
pbs_server --version
```

2.30.2 Description

The `pbs_server` command starts the operation of a batch server on the local host. Typically, this command will be in a local boot file such as `/etc/rc.local`. If the batch server is already in execution, `pbs_server` will exit with an error. To insure that the `pbs_server` command is not runnable by the general user community, the server will only execute if its real and effective UID is zero.

The server will record a diagnostic message in a log file for any error occurrence. The log files are maintained in the `server_logs` directory below the home directory of the server. If the log file cannot be opened, the diagnostic message is written to the system console. The server writes its PBS version and build information to the logfile whenever it starts up or the logfile is rolled to a new file.

To kill the server:

UNIX/Linux:

qterm (see qterm(8B))

or

"kill <server_pid>", which sends a SIGTERM.

Windows:

if you're running "`pbs_server -N`" for a standalone mode server, use

<cntrl>-<break>.

2.30.3 Options to pbs_server

-a <value>

When *True*, the server is in state "*active*" and the scheduler is called to schedule jobs. When *False*, the server is in state "*idle*" and the scheduler is not called to schedule jobs. Sets the server's `scheduling` attribute. If the `-a T|F` option is not specified, the server uses the prior value for the `scheduling` attribute.

Format: Boolean

-A acctfile

Specifies an absolute path name for the file to use as the accounting file. If not specified, the file is named for the current date in the `PBS_HOME/server_priv/accounting` directory.

-C

The server starts up, creates the database, and exits. Windows only.

-d config_path

Specifies the path of the directory which is home to the servers configuration files, `PBS_HOME`. A host may have multiple servers. Each server must have a different configuration directory. The default configuration directory is given by the symbol `$PBS_HOME` which is typically `/usr/spool/PBS`.

-e mask

Specifies a log event mask to be used when logging. See “log_events” in the `pbs_server_attributes(7B)` man page.

-F seconds

Specifies the number of seconds that the secondary server should wait before taking over when it believes the primary server is down. If the number of seconds is specified as `-1`, the secondary will make one attempt to contact the primary and then become active.

Default: *30 seconds*

-L logfile

Specifies an absolute path name of the file to use as the log file. If not specified, the file is one named for the current date in the `PBS_HOME/server_logs` directory

See the `-d` option.

-M mom_port

Specifies the host name and/or port number on which the server should connect the job executor, MOM. The option argument, `mom_conn`, is one of the forms:

host_name, [:]port_number

or

host_name:port_number

If `host_name` not specified, the local host is assumed. If `port_number` is not specified, the default port is assumed.

See the `-M` option for `pbs_mom(8B)`.

Default: *15002*

-N

The server runs in standalone mode. In Windows, it does not register as a Windows service. On other platforms, MOM will not detach from the current session.

-p port

Specifies the port number on which the server will listen for batch requests. If multiple servers are running on a single host, each must have its own unique port number. This option is for use in testing with multiple batch systems on a single host.

Default: *15001*

-R mom_RPPport

Specifies the port number on which the server should query the up/down status of Mom. See the -R option for `pbs_mom(8B)`.

Default: *15003*

-s replacement_string

Specifies the string to use when replacing spaces in accounting entity names. Only available under Windows.

See the PBS Professional Administrator's Guide.

-S scheduler_port

Specifies the port number to which the server should connect when contacting the Scheduler. The option argument, `scheduler_conn`, is of the same syntax as under the -M option.

Default: *15004*

-t type

Specifies behavior when the server restarts. The `type` argument is one of the following:

cold

All jobs are purged. Positive confirmation is required before this direction is accepted.

create

The server will discard any existing configuration files: server, nodes, queues and jobs, and initialize configuration files to the default values. The server is idled (scheduling set *False*).

hot

All jobs in the *Running* state are retained in that state. Any job that was queued into the *Queued* state from the *Running* state when the server last shut down will be run immediately, assuming the required resources are available. This returns the server to the same state as when it went down. After those jobs are restarted, then normal scheduling takes place for all remaining queued jobs. All other jobs are retained in their current state.

If a job cannot be restarted immediately because of a missing resource, such as a node being down, the server will attempt to restart it periodically for up to 5 minutes. After that period, the server will revert to a normal state, as if `warm` started, and will no longer attempt to restart any remaining jobs which were running prior to the shutdown.

updatedb

Updates format of PBS data from the previous format to the data service format.

warm

All jobs in the *Running* state are retained in that state. All other jobs are maintained in their current state. The job scheduler will typically make new selections for which jobs are placed into execution. `warm` is the default if `-t` is not specified.

--version

The `pbs_server` command returns its PBS version information and exits. This option can only be used alone.

2.30.4 Files

`$PBS_HOME/server_priv`

default directory for configuration files.

`$PBS_HOME/server_logs`

directory for log files recorded by the server.

2.30.5 Signal Handling

On receipt of the following signals, the server performs the defined action:

SIGHUP

The current server log and accounting log are closed and reopened. This allows for the prior log to be renamed and a new log started from the time of the signal.

SIGTERM

Causes a rapid orderly shutdown of `pbs_server`, identical to “`qterm -t quick`”.

SIGSHUTDN

On systems (Unicos) where `SIGSHUTDN` is defined, it also causes an orderly “quick” shutdown of the server.

SIGPIPE, SIGUSR1, SIGUSR2

These signals are ignored.

All other signals have their default behavior installed.

2.30.6 Exit Status

If the server command fails to begin batch operation, the server exits with a value greater than zero.

2.30.7 See Also

The PBS Professional Administrator's Guide and the following manual pages: `qsub` (1B), `pbs_connect`(3B), `pbs_mom`(8B), `pbs_sched`(8B), `pbsnodes`(8B), `qdisable`(8B), `qenable`(8B), `qmgr`(8B), `qrun`(8B), `qstart`(8B), `qstop`(8B), and `qterm`(8B)

2.31 pbs_tclsh

TCL shell with TCL-wrapped PBS API

2.31.1 Synopsis

pbs_tclsh

pbs_tclsh -version

2.31.2 Description

The `pbs_tclsh` is a version of the TCL shell which includes wrapped versions of the PBS external API. The PBS TCL API is documented in the `pbs_tclapi` (3B) manual page.

Root privilege is required in order to query MOM for dynamic resources. Root privilege is not required in order to query MOM for built-in resources and site-defined static resources.

The `pbs_tclsh` command is used to query MOM. For example:

```
> pbs_tclsh
tclsh> openrm <hostname>
<file descriptor>
tclsh> addreq <file descriptor> "loadave"
tclsh> getreq <file descriptor>
<load average>
tclsh> closereq <file descriptor>
```

2.31.3 Options

`--version`

The `pbs_tclsh` command returns its PBS version information and exits. This option can only be used alone.

2.31.4 Standard Error

The `pbs_tclsh` command will write a diagnostic message to standard error for each error occurrence.

2.31.5 See Also

The PBS Professional Administrator's Guide, the PBS Programmer's Guide, and the following manual pages: `pbs_wish(8B)`, `pbs_server(8B)`, `pbs_mom(8B)`, `pbs_sched(8B)`

2.32 pbs_tmrsh

TM-enabled replacement for `rsh/ssh` for use by MPI implementations

2.32.1 Synopsis

```
pbs_tmrsh host [-l username] [-n] command [args ...]
pbs_tmrsh --version
```

2.32.2 Description

The `pbs_tmsh` command attempts to emulate an “`rsh`” connection to the specified host, via underlying calls to the Task Management (TM) API. The program is intended to be used during MPI integration activities, and not by end-users. The initial version of this program is targeted for use with MPICH and HP-MPI.

Running “`pbs_tmsh host command`” will cause a PBS task to be started on “`host`” running “`command`”. The “`host`” may be in IP dot address form.

The environment variables used by the two MPI implementations to point to the `rsh` work-alike (`MPI_REMSH` in the case of HP and `P4_RSHCOMMAND` for MPICH) must be set in the job environment and point to the full path for `pbs_tmsh`.

The file `$PBS_HOME/pbs_environment` will be used to set an environment variable `PATH` to be used to search for the program executable. This applies to both Windows and UNIX. It is expected that a full path will be specified for the command and the `PATH` variable will not be needed.

Output and errors are written to the PBS job’s output and error files, not to standard output/error.

2.32.3 Options

`-l username`

Specifies the username under which to execute the task. If used, username must match the username running the `pbs_tmsh` command.

`-n`

Currently a no-op; provided for MPI implementations that expect to call `rsh` with the “`-n`” option.

`--version`

The `pbs_tmsh` command returns its PBS version information and exits. This option can only be used alone.

2.32.4 Standard Error

The `pbs_tmsh` command will write a diagnostic message to the PBS job’s error file for each error occurrence.

2.32.5 Exit Status

The `pbs_tmrsh` program will exit with the exit status of the remote command or with 255 if an error occurred. This is because `ssh` works this way.

2.32.6 See Also

The PBS Professional Administrator's Guide and the following manual pages:
`pbs_attach(8B)`, `tm(3)`

2.33 `pbs_topologyinfo`

Reports topological information about vnodes

2.33.1 Synopsis

```
pbs_topologyinfo [ -a ] [ -s ]
```

```
pbs_topologyinfo [ -s <vnode name> [<vnode name> ...]]
```

```
pbs_topologyinfo [ -h ]
```

2.33.2 Description

The `pbs_topologyinfo` command reports topological information for one or more vnodes. To use the command, you must specify what kind of topological information you want. The command reports only the requested information.

`pbs_topologyinfo -a -s` reports socket counts for all vnodes that have reported sockets.

`pbs_topologyinfo -s <vnode name>` reports socket count for vnode `<vnode name>`.

This command is not supported on Windows.

This command must be run on the server host.

2.33.3 Permissions for `pbs_topologyinfo`

This command can be run only by root.

2.33.4 Options for **pbs_topologyinfo**

- a, --all
Reports requested topological information for all vnodes. When this option is used alone, the command does not report any information.
- s, --sockets
This option specifies socket count information. The command reports derived socket counts.
- h, --help
Prints usage and exits.

2.33.5 Operands

- <vnode name>
Name of vnode about which to report.

2.33.6 Standard Error

- 0
Success
- 1
Any error following successful command line processing.
If an invalid vnode name is specified, a message is printed to standard error.

2.33.7 See Also

The PBS Professional Administrator's Guide

2.34 **pbs_wish**

TK window shell with TCL-wrapped PBS API

2.34.1 Synopsis

pbs_wish

pbs_wish --version

2.34.2 Description

The `pbs_wish` command is a version of the TK window shell which includes wrapped versions of the PBS external API. The PBS TCL API is documented in the `pbs_tclapi(3B)` manual page.

2.34.3 Options

`--version`

The `pbs_wish` command returns its PBS version information and exits. This option can only be used alone.

2.34.4 Standard Error

The `pbs_wish` command will write a diagnostic message to standard error for each error occurrence.

2.34.5 See Also

The PBS Professional Administrator's Guide and the following manual pages:
`pbs_tclsh(8B)`, `pbs_mom(8B)`, `pbs_server(8B)`, `pbs_sched(8B)`

2.35 pbsdsh

Distributes task(s) to vnodes under PBS

2.35.1 Synopsis

pbsdsh [-c <copies>] [-s] [-v] [-o] -- program [program_args]

pbsdsh [-n <node_index>] [-s] [-v] [-o] -- program [program_args]

pbsdsh --version

2.35.2 Description of **pbsdsh** Command

The **pbsdsh** command allows you to distribute and execute a task on each of the vnodes assigned to your job by executing (spawning) the application on each vnode. The **pbsdsh** command uses the PBS Task Manager, or TM, to distribute the program on the allocated vnodes.

When run without the **-c** or the **-n** option, **pbsdsh** will spawn the program on all vnodes allocated to the PBS job. The spawns take place concurrently; all execute at (about) the same time.

Note that the double dash must come after the options and before the program and arguments. The double dash is only required for Linux.

The following example shows the **pbsdsh** command inside of a PBS batch job. The options indicate that the user wants **pbsdsh** to run the *myapp* program with one argument (*app-arg1*) on all four vnodes allocated to the job (i.e. the default behavior).

```
#!/bin/sh
#PBS -l select=4:ncpus=1
#PBS -l walltime=1:00:00
```

```
pbsdsh ./myapp app-arg1
```

The **pbsdsh** command runs one task for each line in the PBS_NODEFILE. Each MPI rank will get a single line in the PBS_NODEFILE, so if you are running multiple MPI ranks on the same host, you will still get multiple **pbsdsh** tasks on that host.

The **pbsdsh** command is not supported on Windows.

2.35.3 Options to **pbsdsh** Command

-c <copies>

The program is spawned *copies* times on the vnodes allocated, one per vnode, unless *copies* is greater than the number of vnodes. If *copies* is greater than the number of vnodes, it will wrap around, running multiple instances on some vnodes. This option is mutually exclusive with **-n**.

-n <node_index>

The program is spawned only on a single vnode, which is the *node_index* -th vnode allocated. This option is mutually exclusive with **-c**.

-o

No obit request is made for spawned tasks. The program will not wait for the tasks to finish.

- s**
The program is run in turn on each vnode, one after the other.
- v**
Produces verbose output about error conditions and task exit status.
- version**
The `pbsdsh` command returns its PBS version information and exits. This option can only be used alone

2.35.4 Operands

- program**
The first operand, `program`, is the program to execute. The double dash must precede the program under Linux.
- program_args**
Additional operands, `program_args`, are passed as arguments to the program.

2.35.5 Standard Error

The `pbsdsh` command writes a diagnostic message to standard error for each error occurrence.

2.35.6 See Also

The PBS Professional User's Guide, the PBS Professional Administrator's Guide, `qsub(1B)`, `tm(3)`

2.36 pbsfs

Shows or manipulates PBS fairshare usage data

2.36.1 Synopsis

```
pbsfs -[t|p]
pbsfs -g entity
pbsfs -s entity usage_value
pbsfs -d
pbsfs -e
pbsfs -c entity1 entity2
pbsfs --version
```

2.36.2 Description

The `pbsfs` command is used to print or manipulate the PBS scheduler's fairshare usage data. Some options should only be used when the scheduler is not running. There are multiple parts to a fairshare node and you can print these data in different formats. The `pbsfs` command must be run by root; otherwise it will print the error message, "Unable to access fairshare data".

The data:

fairshare entity

The entity in the fairshare tree.

group

The group ID the node is in (i.e. the node's parent).

cgroup

The group ID of this group

shares

The number of shares the group has

usage

The amount of usage

percentage

The percentage the entity has of the tree. Note that only the leaf nodes sum to 100%. If all of the nodes are summed, the result will be greater than 100%. Only the leaf nodes of the tree are fairshare entities.

usage / perc

The value the scheduler will use to pick which entity has priority over another. The smaller the number the higher the priority.

Path from root

The path from the root of the tree to the node. This is useful because the scheduler will look down the path to compare two nodes to see which has the higher priority.

resource

The resource for which the scheduler accumulates usage for its fairshare calculations. This defaults to `cput` (CPU seconds) but can be set in the scheduler's config file.

2.36.3 Options to `pbs fs`

Scheduler can be running or down:

-t

Print the fairshare tree in a hierarchical format.

-p

Print the fairshare tree in a flat format with more data.

-g entity

Print one entry with all data and print the path from the root of the tree to the node.

-c entity1 entity2

Compare two fairshare entities

--version

The `pbs fs` command returns its PBS version information and exits. This option can only be used alone.

Scheduler must be down:

-s entity usage_value

Set entity's usage value to `usage_value`. Please note that editing a non-leaf node is ignored. All non-leaf node usage values are calculated each time the scheduler is run/HUPed.

-d

Decay the fairshare tree (divide all values in half)

-e

Trim fairshare tree to just the entities in the `resource_group` file

2.36.4 See Also

The PBS Professional Administrator's Guide, `pbs_sched`(8B)

2.37 pbsnodes

Queries PBS host status or mark hosts free or offline

2.37.1 Synopsis

```
pbsnodes [-o | -r ] [-s server] hostname [hostname ...]
```

```
pbsnodes [-l] [-s server]
```

```
pbsnodes -a [-v] [-s server]
```

```
pbsnodes --version
```

2.37.2 Description

The **pbsnodes** command is used to query the status of hosts, or to mark hosts *FREE* or *OFFLINE*. The **pbsnodes** command obtains host information by sending a request to the PBS server.

To print the status of the specified host or hosts, run **pbsnodes** with no options (except the -s option) and with a list of hosts.

To print the command usage, run **pbsnodes** with no options and without a list of hosts.

PBS Manager or Operator privilege is required to execute **pbsnodes** with the -o , or -r options.

To remove a node from the scheduling pool, mark it *OFFLINE*. If it is marked *DOWN*, when the server next queries the MOM, and can connect, the node will be marked *FREE*.

For hosts with multiple vnodes, **pbsnodes** operates on a host and all of its vnodes, where the hostname is resources_available.host. To offline a single vnode in a multi-vnoded system, use:

```
qmgr -c "set node <vnode name> state=offline"
```

Users without operator or manager privilege cannot view custom resources which have been created to be invisible to users.

To act on individual vnodes, use the **qmgr** command.

2.37.3 Options to pbsnodes

(no options) If neither options nor a host list is given, the **pbsnodes** command prints usage syntax.

-a

Lists all hosts and all their attributes (available and used.)

When listing a host with multiple vnodes:

The output for the jobs attribute lists all the jobs on all the vnodes on that host. Jobs that run on more than one vnode will appear once for each vnode they run on.

For consumable resources, the output for each resource is the sum of that resource across all vnodes on that host.

For all other resources, e.g. string and boolean, if the value of that resource is the same on all vnodes on that host, the value is returned. Otherwise the output is the literal string “<various>”.

-l

Lists all hosts marked as *DOWN* or *OFFLINE*. Each such host’s state and comment attribute (if set) is listed. If a host also has state *STATE-UNKNOWN*, that will be listed. For hosts with multiple vnodes, only hosts where all vnodes are marked as *DOWN* or *OFFLINE* are listed.

-o host_list

Marks listed hosts as *OFFLINE* even if currently in use. This is different from being marked *DOWN*. A host that is marked *OFFLINE* will continue to execute the jobs already on it, but will be removed from the scheduling pool (no more jobs will be scheduled on it.)

For hosts with multiple vnodes, `pbsnodes` operates on a host and all of its vnodes, where the hostname is `resources_available.host`, which is the name of the natural vnode. To offline a single vnode in a multi-vnoded system, use:

```
qmgr: qmgr -c "set node <vnode name> state=offline"
```

Requires PBS Manager or Operator privilege.

-r host_list

Clears *OFFLINE* from listed hosts.

-s server

Specifies the PBS server to which to connect.

-v

Can only be used with the **-a** option. Prints one entry for each vnode in the PBS complex. (Information for all hosts is displayed.)

The output for the jobs attribute for each vnode lists the jobs executing on that vnode. The output for resources and attributes lists that for each vnode.

--version

The `pbsnodes` command returns its PBS version information and exits.
This option can only be used alone.

2.37.4 Operands

server

Specifies the server to which to connect. Default: default server.

host_list

Specifies the host(s) whose status will be returned. Format:

hostname [hostname ...]

2.37.5 Exit Status

Zero upon success.

Greater than zero, if:

- incorrect operands are given,
- `pbsnodes` cannot connect to the server,
- there is an error querying the server for the nodes.

2.37.6 See Also

The PBS Professional Administrator's Guide, `pbs_server(8B)` and `qmgr(8B)`

2.38 pbsrun

General-purpose wrapper script for `mpirun`

2.38.1 Synopsis

pbsrun

pbsrun --version

2.38.2 Description

`pbsrun` is a wrapper script for any of several versions of `mpirun`. This provides a user-transparent way for PBS to control jobs which call `mpirun` in their job scripts. The `pbsrun_wrap` script instantiates `pbsrun` so that the wrapper script for the specific version of `mpirun` being used has the same name as that version of `mpirun`.

If the `mpirun` wrapper script is run inside a PBS job, then it will translate any `mpirun` call of the form:

```
mpirun [options] <executable> [args]
```

into

```
mpirun [options] pbs_attach [special_option_to_pbs_attach] <executable>
[args]
```

where [special options] refer to any option needed by `pbs_attach` to do its job (e.g. `-j $PBS_JOBID`).

If the wrapper script is executed outside of PBS, a warning is issued about “not running under PBS”, but it proceeds as if the actual program had been called in standalone fashion.

The `pbsrun` wrapper script is not meant to be executed directly but instead it is instantiated by `pbsrun_wrap`. It is copied to the target directory and renamed “`pbsrun.<mpirun version/flavor>`” where `<mpirun version/flavor>` is a string that identifies the `mpirun` version being wrapped (e.g. `ch_gm`).

The `pbsrun` script, if executed inside a PBS job, runs an initialization script, named `$PBS_EXEC/lib/MPI/pbsrun.<mpirun version/flavor>.init`, then parses `mpirun`-like arguments from the command line, sorting which options and option values to retain, to ignore, or to transform, before calling the actual `mpirun` script with a “`pbs_attach`” prefixed to the executable. The actual `mpirun` to call is found by tracing the link pointed to by `$PBS_EXEC/lib/MPI/pbsrun.<mpirun version/flavor>.link`.

For all of the wrapped MPIS, the maximum number of ranks that can be launched is the number of entries in `$PBS_NODEFILE`.

The wrapped MPIs are:

- MPICH-GM's `mpirun` (`mpirun.ch_gm`) with `rsh/ssh`
- MPICH-MX's `mpirun` (`mpirun.ch_mx`) with `rsh/ssh`
- MPICH-GM's `mpirun` (`mpirun.mpd`) with MPD
- MPICH-MX's `mpirun` (`mpirun.mpd`) with MPD
- MPICH2's `mpirun`
- Intel MPI's `mpirun`
- MVAPICH1's `mpirun`
- MVAPICH2's `mpiexec`
- IBM's `poe`

2.38.3 Options

`--version`

The `pbsrun` command returns its PBS version information and exits. This option can only be used alone.

2.38.4 Initialization Script

The initialization script, called `$PBS_EXEC/lib/MPI/pbsrun.<mpirun version/ flavor>.init`, where `<mpirun version/ flavor>` reflects the `mpirun` flavor/version being wrapped, can be modified by an administrator to customize against the local flavor/version of `mpirun` being wrapped.

Inside this sourced init script, 8 variables are set:

```
options_to_retain="-optA -optB <val> -optC <val1> val2> ..."
options_to_ignore="-optD -optE <n> -optF <val1> val2> ..."
options_to_transform="-optG -optH <val> -optI <val1> val2> ..."
options_to_fail="-optY -optZ ..."
options_to_configfile="-optX <val> ..."
options_with_another_form="-optW <val> ..."
pbs_attach=pbs_attach
options_to_pbs_attach="-J $PBS_JOBID"
```

2.38.4.1 Initialization Script Options

options_to_retain

Space-separated list of options and values that `pbsrun.<mpirun version/flavor>` passes on to the actual `mpirun` call. options must begin with “-” or “--”, and option arguments must be specified by some arbitrary name with left and right arrows, as in “<val1>”.

options_to_ignore

Space-separated list of options and values that `pbsrun.<mpirun version/flavor>` does not pass on to the actual `mpirun` call. Options must begin with “-” or “--”, and option arguments must be specified by arbitrary names with left and right arrows, as in “<n>”.

options_to_transform

Space-separated list of options and values that `pbsrun` modifies before passing on to the actual `mpirun` call.

option_to_fail

Space-separated list of options that will cause `pbsrun` to exit upon encountering a match.

options_to_configfile

Single option and value that refers to the name of the “configfile” containing command line segments found in certain versions of `mpirun`.

options_with_another_form

Space-separated list of options and values that can be found in `options_to_retain`, `options_to_ignore`, or `options_to_transform`, whose syntax has an alternate, unsupported form.

pbs_attach

Path to `pbs_attach`, which is called before the `<executable>` argument of `mpirun`.

options_to_pbs_attach

Special options to pass to the `pbs_attach` call. You may pass variable references (e.g. `$PBS_JOBID`) and they are substituted by `pbsrun` to actual values.

If `pbsrun` encounters any option not found in `options_to_retain`, `options_to_ignore`, and `options_to_transform`, then it is flagged as an error.

These functions are created inside the init script. These can be modified by the PBS administrator.

```
transform_action () {
# passed actual values of $options_to_transform
args=$*
}
```

```
boot_action () {
mpirun_location=$1
}
```

```
evaluate_options_action () {
# passed actual values of transformed options
args=$*
}
```

```
configfile_cmdline_action () {
args=$*
}
```

```
end_action () {
mpirun_location=$1
}
```

transform_action()

The `pbsrun.<mpirun version/flavor>` wrapper script invokes the function `transform_action()` (called once on each matched item and value) with actual options and values received matching one of the “options_to_transform”. The function returns a string to pass on to the actual `mpirun` call.

boot_action()

Performs any initialization tasks needed before running the actual `mpirun` call. For instance, GM's MPD requires the MPD daemons to be user-started first. This function is called by the `pbsrun.<mpirun version/flavor>` script with the location of actual `mpirun` passed as the first argument. Also, the `pbsrun.<mpirun version/flavor>` checks for the exit value of this function to determine whether or not to progress to the next step.

evaluate_options_action()

Called with the actual options and values that resulted after consulting `options_to_retain`, `options_to_ignore`, `options_to_transform`, and executing `transform_action()`. This provides one more chance for the script writer to evaluate all the options and values in general, and make any necessary adjustments, before passing them on to the actual `mpirun` call. For instance, this function can specify what the default value is for a missing `-np` option.

configfile_cmdline_action()

Returns the actual options and values to be put in before the `option_to_configfile` parameter.

configfile_firstline_action()

Returns the item that is put in the first line of the configuration file specified in the `option_to_configfile` parameter.

end_action()

Called by `pbsrun.<mpirun version/flavor>` at the end of execution. It undoes any action done by `transform_action()`, like cleanup of temporary files. It is also called when `pbsrun.<mpirun version/flavor>` is prematurely killed. This function is called with the location of actual `mpirun` passed as first argument.

The actual `mpirun` program to call is the path pointed to by `$PBS_EXEC/lib/MPI/pbsrun.<mpirun version/flavor>.link`."

2.38.4.2 Modifying *.init Scripts

In order for administrators to modify `*.init` scripts without breaking package verification in RPM, master copies of the initialization scripts are named `*.init.in`. `pbsrun_wrap` instantiates the `*.init.in` files as `*.init`. For instance, `$PBS_EXEC/lib/MPI/pbsrun.mpich2.init.in` is the master copy, and `pbsrun_wrap` instantiates it as `$PBS_EXEC/lib/MPI/pbsrun.mpich2.init`. `pbsrun_unwrap` takes care of removing the `*.init` files.

2.38.5 Versions/Flavors of `mpirun`

2.38.5.1 MPICH-GM's `mpirun` (`mpirun.ch_gm`) with `rsh/ssh`: `pbsrun.ch_gm`

2.38.5.1.i Syntax

```
pbsrun.ch_gm <options> <executable> <arg1> <arg2> ... <argn>
```

The PBS wrapper script to MPICH-GM's `mpirun` (`mpirun.ch_gm`) with `rsh/ssh` process startup method is named `pbsrun.ch_gm`.

If executed inside a PBS job, this allows for PBS to track all MPICH-GM processes started by `rsh/ssh` so that PBS can perform accounting and have complete job control.

If executed outside of a PBS job, it behaves exactly as if standard `mpirun.ch_gm` was used.

2.38.5.1.ii Options Handling

If executed inside a PBS job script, all `mpirun.ch_gm` options given are passed on to the actual `mpirun` call with these exceptions:

`-machinefile` <file>

The file argument contents are ignored and replaced by the contents of the `$PBS_NODEFILE`.

`-np`

If not specified, the number of entries found in the `$PBS_NODEFILE` is used.

`-pg`

The use of the `-pg` option, for having multiple executables on multiple hosts, is allowed but it is up to user to make sure only PBS hosts are specified in the process group file; MPI processes spawned are not guaranteed to be under the control of PBS.

2.38.5.1.iii Wrap/Unwrap

To wrap MPICH-GM's `mpirun` script:

```
# pbsrun_wrap [MPICH-GM_BIN_PATH]/mpirun.ch_gm pbsrun.ch_gm
```

To unwrap MPICH-GM's `mpirun` script:

```
# pbsrun_unwrap pbsrun.ch_gm
```

2.38.5.2 MPICH-MX's `mpirun` (`mpirun.ch_mx`) with `rsh/ssh`: `pbsrun.ch_mx`

2.38.5.2.i Syntax

pbsrun.ch_mx <options> <executable> <arg1> <arg2> ... <argn>

The PBS wrapper script to MPICH-MX's `mpirun` (`mpirun.ch_gm`) with `rsh/ssh` process startup method is named `pbsrun.ch_mx`.

If executed inside a PBS job, this allows for PBS to track all MPICH-MX processes started by `rsh/ssh` so that PBS can perform accounting and has complete job control.

If executed outside of a PBS job, it behaves exactly as if standard `mpirun.ch_mx` was used.

2.38.5.2.ii Options HANDLING

If executed inside a PBS job script, all `mpirun.ch_gm` options given are passed on to the actual `mpirun` call with some exceptions:

-machinefile <file>

The file argument contents is ignored and replaced by the contents of the `$PBS_NODEFILE`.

-np

If not specified, the number of entries found in the `$PBS_NODEFILE` is used.

-pg

The use of the `-pg` option, for having multiple executables on multiple hosts, is allowed but it is up to user to make sure only PBS hosts are specified in the process group file; MPI processes spawned are not guaranteed to be under the control of PBS.

2.38.5.2.iii Wrap/Unwrap

To wrap MPICH-MX's `mpirun` script:

```
# pbsrun_wrap [MPICH-MX_BIN_PATH]/mpirun.ch_mx pbsrun.ch_mx
```

To unwrap MPICH-MX's `mpirun` script:

```
# pbsrun_unwrap pbsrun.ch_mx
```

2.38.5.3 MPICH-GM's `mpirun` (`mpirun.mpd`) with MPD: `pbsrun.gm_mpd`

2.38.5.3.i Syntax

pbsrun.gm_mpd <options> <executable> <arg1> <arg2> ... <argn>

The PBS wrapper script to MPICH-GM's `mpirun` (`mpirun.ch_gm`) with MPD process startup method is called `pbsrun.gm_mpd`.

If executed inside a PBS job, this allows for PBS to track all MPICH-GM processes started by the MPD daemons so that PBS can perform accounting have and complete job control.

If executed outside of a PBS job, it behaves exactly as if standard `mpirun.ch_gm` with MPD was used.

2.38.5.3.ii Options Handling

If executed inside a PBS job script, all `mpirun.ch_gm` with MPD options given are passed on to the actual `mpirun` call with these exceptions:

-m <file>

The file argument contents are ignored and replaced by the contents of the `$PBS_NODEFILE`.

-np

If not specified, the number of entries found in the `$PBS_NODEFILE` is used.

-pg

The use of the `-pg` option, for having multiple executables on multiple hosts, is allowed but it is up to user to make sure only PBS hosts are specified in the process group file; MPI processes spawned are not guaranteed to be under the control of PBS.

2.38.5.3.iii Startup/Shutdown

The script starts MPD daemons on each of the unique hosts listed in `$PBS_NODEFILE`, using either `rsh` or `ssh` method based on value of environment variable `RSHCOMMAND`. The default is `rsh`.

The script also takes care of shutting down the MPD daemons at the end of a run.

2.38.5.3.iv Wrap/Unwrap

To wrap MPICH-GM's `mpirun` script with MPD:

```
# pbsrun_wrap [MPICH-GM_BIN_PATH]/mpirun.mpd pbsrun.gm_mpd
```

To unwrap MPICH-GM's `mpirun` script with MPD:

```
# pbsrun_unwrap pbsrun.gm_mpd
```

2.38.5.4 MPICH-MX's `mpirun` (`mpirun.mpd`) with MPD: `pbsrun.mx_mpd`

2.38.5.4.i Syntax

```
pbsrun.mx_mpd <options> <executable> <arg1> <arg2> ... <argn>
```

The PBS wrapper script to MPICH-MX's `mpirun` (`mpirun.ch_mx`) with MPD process startup method is called `pbsrun.mx_mpd`.

If executed inside a PBS job, this allows for PBS to track all MPICH-MX processes started by the MPD daemons so that PBS can perform accounting and have complete job control.

If executed outside of a PBS job, it behaves exactly as if standard `mpirun.ch_mx` with MPD was used.

2.38.5.4.ii Options Handling

If executed inside a PBS job script, all `mpirun.mx_mpd` with MPD options given are passed on to the actual `mpirun` call with these exceptions:

-m <file>

The file argument contents are ignored and replaced by the contents of the `$PBS_NODEFILE`.

-np

If not specified, the number of entries found in the `$PBS_NODEFILE` is used.

-pg

The use of the `-pg` option, for having multiple executables on multiple hosts, is allowed but it is up to user to make sure only PBS hosts are specified in the process group file; MPI processes spawned are not guaranteed to be under the control of PBS.

2.38.5.4.iii Startup/Shutdown

The script starts MPD daemons on each of the unique hosts listed in `$PBS_NODEFILE`, using either `rsh` or `ssh` method, based on value of environment variable `RSHCOMMAND` - `rsh` is the default.

The script also takes care of shutting down the MPD daemons at the end of a run.

2.38.5.4.iv Wrap/Unwrap

To wrap MPICH-MX's `mpirun` script with MPD:

```
# pbsrun_wrap [MPICH-MX_BIN_PATH]/mpirun.mpd pbsrun.mx_mpd
```

To unwrap MPICH-MX's `mpirun` script with MPD:

```
# pbsrun_unwrap pbsrun.mx_mpd
```

2.38.5.5 MPICH2's `mpirun: pbsrun.mpich2`

2.38.5.5.i Syntax

```
pbsrun.mpich2 [global args] [local args] executable [args] [: [local args] executable [args]]
```

- or -

```
pbsrun.mpich2 -configfile <configfile>
```

where <`configfile`> contains command line segments as lines:

```
[local args] executable1 [args]
```

```
[local args] executable2 [args]
```

```
[local args] executable3 [args]
```

The PBS wrapper script to MPICH2's `mpirun` is called `pbsrun.mpich2`.

If executed inside a PBS job, this allows for PBS to track all MPICH2 processes so that PBS can perform accounting and have complete job control.

If executed outside of a PBS job, it behaves exactly as if standard MPICH2's `mpirun` was used.

2.38.5.5.ii Options Handling

If executed inside a PBS job script, all MPICH2's `mpirun` options given are passed on to the actual `mpirun` call with these exceptions:

-host and -ghost

For specifying the execution host to run on. Not passed on to the actual `mpirun` call.

-machinefile <file>

The file argument contents are ignored and replaced by the contents of the `$PBS_NODEFILE`.

MPICH2's `mpirun -localonly <x>`

For specifying the <`x`> number of processes to run locally. Not supported. The user is advised instead to use the equivalent arguments: `-np <x> -localonly`. The reason for this is that the `pbsrun` wrapper script cannot handle

a variable number of arguments to an option (e.g. “-localonly” has 1 argument and “-localonly <x>” has 2 arguments).

-np

If user did not specify a **-np** option, then no default value is provided by the PBS wrapper scripts. It is up to the local `mpirun` to decide what the reasonable default value should be, which is usually 1.

2.38.5.5.iii Startup/Shutdown

The script takes care of ensuring that the MPD daemons on each of the hosts listed in the `$PBS_NODEFILE` are started. It also takes care of ensuring that the MPD daemons have been shut down at the end of MPI job execution.

2.38.5.5.iv Wrap/Unwrap

To wrap MPICH2's `mpirun` script:

```
# pbsrun_wrap [MPICH2_BIN_PATH]/mpirun pbsrun.mpich2
```

To unwrap MPICH2's `mpirun` script:

```
# pbsrun_unwrap pbsrun.mpich2
```

In the case where MPICH2 uses `mpirun.py`, run `pbsrun_wrap` on `mpirun.py` itself.

2.38.5.6 Intel MPI's `mpirun`: `pbsrun.intelmpi`

2.38.5.6.i Syntax

```
pbsrun.intelmpi [mpdboot options] [mpiexec options] executable [prog-args] [: [mpiexec options] executable [prog-args]]
```

- or -

```
pbsrun.intelmpi [mpdboot options] -f <configfile>
```

where `[mpdboot options]` are any options to pass to the `mpdboot` program, which is automatically called by Intel MPI's `mpirun` to start MPDs, and `<configfile>` contains command line segments as lines.

The PBS wrapper script to Intel MPI's `mpirun` is called `pbsrun.intelmpi`.

If executed inside a PBS job, this allows for PBS to track all Intel MPI processes so that PBS can perform accounting and have complete job control.

If executed outside of a PBS job, it behaves exactly as if standard Intel MPI's `mpirun` was used.

2.38.5.6.ii Options Handling

If executed inside a PBS job script, all of the options to the PBS interface to MPI's `mpirun` are passed to the actual `mpirun` call with these exceptions:

-host and -ghost

For specifying the execution host to run on. Not passed on to the actual `mpirun` call.

-machinefile <file>

The file argument contents are ignored and replaced by the contents of the `$PBS_NODEFILE`.

mpdboot options --totalnum=* and --file=*

Ignored and replaced by the number of unique entries in `$PBS_NODEFILE` and name of `$PBS_NODEFILE` respectively.

arguments to mpdboot options --file=* and -f <mpd_hosts_file>

Replaced by `$PBS_NODEFILE`.

-s

If `pbsrun.intelmpi` is called inside a PBS job, Intel MPI's `mpirun -s` argument to `mpdboot` are not supported as this closely matches the `mpirun` option `-s <spec>`. The user can simply run a separate `mpdboot -s` before calling `mpirun`. A warning message is issued by `pbsrun.intelmpi` upon encountering a `-s` option telling users of the supported form.

-np

If the user does not specify a `-np` option, then no default value is provided by the PBS wrap scripts. It is up to the local `mpirun` to decide what the reasonable default value should be, which is usually 1.

2.38.5.6.iii Startup/Shutdown

Intel MPI's `mpirun` itself takes care of starting/stopping the MPD daemons.

`pbsrun.intelmpi` always passes the arguments `-totalnum=<number of mpds to start>` and `-file=<mpd_hosts_file>` to the actual `mpirun`, taking its input from unique entries in `$PBS_NODEFILE`.

2.38.5.6.iv Wrap/Unwrap

To wrap Intel MPI's `mpirun` script:

```
# pbsrun_wrap [INTEL_MPI_BIN_PATH]/mpirun pbsrun.intelmpi
```

To unwrap Intel MPI's `mpirun` script:

```
# pbsrun_unwrap pbsrun.intelmpi
```

2.38.5.7 MVAPICH1's `mpirun`: `pbsrun.mvapich1`

2.38.5.7.i Syntax

pbsrun.mvapich1 <mpirun options> <executable> <options>

The PBS wrapper script to MVAPICH1's `mpirun` is called `pbsrun.mvapich1`.

Only one executable can be specified. MVAPICH1 allows the use of InfiniBand.

If executed inside a PBS job, this allows for PBS to be aware of all MVAPICH1 ranks and track their resources, so that PBS can perform accounting and have complete job control.

If executed outside of a PBS job, it behaves exactly as if standard `mpirun` was used.

2.38.5.7.ii Options Handling

If executed inside a PBS job script, all `mpirun` options given are passed on to the actual `mpirun` call with these exceptions:

`-map` <list>

The `map` option is ignored.

`-exclude` <list>

The `exclude` option is ignored.

`-machinefile` <file>

The `machinefile` option is ignored.

`-np`

If not specified, the number of entries found in the `$PBS_NODEFILE` is used.

2.38.5.7.iii Wrap/Unwrap

To wrap MVAPICH1's `mpirun` script:

```
# pbsrun_wrap <path-to-actual-mpirun> pbsrun.mvapich1
```

To unwrap MVAPICH1's `mpirun` script:

```
# pbsrun_unwrap pbsrun.mvapich1
```

2.38.5.8 MVAPICH2's `mpiexec`: `pbsrun.mvapich2`

2.38.5.8.i Syntax

pbsrun.mvapich2 <mpiexec args> executable <executable's args> [: <mpiexec args> executable <executable's args>]

The PBS wrapper script to MVAPICH2's `mpiexec` is called `pbsrun.mvapich2`.

Multiple executables can be specified using the colon notation. MVAPICH2 allows the use of InfiniBand.

If executed inside a PBS job, this allows for PBS to be aware of all MVAPICH2 ranks and track their resources, so that PBS can perform accounting and have complete job control.

If executed outside of a PBS job, it behaves exactly as if standard `mpiexec` was used.

2.38.5.8.ii Options Handling

If executed inside a PBS job script, all `mpiexec` options given are passed on to the actual `mpiexec` call with these exceptions:

`-host <host>`

The host argument contents are ignored.

`-machinefile <file>`

The file argument contents are ignored and replaced by the contents of the `$PBS_NODEFILE`.

2.38.5.8.iii Wrap/Unwrap

To wrap MVAPICH2's `mpiexec` script:

```
# pbsrun_wrap <path-to-actual-mpiexec> pbsrun.mvapich2
```

To unwrap MVAPICH2's `mpiexec` script:

```
# pbsrun_unwrap pbsrun.mvapich2
```

2.38.5.9 IBM's `poe: pbsrun.poe`

2.38.5.9.i Syntax

```
pbsrun.poe <options> <executable> <arg1> <arg2> ... <argn>
```

The PBS wrapper script to IBM's `poe` is called `pbsrun.poe`.

MPI is supported under IBM's Parallel Operating Environment (POE) on AIX. Under AIX, the program `poe` is used to start user processes on remote machines. PBS will manage the IBM HPS in US (User Space) mode.

The PBS wrapper script to IBM's `poe` allows LAPI or MPI programs to use InfiniBand or the HPS in US mode.

If executed inside a PBS job, this allows for PBS to track all resources and MPI ranks. PBS can perform accounting and have complete job control.

If executed outside of a PBS job, it behaves exactly as if standard `poe` was used.

The script will use the `-eulib {ip | us}` option and the `MP_EULIB` environment variable to indicate use of US mode, to maintain compatibility with standard `poe`.

2.38.5.9.ii Options and Environment Variables

Users submitting jobs whose programs use `poe` can set environment variables instead of using options to `poe`. The equivalent environment variable is listed with its `poe` option. If executed inside a PBS job script, all `pbsrun.poe` options and environment variables except the following are passed on to `poe`:

-devtype, `MP_DEVTYPE`

If InfiniBand is not specified in either the option or the environment variable, the InfiniBand interconnect is not used for the program.

-euiddevice, `MP_EUIDEVICE`

Ignored by PBS.

-eulib {ip|us}, `MP_EULIB`

If the command line option `-eulib` is set, it will take precedence over the `MP_EULIB` environment variable. If set to “*us*”, the program uses User Space mode. If set to any other value, that value is passed to IBM `poe`.

-hostfile, -hfile, `MP_HOSTFILE`

Ignored. If this is specified, PBS prints the following:

```
"pbsrun.poe: Warning, -hostfile value replaced by PBS"
```

or

```
"pbsrun.poe: Warning -hfile value replaced by PBS"
```

If this environment variable is set when a `poe` job is submitted, PBS prints the following error message:

```
"pbsrun.poe: Warning MP_HOSTFILE value replaced by PBS"
```

-instances, `MP_INSTANCES`

The option and the environment variable are treated differently:

-instances

If the option is set, PBS prints a warning:

```
"pbsrun.poe: Warning, -instances cmd line option removed by PBS"
```

`MP_INSTANCES`

If the environment variable is set, PBS uses it to calculate the number of network windows for the job. The maximum value allowed can be requested by using the string “*max*” for the environment variable. If the environment variable is set to a value greater than the maximum allowed value, it is replaced with the maximum allowed value. The default maximum value is 4.

-procs, MP_PROCS

This option or environment variable should be set to the total number of `mpiprocs` requested by the job when using US mode. If neither this option nor the `MP_PROCS` environment variable is set, PBS uses the number of entries in `$PBS_NODEFILE`. If this option is set to N , and the job is submitted with a total of M `mpiprocs`:

If $N \geq M$:

The value N is passed to IBM `poe`.

If $N < M$ and US mode is not being used:

The value N is passed to `poe`.

If $N < M$ and US mode is being used:

US mode is turned off and a warning is printed:

`"pbsrun.poe: Warning, user mode disabled due to MP_PROCS setting"`

2.38.5.9.iii Wrap/Unwrap

To wrap IBM's `poe`:

```
# pbsrun_wrap <path_to_actual_poe> pbsrun.poe
```

To unwrap the IBM `poe`:

```
# pbsrun_unwrap pbsrun.poe
```

2.38.6 Requirements

The `mpirun` being wrapped must be installed and working on all the nodes in the PBS cluster.

2.38.7 Errors

If `pbsrun` encounters any option not found in `options_to_retain`, `options_to_ignore`, and `options_to_transform`, then it is flagged as an error.

2.38.8 See Also

The PBS Professional Administrator's Guide

`pbs_attach(8B)`, `pbsrun_wrap(8B)`, `pbsrun_unwrap(8B)`

2.39 pbsrun_unwrap

Unwraps `mpirun`, reversing `pbsrun_wrap`

2.39.1 Synopsis

pbsrun_unwrap pbsrun.<mpirun version/flavor>

pbsrun_unwrap --version

2.39.2 Description

The `pbsrun_unwrap` script is used to reverse the actions of the `pbsrun_wrap` script.

Use `pbsrun_wrap` to wrap `mpirun`.

2.39.3 Usage

2.39.3.1 Syntax:

pbsrun_unwrap pbsrun.<mpirun version/flavor>

For example, running the following:

pbsrun_unwrap pbsrun.ch_gm

causes the following actions:

Checks for a link in `$PBS_EXEC/lib/MPI/pbsrun.ch_gm.link`; If one exists, get the pathname it points to:

```
/opt/mpich-gm/bin/mpirun.ch_gm.actual
```

```
rm $PBS_EXEC/lib/MPI/pbsrun.mpirun.ch_gm.link
```

```
rm /opt/mpich-gm/bin/mpirun.ch_gm
```

```
rm $PBS_EXEC/bin/pbsrun.ch_gm
```

```
mv /opt/mpich-gm/bin/mpirun.ch_gm.actual /opt/mpich-gm/bin/mpirun.ch_gm
```

2.39.4 Options

--version

The `pbsrun_unwrap` command returns its PBS version information and exits. This option can only be used alone.

2.39.5 See Also

The PBS Professional Administrator's Guide

`pbs_attach(8B)`, `pbsrun(8B)`, `pbsrun_wrap(8B)`

2.40 pbsrun_wrap

General-purpose script for wrapping `mpirun` in `pbsrun`

2.40.1 Synopsis

`pbsrun_wrap [-s] <path_to_actual_mpirun> pbsrun.<mpirun version/flavor>`

`pbsrun_wrap --version`

2.40.2 Description

The `pbsrun_wrap` script is used to wrap any of several versions of `mpirun` in `pbsrun`. The `pbsrun_wrap` script creates a symbolic link with the same path and name as the `mpirun` being wrapped. This calls `pbsrun`, which uses `pbs_attach` to give MOM control of jobs. The result is transparent to the user; when `mpirun` is called from inside a PBS job, PBS can monitor and control the job, but when `mpirun` is called from outside of a PBS job, it behaves as it would normally. See the `pbs_attach(8B)` and `pbsrun(8B)` man pages.

Use `pbsrun_unwrap` to reverse the process.

2.40.3 Options

-s

Sets the “`strict_pbs`” options in the various initialization scripts (e.g. `pbsrun.bgl.init`, `pbsrun.ch_gm.init`, etc...) to `1` from the default `0`. This means that the `mpirun` being wrapped by `pbsrun` will only be executed if inside a PBS environment. Otherwise, the user will get the error:

Not running under PBS exiting since `strict_pbs` is enabled; execute only in PBS

--version

The `pbsrun_wrap` command returns its PBS version information and exits. This option can only be used alone.

2.40.4 USAGE

2.40.4.1 Syntax:

```
pbsrun_wrap [-s] <path_to_actual_mpirun> pbsrun.<mpirun version/flavor>
```

Any `mpirun` version/flavor that can be wrapped has an initialization script ending in “.init”, found in `$PBS_EXEC/lib/MPI`:

```
$PBS_EXEC/lib/MPI/pbsrun.<mpirun version/flavor>.init.
```

The `pbsrun_wrap` script instantiates the `pbsrun` wrapper script as `pbsrun.<mpirun version/flavor>` in the same directory where `pbsrun` is located, and sets up the link to actual `mpirun` call via the symbolic link

```
$PBS_EXEC/lib/MPI/pbsrun.<mpirun version/flavor>.link
```

For example, running:

```
pbsrun_wrap /opt/mpich-gm/bin/mpirun.ch_gm pbsrun.ch_gm
```

causes the following actions:

Save original `mpirun.ch_gm` script:

```
mv /opt/mpich-gm/bin/mpirun.ch_gm /opt/mpich-gm/bin/mpirun.ch_gm.actual
```

Instantiate `pbsrun` wrapper script as `pbsrun.ch_gm`:

```
cp $PBS_EXEC/bin/pbsrun $PBS_EXEC/bin/pbsrun.ch_gm
```

Link “`mpirun.ch_gm`” to actually call “`pbsrun.ch_gm`”:

```
ln -s $PBS_EXEC/bin/pbsrun.ch_gm /opt/mpich-gm/bin/mpirun.ch_gm
```

Create a link so that “`pbsrun.ch_gm`” calls “`mpirun.ch_gm.actual`”:

```
ln -s /opt/mpich-gm/bin/mpirun.ch_gm.actual $PBS_EXEC/lib/MPI/  
pbsrun.ch_gm.link
```

2.40.5 Requirements

The `mpirun` being wrapped must be installed and working on all the nodes in the PBS cluster.

2.40.6 See Also

The PBS Professional Administrator’s Guide

`pbs_attach(8B)`, `pbsrun(8B)`, `pbsrun_unwrap(8B)`

2.41 printjob

Prints job information

2.41.1 Synopsis

```
printjob [-a | -s ] job ID
printjob [-a ] <file path> [<file path>...]
printjob --version
```

2.41.2 Description

The `printjob` command is used to print job information. You can print information either from the server, using a job ID, or from the execution host, using a file path.

Whether or not a MOM is running on the server host, you must use the job ID at the server host.

By default all the job data including job attributes are printed. This can be suppressed with the `-a` option.

You can print out just the job script using the `-s` option at the server and execution hosts.

This command is mainly useful for troubleshooting, as during normal operation, the `qstat (8B)` command is the preferred method for displaying job-specific data and attributes.

2.41.3 Permissions

In order to execute `printjob`, the user must have root or Windows Administrator privilege.

2.41.4 Options to `printjob`

- (no options> Prints all job data including job attributes.
- a Suppresses the printing of job attributes. Cannot be used with `-s` option.
- s Prints out the job script only. Cannot be used with `-a` option. Cannot be used with *file path* argument.

--version

The `printjob` command returns its PBS version information and exits. This option can only be used alone.

2.41.5 Operands for `printjob`

<file path>

The `printjob` command accepts one or more file path operands at the execution host. Files are found in `PBS_HOME/mom_priv/jobs/` on the primary execution host. File path must include full path to file. Cannot be used with `-s` option.

<job ID>

The `printjob` command accepts a job ID at the server host. The format is described in [section , “Job Identifier”, on page 406](#). Data service must be running.

2.41.6 Standard Error

The `printjob` command writes a diagnostic message to standard error for each error occurrence.

2.41.7 Exit Status

Zero upon successful processing of all the operands presented to the `printjob` command. Greater than zero if the `printjob` command fails to process any operand.

2.41.8 See Also

The PBS Professional Administrator’s Guide, [section 2.30, “pbs_server”, on page 90](#), and [section 2.57, “qstat”, on page 194](#)

2.42 `qalter`

Alters a PBS job

2.42.1 Synopsis

```
qalter [-a date_time] [-A account_string] [-c interval] [-e path] [-h hold_list] [-j join] [-k
keep] [-l resource_list] [-m mail_events] [-M user_list] [-N name] [-o path] [-p priority]
[-P project] [-r c] [-S path] [-u user_list] [-W additional_attributes] job_identifier_list
qalter --version
```

2.42.2 Description

The `qalter` command is used to alter one or more PBS batch jobs. The attributes listed with the options to the `qalter` command can be modified. If any of the modifications to a job fails, none of the job's attributes is modified.

A job that is in the process of provisioning cannot be altered.

2.42.2.1 Required privilege

- A non-privileged user may only lower the limits for resources
- A Manager or Operator may lower or raise requested resource limits, except for per-process limits such as `pcput` and `pmem`, because these are set when the process starts, and enforced by the kernel.
- The `qalter` command cannot be used by a non-privileged user to alter a custom resource which has been created to be invisible or read-only for users.

2.42.2.2 Modifying resources and job placement:

If a job is running, the only resources that can be modified are `cput` (CPU time) and `walltime`.

If a job is queued, any resource mentioned in the options to the `qalter` command can be modified, but requested modifications must fit within the limits set at the server and queue for the amount of each resource allocated for queued jobs. If a requested modification does not fit within these limits, the modification is rejected.

Note that a job's resource request must fit within the queue's and server's resource run limits. If a modification to a resource exceeds the amount of the resource allowed by the queue or server to be used by running jobs, the job is never run.

Resources are modified by using the `-l` option, either in chunks inside of selection statements, or in job-wide modifications using `resource_name=value` pairs. The selection statement is of the form:

```
-l select=[N:]chunk[+[N:]chunk ...]
```

where *N* specifies how many of that chunk, and a chunk is of the form:

resource_name=value[:resource_name=value ...]

Job-wide *resource_name=value* modifications are of the form:

-l resource_name=value[,resource_name=value ...]

Placement of jobs on nodes is changed using the place statement:

-l place=modifier[:modifier]

where modifier is any combination of *group*, *excl*, and/or one of *free*|*pack*|*scatter*.

For more on resource requests, usage limits and job placement, see `pbs_resources(7B)`.

2.42.2.3 Modifying attributes:

The user alters job attributes by giving options to the `qalter` command. Each `qalter` option changes a job attribute.

The behavior of the `qalter` command may be affected by any site hooks. Site hooks can modify the job's attributes, change its routing, etc.

2.42.3 Options to `qalter`

`-a date_time`

Changes the point in time after which the job is eligible for execution. Given in pairs of digits. Sets job's `Execution_Time` attribute to `date_time`.

Format: *Datetime*

Each portion of the date defaults to the current date, as long as the next-smaller portion is in the future. For example, if today is the 3rd of the month and the specified day `DD` is the 5th, the month `MM` will be set to the current month.

If a specified portion has already passed, the next-larger portion will be set to one after the current date. For example, if the day `DD` is not specified, but the hour `hh` is specified to be 10:00 a.m. and the current time is 11:00 a.m., the day `DD` will be set to tomorrow.

The job's `Execution_Time` attribute can be altered after the job has begun execution, in which case it will not take effect until the job is rerun.

-A account_string

Replaces the accounting string associated with the job. Used for labeling accounting data. Sets job's **Account_Name** attribute to `account_string`. This attribute cannot be altered once the job has begun execution.

Format: String

-c checkpoint_spec

Changes when the job will be checkpointed. Sets job's **Checkpoint** attribute. An **\$action** script is required to checkpoint the job. This attribute can be altered after the job has begun execution, in which case the new value will not take effect until the job is rerun.

See the `pbs_mom(8B)` man page.

The argument `checkpoint_spec` can take on one of the following values:

c

Checkpoint at intervals, measured in CPU time, set on job's execution queue. If no interval set at queue, job is not checkpointed

c=<minutes of CPU time>

Checkpoint at intervals of specified number of minutes of job CPU time. This value must be > 0 . If interval specified is less than that set on job's execution queue, queue's interval is used.

Format: Integer

w

Checkpoint at intervals, measured in walltime, set on job's execution queue. If no interval set at queue, job is not checkpointed.

w=<minutes of walltime>

Checkpoint at intervals of the specified number of minutes of job walltime. This value must be greater than zero. If the interval specified is less than that set on the execution queue in which the job resides, the queue's interval is used.

Format: Integer

n

No checkpointing.

s

Checkpoint only when the server is shut down.

u

Unset. Defaults to behavior when interval argument is set to `s`.

Default: `u`.

Format: *String*.

-e path

Replaces the path to be used for the job's standard error stream. Sets job's `Error_Path` attribute to path.

Format: *[hostname:]path_name*

The path will be interpreted as follows:

path_name

If *path_name* is a relative path, then it is taken to be relative to the current working directory of the `qalter` command, where it is executing on the current host.

If *path_name* is an absolute path, then it is taken to be an absolute path on the current host where the `qalter` command is executing.

hostname:path_name

If *path_name* is a relative path, then it is taken to be relative to the user's home directory on the host named *hostname*.

If *path_name* is an absolute path, then it is the absolute path on the host named *hostname*.

If *path_name* does not include a filename, the default filename will be *jobid.ER*

If the `-e` option is not specified, the default filename for the standard error stream is used.

Format: *job_name.esquence_number*

This attribute can be altered after the job has begun execution, in which case the new value will not take effect until the job is rerun.

-h hold_list

Updates the job's hold list. Adds *hold_list* to the job's `Hold_Types` attribute. The *hold_list* is a string of one or more of the following:

Table 2-4: Hold Types

Hold Type	Meaning
<i>u</i>	Add a USER hold.
<i>o</i>	Add OTHER hold. Requires operator privilege.
<i>n</i>	Clear the holds for which the user has privilege.

-j join

This attribute can be altered after the job has begun execution, in which case the new value will not take effect until the job is rerun.

Changes whether and how to join the job's standard error and standard output streams. Sets job's **Join_Path** attribute to **join**.

Possible values of join:

Table 2-5: Join Path Options

Value	Meaning
<i>oe</i>	Standard error and standard output are merged into standard output.
<i>eo</i>	Standard error and standard output are merged into standard error.
<i>n</i>	Standard error and standard output are not merged.

This attribute can be altered after the job has begun execution, in which case the new value will not take effect until the job is rerun.

Default: not merged.

-k keep

Changes whether and which of the standard output and standard error streams will be retained on the execution host. Overrides default path names for these streams. Sets the job's **Keep_Files** attribute to **keep**.

The keep argument can take on the following values:

Table 2-6: Keep Argument Values

Option	Meaning
<i>e</i>	The standard error stream is retained on the execution host, in the job's staging and execution directory. The filename will be: <i>job_name.e<sequence number></i>
<i>o</i>	The standard output stream is retained on the execution host, in the job's staging and execution directory. The filename will be: <i>job_name.o<sequence number></i>
<i>eo, oe</i>	Both standard output and standard error streams are retained on the execution host, in the job's staging and execution directory.
<i>n</i>	Neither stream is retained.

This attribute cannot be altered once the job has begun execution.

In the case where output and/or error is retained on the execution host in a job-specific staging and execution directory created by PBS, these files are deleted when PBS deletes the directory.

Default: neither is retained.

-l resource_arg

Allows the user to change requested resources and job placement. Sets job's **Resource_list** attribute to **resource_arg**. Uses resource request syntax. Requesting a resource places a limit on its usage. Users without manager or operator privilege cannot alter a custom resource which was created to be invisible or read-only for users.

Requesting resources in chunks:

Format: *-l select=[N:]chunk[+[N:]chunk ...]*

where N specifies how many of that chunk, and a chunk is:

Format: *resource_name=value[:resource_name=value ...]*

Requesting job-wide resources:

Format: *-l resource_name=value[,resource_name=value ...]*

Specifying placement of jobs:

Format: *-l place=[arrangement][[: sharing][: grouping]*

where

arrangement is one of *free* | *pack* | *scatter*

sharing is one of *excl* | *shared*

grouping can have only one instance of *group=resource*

and where

free

Place job on any vnode(s).

pack:

All chunks will be taken from one host.

scatter

Only one chunk with any MPI processes will be taken from a host. A chunk with no MPI processes may be taken from the same node as another chunk.

excl

Only this job uses the vnodes chosen.

shared

This job can share the vnodes chosen.

group=resource

Chunks will be grouped according to a resource. All nodes in the group must have a common value for the resource, which can be either the built-in resource host or a site-defined node-level resource.

If a requested modification to a resource would exceed the job's queue's limits, the resource request will be rejected. For a running job, resources may only be reduced. Which resources can be altered is system-dependent.

If the job was submitted with an explicit "*-l select=*", then node level resources must be qaltered using the "*-l select=*" form. In this case a node level resource RES cannot be qaltered with the "*-l RES*" form.

Examples:

1. Submit the job:

```
% qsub -l select=1:ncpus=2:mem=512mb jobscript
```

Job's ID is 230

2. `qalter` the job using “-l RES” form:

```
% qalter -l ncpus=4 230
```

Error reported by `qalter`:

```
qalter: Resource must only appear in "select" specification when
select is used: ncpus 230
```

3. `qalter` the job using the “-l select=” form:

```
% qalter -l select=1:ncpus=4:mem=512mb 230
```

No error reported by `qalter`:

```
%
```

For more on resource requests, usage limits and job placement, see `pbs_resources(7B)`.

-m mail_events

Changes the set of conditions under which mail about the job is sent. Sets job's `Mail_Points` attribute to `mail_events`. The `mail_events` argument can be either “*n*” or any combination of “*a*”, “*b*”, and “*e*”.

Table 2-7: Mail Events Options

Option	Meaning
<i>n</i>	No mail will be sent.
<i>a</i>	Mail is sent when the job is aborted by the batch system.
<i>b</i>	Mail is sent when the job begins execution.
<i>e</i>	Mail is sent when the job terminates.

Format: *String*.

Default value: “*a*”.

-M user_list

Alters list of users to whom mail about the job is sent. Sets job's `Mail_Users` attribute to `user_list`.

Format: *user[@host][,user[@host],...]*

Default: job owner.

-N name

Renames the job. Sets job's `Job_Name` attribute to `name`.

Format: string, up to 15 characters in length. It must consist of an alphabetic character followed by printable, non-white-space characters.

Default: if a script is used to submit the job, the job's name is the name of the script. If no script is used, the job's name is "*STDIN*".

-o path

Alters path to be used for the job's standard output stream. Sets job's `Output_Path` attribute to `path`.

Format: *[hostname:]path_name*

The path will be interpreted as follows:

path_name

If *path_name* is a relative path, then it is taken to be relative to the current working directory of the command, where it is executing on the current host.

If *path_name* is an absolute path, then it is taken to be an absolute path on the current host where the command is executing.

hostname:path_name

If *path_name* is a relative path, then it is taken to be relative to the user's home directory on the host named *hostname*.

If *path_name* is an absolute path, then it is the absolute path on the host named *hostname*.

If *path_name* does not include a filename, the default filename will be *jobid.OU*

If the `-o` option is not specified, the default filename for the standard output stream is used. It has this form:

job_name.osequence_number

This attribute can be altered after the job has begun execution, in which case the new value will not take effect until the job is rerun.

-p priority

Alters priority of the job.

Sets job's `Priority` attribute to `priority`.

This attribute can be altered after the job has begun execution, in which case the new value will not take effect until the job is rerun.

Format: host-dependent integer.

Range: [-1024, +1023] inclusive.

Default: *zero*.

-P project

Specifies a project for the job. Sets job's **project** attribute to specified value.

Format: String.

Project name can contain any characters except for the following: Slash ("/"), left bracket ("["), right bracket ("]"), double quote ("\""), semicolon(";"), colon(":"), vertical bar("|"), left angle bracket("<"), right angle bracket(">"), plus("+"), comma(","), question mark("?"), and asterisk("*").

Default value: "*_pbs_project_default*".

-r y|n

Changes whether the job is rerunnable. Sets job's **Rerunnable** attribute to the argument. Does not affect how job is treated when the job was unable to begin execution.

See the **qrerun (1B)** command.

Format: single character, "*y*" or "*n*".

y

Job is rerunnable.

n

Job is not rerunnable.

Default: "*y*".

-S path_list

Specifies the interpreter or shell path for the job script. Sets job's **Shell_Path_List** attribute to *path_list*.

The **path_list** argument is the full path to the interpreter or shell including the executable name.

Only one path may be specified without a host name. Only one path may be specified per named host. The path selected is the one whose host name is that of the server on which the job resides.

This attribute can be altered after the job has begun execution, but in this case the new value will not take effect until the job is rerun.

Format:

path[*@host*][*,path@host ...*]

Default: user's login shell on execution node.

Example of using `bash` via a directive:

```
#PBS -S /bin/bash@mars,/usr/bin/bash@jupiter
```

Example of running a Python script from the command line on UNIX/Linux:

```
qsub -S $PBS_EXEC/bin/pbs_python <script name>
```

Example of running a Python script from the command line on Windows:

```
qsub -S %PBS_EXEC%\bin\pbs_python.exe <script name>
```

-u user_list

Alters list of usernames. Job will be run under a username from this list. Sets job's `User_List` attribute to `user_list`.

Only one username may be specified without a host name. Only one username may be specified per named host. The server on which the job resides will select first the username whose host name is the same as the server name. Failing that, the next selection will be the username with no specified hostname. The usernames on the server and execution hosts must be the same. The job owner must have authorization to run as the specified user.

This attribute cannot be altered once the job has begun execution.

Format: `user[@host]/,user@host ...]`

Default: job owner (username on submit host.)

-W additional_attributes

The `-W` option allows change in specification of additional job attributes.

Format: `-W attribute_name = value[,attribute_name=value...]`

If white space occurs within the `additional_attributes` argument, or the equal sign ("`=`") occurs within an `attribute_value` string, then that must be enclosed with single or double quotes. PBS supports the following attributes within the `-W` option:

depend=dependency_list

Defines dependencies between this and other jobs. Sets the job's `depend` attribute to `dependency_list`. The `dependency_list` has the form:

```
type:arg_list[,type:arg_list ...]
```

where except for the `on` type, the `arg_list` is one or more PBS job IDs in the form:

```
jobid[:jobid ...]
```

The type can be:

after: arg_list

This job may be scheduled for execution at any point after all jobs in *arg_list* have started execution.

afterok: arg_list

This job may be scheduled for execution only after all jobs in *arg_list* have terminated with no errors. See [section 2.42.6.1, “Warning about exit status with csh.”](#), on page 143.

afternotok: arg_list

This job may be scheduled for execution only after all jobs in *arg_list* have terminated with errors. See [section 2.42.6.1, “Warning about exit status with csh.”](#), on page 143.

afterany: arg_list

This job may be scheduled for execution after all jobs in *arg_list* have terminated, with or without errors.

before: arg_list

Jobs in *arg_list* may begin execution once this job has begun execution.

beforeok: arg_list

Jobs in *arg_list* may begin execution once this job terminates without errors. See [section 2.42.6.1, “Warning about exit status with csh.”](#), on page 143.

beforenotok: arg_list

If this job terminates execution with errors, then jobs in *arg_list* may begin. See [section 2.42.6.1, “Warning about exit status with csh.”](#), on page 143.

beforeany: arg_list

Jobs in *arg_list* may begin execution once this job terminates execution, with or without errors.

on: count

This job may be scheduled for execution after *count* dependencies on other jobs have been satisfied. This type is used in conjunction with one of the *before* types listed. *count* is an integer greater than 0.

Restrictions:

Job IDs in the *arg_list* of *before* types must have been submitted with a type of *on*.

To use the *before* types, the user must have the authority to alter the jobs in *arg_list*. Otherwise, the dependency is rejected and the new job aborted.

Error processing of the existence, state, or condition of the job on which the newly-submitted job is a deferred service, i.e. the check is performed after the job is queued. If an error is detected, the new job will be deleted by the server. Mail will be sent to the job submitter stating the error.

Dependency examples:

```
qalter -W depend = afterok:123.host1.domain.com /tmp/script
qalter -W depend= before:234.host1.com:235.host1.com /tmp/
script
```

group_list=g_list

Alters list of group names. Job will be run under a group name from this list. Sets job's `group_List` attribute to `g_list`.

Only one group name may be specified without a host name. Only one group name may be specified per named host. The server on which the job resides will select first the group name whose host name is the same as the server name. Failing that, the next selection will be the group name with no specified hostname. The group names on the server and execution hosts must be the same.

Format: *group*[*@host*][*,group@host ...*]

Default: login group name of job owner.

sandbox=<value>

Changes which directory PBS uses for the job's staging and execution.

Allowed values:

PRIVATE

PBS creates a job-specific directory for staging and execution.

HOME or **unset**

PBS uses the user's home directory for staging and execution.

Format: String

stagein=path_list

stageout=path_list

Changes files or directories to be staged-in before execution or staged-out after execution is complete. Sets the job's `stagein` and `stageout` attributes to the specified `path_lists`. On completion of the job, all

staged-in and staged-out files and directories are removed from the execution host(s). The `path_list` has the form:

filespec[.filespec]

where *filespec* is

local_path@hostname:remote_path

regardless of the direction of the copy. The name *local_path* is the name of the file or directory on the primary execution host. It can be relative to the staging and execution directory on the execution host, or it can be an absolute path.

The “@” character separates *local_path* from *remote_path*.

The name *remote_path* is the path on *hostname*. The name can be relative to the staging and execution directory on the primary execution host, or it can be an absolute path.

If `path_list` has more than one *filespec*, i.e. it contains commas, it must be enclosed in double-quotes.

umask=NNNN

Alters the umask with which the job will be started. Controls umask of job’s standard output and standard error. Sets job’s `umask` attribute to *NNNN*. Can be used with one to four digits; typically two.

The following example allows group and world read on the job’s output:

```
-W umask=33
```

Default value: *077*

--version

The `qalter` command returns its PBS version information and exits. This option can only be used alone.

2.42.4 Operands

The `qalter` command accepts a *job_identifier_list* as its operand. The *job_identifier_list* is one or more job IDs for normal jobs or array jobs. Individual subjobs of an array job are not alterable.

Note that some shells require that you enclose a job array ID in double quotes.

2.42.5 Standard Error

The `qalter` command will write a diagnostic message to standard error for each error occurrence.

2.42.6 Exit Status

Zero upon successful processing of input. Exit value will be greater than zero upon failure of `qalter`.

2.42.6.1 Warning about exit status with `csch`:

If a job is run in `csch` and a `.logout` file exists in the home directory in which the job executes, the exit status of the job is that of the `.logout` script, not the job script. This may impact any inter-job dependencies.

2.42.7 See Also

The PBS Professional User's Guide, the PBS Professional Administrator's Guide, `pbs_job_attributes(7B)`, `pbs_resources(7B)`, `qdel(1B)`, `qhold(1B)`, `qmove(1B)`, `qmsg(1B)`, `qrerun(1B)`, `qrls(1B)`, `qselect(1B)`, `qstat(1B)`, `qsub(1B)`

2.43 `qdel`

Deletes PBS jobs

2.43.1 Synopsis

```
qdel [ -x ] [ -Wforce | -Wsuppress_email=<N> ] job_identifier [job_identifier ...]  
qdel --version
```

2.43.2 Description

The `qdel` command deletes jobs in the order given, whether they are at the local server or at a remote server.

The `qdel` command is used without options to delete queued, running, held, or suspended jobs, while the `-x` option gives it the additional capacity to delete finished or moved jobs. With the `-x` option, this command can be used on finished and moved jobs, in addition to queued, running, held, or suspended jobs.

When this command is used without the `-x` option, if job history is enabled, the deleted job's history is retained. The `-x` option is used to additionally remove the history of the job being deleted.

A PBS job may be deleted by its owner, an operator, or the administrator. The server deletes a PBS job by sending a `SIGTERM` signal, then, if there are remaining processes, a `SIGKILL` signal.

If someone other than the job's owner deletes the job, mail is sent to the job's owner, or to a list of mail recipients if specified during `qsub`. See the `qsub(1B)` man page.

2.43.2.1 How Behavior of `qdel` Command Can Be Affected

The server's `default_qdel_arguments` attribute may affect the behavior of the `qdel` command. This attribute is settable by the administrator via the `qmgr` command. The attribute may be set to `"-Wsuppress_email=<N>"`. The server attribute is overridden by command line arguments. See [section 6.6, "Server Attributes", on page 314](#).

If the job is in the process of provisioning, it can be deleted only by using the `-W force` option.

2.43.2.2 Sequence of Events

1. The job's running processes are killed.
2. The epilogue runs.
3. Files that were staged in are staged out. This includes
4. standard out (.o) and standard error (.e) files.
5. Files that were staged in or out are deleted.
6. The job's temp directory is removed.
7. The job is removed from the MOM(s) and the server.

2.43.3 Options to `qdel`

(no options)

Can delete queued, running, held, or suspended jobs. Does not delete job history for specified job(s).

`-W force`

Deletes the job whether or not the job's execution host is reachable. Deletes the job whether or not the job is in the process of provisioning. Cannot be used with the `-Wsuppress_email` option.

-Wsuppress_email=<N>

Sets limit on number of emails sent when deleting multiple jobs. If $N \geq 1$ and N or more *job_identifiers* are given, N emails are sent. If $N \geq 1$ and less than N job identifiers are given, the number of emails is the same as the number of jobs. If $N = 0$, this option is ignored. If $N = -1$, no mail is sent.

Note that there is no space between “W” and “suppress_email”.

The <N> argument is an integer.

Cannot be used with -Wforce option.

-x

Can delete running, queued, suspended, held, finished, or moved jobs.

Deletes job history for the specified job(s).

--version

The qdel command returns its PBS version information and exits. This option can only be used alone.

2.43.4 Operands

The qdel command accepts one or more *job_identifier* operands. These operands can be job identifiers, job array identifiers, job array range identifiers, or subjob identifiers. See [Chapter 7, "Formats", on page 403](#).

Job array identifiers must be enclosed in double quotes for some shells.

2.43.5 Standard Error

The qdel command writes a diagnostic message to standard error for each error occurrence.

2.43.6 Exit Status

Zero upon successful processing of input.

Greater than zero upon error.

2.43.7 See Also

The PBS Professional User's Guide, the PBS Professional Administrator's Guide, `pbs_queue_attributes(7B)`, `pbs_server_attributes(1B)`, `qsub(1B)`, `qsig(1B)`, `pbs_deljob(3B)`

2.44 **qdisable**

Prevents jobs from being enqueued in a queue

2.44.1 **Synopsis**

qdisable destination ...

qdisable --version

2.44.2 **Description**

The **qdisable** command directs that a destination queue should no longer accept batch jobs. If the command is accepted, the queue will no longer accept Queue Job requests which specified the disabled queue. Jobs which already reside in the queue will continue to be processed. This allows a queue to be “drained.”

In order to execute **qdisable**, the user must have PBS Operator or Manager privilege.

2.44.3 **Options**

--version

The **qdisable** command returns its PBS version information and exits. This option can only be used alone.

2.44.4 **Operands**

The **qdisable** command accepts one or more destination operands. The operands take one of three forms:

queue

@server

queue@server

If *queue* is specified, the request is to disable that queue at the default server. If the *@server* form is given, the request is to disable all the queues at that server. If a full destination identifier, *queue@server*, is given, the request is to disable the named queue at the named server.

2.44.5 Standard Error

The `qdisable` command will write a diagnostic message to standard error for each error occurrence.

2.44.6 Exit Status

Upon successful processing of all the operands presented to the `qdisable` command, the exit status will be a value of zero.

If the `qdisable` command fails to process any operand, the command exits with a value greater than zero.

2.44.7 See Also

The PBS Professional Administrator's Guide and the following manual pages: `pbs_server(8B)`, `qmgr(8B)`, and `qenable(8B)`

2.45 `qenable`

Allow jobs to be enqueued in a queue

2.45.1 Synopsis

qenable destination ...

qenable --version

2.45.2 Description

The `qenable` command directs that a destination queue should accept batch jobs.

The `qenable` command sends a Manage request to the batch server specified by queue. If the command is accepted, the destination will accept Queue Job requests which specified the queue.

In order to execute `qenable`, the user must have PBS Operator or Manager privilege.

2.45.3 Options

`--version`

The `qenable` command returns its PBS version information and exits. This option can only be used alone.

2.45.4 Operands

The `qenable` command accepts one or more destination operands. The operands are one of three forms:

queue

@server

queue@server

If *queue* is specified, the request is to enable that queue at the default server. If the *@server* form is given, the request is to enable all the queues at that server. If a full destination identifier, *queue@server*, is given, the request is to enable the named queue at the named server.

2.45.5 Standard Error

The `qenable` command will write a diagnostic message to standard error for each error occurrence.

2.45.6 Exit Status

Upon successful processing of all the operands presented to the `qenable` command, the exit status will be a value of zero.

If the `qenable` command fails to process any operand, the command exits with a value greater than zero.

2.45.7 See Also

The PBS Professional Administrator's Guide and the following manual pages:
`pbs_server(8B)`, `qdisable(8B)`, and `qmgr(8B)`

2.46 qhold

Holds PBS batch jobs

2.46.1 Synopsis

qhold [-h hold_list] job_identifier_list

qhold --version

2.46.2 Description

The **qhold** command requests that a server place one or more holds on a job. A job that has a hold is not eligible for execution. Supported holds: *USER*, *OTHER* (also known as *operator*), *SYSTEM*, and *bad password*.

A user may place a *USER* hold upon any job the user owns. An operator, who is a user with operator privilege, may place either a *USER* hold or an *OTHER* hold on any job. The batch administrator may place any hold on any job.

The **p** option can only be set by root or admin user via **qhold -h p**. The owning user can release with **qrls -h p** or query by **qselect -h p**.

If no **-h** option is given, the *USER* hold will be applied to the jobs described by the *job_identifier_list* operand list.

If the job identified by *job_identifier_list* is in the queued, held, or waiting states, then all that occurs is that the hold type is added to the job. The job is then placed into the held state if it resides in an execution queue.

If the job is running, then the result of the **qhold** command depends upon whether the job can be checkpointed. The job can be checkpointed if the OS supports checkpointing, or if the application being checkpointed supports checkpointing. See the PBS Professional Administrator's Guide. If the job can be checkpointed, the following happens:

- The job is checkpointed and its execution is interrupted.
- The resources assigned to the job are released.
- The job is placed in the held state in the execution queue.
- The job's **Hold_Types** attribute is set to *u* for *User Hold*.

If checkpoint / restart is not supported, **qhold** simply sets the job's **Hold_Types** attribute to *u*. The job continues to execute.

A job's dependency places a system hold on the job. When the dependency is satisfied, the system hold is removed. This system hold is the same as the one set by an administrator. If the administrator sets a system hold on a job with a dependency, then when the dependency is satisfied, the job becomes eligible for execution.

The `qhold` command can be used on job arrays, but not on subjobs or ranges of subjobs. If the job is in the process of provisioning, it cannot be held.

2.46.3 Options to `qhold`

`-h hold_list`

Defines the types of holds to be placed on the job.

The `hold_list` argument is a string consisting of one or more of the letters “*u*”, “*o*”, or “*s*” in any combination or the character “*n*” or “*p*”. The hold type associated with each letter is:

Table 2-8: Hold Types

Hold Type	Meaning
<i>u</i>	<i>USER</i>
<i>o</i>	<i>OTHER</i>
<i>s</i>	<i>SYSTEM</i>
<i>n</i>	<i>None</i>
<i>p</i>	<i>Bad password</i>

`--version`

The `qhold` command returns its PBS version information and exits. This option can only be used alone.

2.46.4 Operands

The `qhold` command accepts a *job_identifier_list* which is one or more space-separated job IDs in the form:

sequence_number[*.server_name*][*@server*]

Note that some shells require that you enclose a job array identifier in double quotes.

2.46.5 Standard Error

The `qhold` command will write a diagnostic message to standard error for each error occurrence.

2.46.6 Exit Status

Zero upon successful processing of all the operands.

Greater than zero if the `qhold` command fails to process any operand.

2.46.7 See Also

The PBS Professional User's Guide, the PBS Professional Administrator's Guide, `qrls(1B)`, `qalter(1B)`, `qsub(1B)`, `pbs_alterjob(3B)`, `pbs_holdjob(3B)`, `pbs_rlsjob(3B)`, `pbs_job_attributes(7B)`, `pbs_resources(7B)`

2.47 qmgr

Administrator's command interface for managing PBS

2.47.1 Synopsis

```
qmgr [-a] [-c directive] [-e] [-n] [-z] [server [server]...]
```

```
qmgr import hook <hook_name> <content-type> <content-encoding> {<input_file>|-}
```

```
qmgr export hook <hook_name> <content-type> <content-encoding> [<output_file>]
```

```
qmgr --version
```

2.47.2 Description

The PBS manager command, `qmgr`, provides a command-line interface to the PBS Server. The `qmgr` command is used to create or delete queues, vnodes, and hooks, to set or change vnode, queue, hook, server, or scheduler attributes, including resources, and to view information about hooks, queues, vnodes, the server, and the scheduler.

For information about attributes, see [Chapter 6, "Attributes", on page 309](#).

Attributes whose values are unset do not appear in the output of the `qmgr` command.

If `qmgr` is invoked without the `-c` option and standard output is connected to a terminal, `qmgr` writes a prompt to standard output and reads a directive from standard input. See [section 2.47.4.1, “Directive Syntax”, on page 153](#).

For a `qmgr` prompt, type:

```
qmgr <return>
```

For syntax and command information, type “`help`” at the `qmgr` prompt.

2.47.2.1 Required Privilege

The `qmgr` command requires different levels of privilege depending on the operation to be performed. All users can list or print attributes. PBS Operator or Manager privilege is required in order to set or change vnode, queue, hook, server, or scheduler attributes. PBS Manager privilege is required in order to create or delete queues, vnodes, and hooks.

Under UNIX/Linux, root privilege is required in order to operate on hooks or the `job_sort_formula` server attribute. Under Windows, this must be done from the installation account. For domained environments, the installation account must be a local account that is a member of the local Administrators group on the local computer. For standalone environments, the installation account must be a local account that is a member of the local Administrators group on the local computer.

Users without manager or operator privilege cannot view custom resources which were created to be invisible to users.

2.47.2.2 When To Run `qmgr` At Server Host

When operating on hooks or on the `job_sort_formula` server attribute, the `qmgr` command must be run at the server host.

2.47.3 Options to `qmgr`

The following table lists the options to the `qmgr` command:

Table 2-9: `qmgr` Options

Option	Action
<code>-a</code>	Abort <code>qmgr</code> on any syntax errors or any requests rejected by a server.

Table 2-9: qmgr Options

Option	Action
-c directive	Execute a single command and exit qmgr. The command must be enclosed in quote marks, e.g. <code>qmgr -c "print server"</code>
-e	Echo all commands to standard output.
-n	No commands are executed; syntax checking only is performed.
server [,server ...]	Makes the specified server(s) active. See section 2.47.6.1, “Making Objects Active”, on page 159 .
-z	No errors are written to standard error.
--version	The qmgr command returns its PBS version information and exits. This option can only be used alone.

2.47.4 Directives

A *qmgr directive* is a *command* together with the *object* to be operated on, the *attribute* belonging to the object that is to be changed, the *operator*, and the *value* the attribute will take.

A directive is terminated by a newline or a semicolon (“;”). Multiple directives may be entered on a single line. A directive may extend across lines by escaping the newline with a backslash (“\”).

2.47.4.1 Directive Syntax

A qmgr directive takes one of the following forms:

command <object type> [object names] [attribute OP value[,attribute OP value,...]]

import hook <hook_name> <content-type> <content-encoding> {<input_file>|-}

export hook <hook_name> <content-type> <content-encoding> [<output_file>]

The directive can be used from the command line or from within the qmgr command. To use a directive from the command line, enclose the command and its arguments in single or double quotes:

qmgr -c “command <object type> [object names] [attribute OP value[,attribute OP value,...]]”

For example:

```
qmgr -c 'print queue Q1'
```

To use a directive from within the `qmgr` command, first start `qmgr`:

```
qmgr <return>
```

At the `qmgr` prompt, enter the command and its arguments. For example:

```
Qmgr: print queue Q1
```

Each command is explained in the following subsections.

2.47.4.2 Comments

Comments begin with the “#” character and continue to the end of the line. Comments and blank lines are ignored by `qmgr`.

2.47.5 Arguments to Commands

2.47.5.1 Objects

The `qmgr` command can operate on servers, schedulers, queues, vnodes, and hooks. Each of these can be abbreviated in a directive. The following table lists the objects and their abbreviations:

Table 2-10: qmgr Objects

Object Name	Abbr	Object	Can be Created/ Deleted By:	Can be Modified By:
<i>server</i>	<i>s</i>	A Server	No one	Operator, Manager
<i>queue</i>	<i>q</i>	A queue	Operator, Manager	Operator, Manager
<i>node</i>	<i>n</i>	A vnode	Operator, Manager	Operator, Manager
<i>hook</i>	<i>h</i>	A hook	UNIX/Linux: root; Windows: installation account	UNIX/Linux: root; Windows: installation account
<i>sched</i>	<i>sc</i>	A scheduler	No one	Operator, Manager

2.47.5.2 Specifying Server

The `qmgr` command operates on objects (queues, vnodes, etc.) at the active server. There is always at least one active server; the default server is the active server unless other servers have been made active.

The default server is the server managing the host where the `qmgr` command runs, meaning it is the server specified in that host's `pbs.conf` file.

You can specify the server you want:

`@default`

Specifies the default server.

`@<server name>`

Specifies a named server.

`@active`

Specifies all active servers.

Server names have the following format:

`hostname[:port]`

where *hostname* is the fully-qualified domain name of the host on which the server is running and *port* is the port number to which to connect. If *port* is not specified, the default port number is used.

2.47.5.3 Object Names

In a `qmgr` directive, *object names* is a list of one or more names of specific objects. All objects in a list must be of the same type. The name list is in the form:

`<object name>[@<server>][,<object name>[@<server>] ...]`

where `<server>` is replaced in the directive with “*default*”, “*active*”, or “*<server name>*”. There must be no space between the name and the `@` sign.

Example 2-1: List queues `workq`, `slowq`, and `fastq` at the active server:

`qmgr: list queue workq,slowq,fastq`

Example 2-2: List queues `Queue1` at the default server, `Queue2` at `Server2`, and `Queue3` at the active server:

`qmgr: list queue Queue1@default,Queue2@Server2,Queue3@active`

Name lists must not contain white space between entries.

The administrator specifies the name of an object when creating the object.

Node attributes cannot be used as node names.

2.47.5.4 Specifying Objects

You can specify objects in the following ways:

<object type>

Acts on the active objects of the named type at the active server.

For example, to list all active vnodes, along with their attributes, at the active server:

```
list node
```

<object type> @*<server>* (note space before @ sign)

Acts on the active objects of the named type at the specified server.

For example, to list all active vnodes at the default server, along with their attributes:

```
list node @default
```

For example, to print out all queues at the default server, along with their attributes:

```
qmgr -c "print queue @default"
```

<object type> *<object name>*

Acts on the named object.

For example, to list Node1 and its attributes:

```
list node Node1
```

<object type> *<object name>*@*<server>*

Acts on the named object at the specified server.

For example, to list Node1 at the default server, along with the attributes of Node1:

```
list node Node1@default
```

2.47.5.5 Attributes

In a qmgr directive, *attribute* is the name of the attribute belonging to the object on which qmgr is to operate. You can set or modify the value of this attribute.

If the attribute is one which describes a set of resources, then the attribute is specified in the form:

attribute.<resource name>

For example, to set the amount of memory on a vnode:

```
Qmgr: set node Vnode1 resources_available.mem = 2mb
```

Any attribute value set via `qmgr` containing commas, whitespace or the hashmark must be enclosed in double quotes. For example:

```
Qmgr: set node Vnode1 comment="Node will be taken offline Friday at
1:00 for memory upgrade."
```

2.47.5.6 Operators

In a `qmgr` directive, *OP* is the operation to be performed with the attribute and its value. Operators are listed here:

Table 2-11: Operators

Operator	Effect
=	Sets the value of the attribute. If the attribute has an existing value, the current value is replaced with the new value.
+=	Increases the current value of the attribute by the amount in the new value. When used for a string array, adds the new value as another string after a comma.
-=	Decreases the current value of the attribute by the amount in the new value. When used for a string array, removes the first matching string.

Example 2-3: Set routing destination for queue Queue1 to be Dest1:

```
Qmgr: set queue route_destinations = Dest1
```

Example 2-4: Add new routing destination for queue Queue1:

```
Qmgr: set queue route_destinations += Dest2
```

Example 2-5: Remove new routing destination for queue Queue1:

```
Qmgr: set queue route_destinations -= Dest2
```

2.47.5.7 Attribute Values

In a `qmgr` directive, *value* is the value to assign to an attribute. An attribute's value must be in the correct format for the attribute's type. Each attribute's type is listed in [Chapter 6, "Attributes", on page 309](#). Each format is described in [Chapter 7, "Formats", on page 403](#).

If the value includes whitespace, commas or other special characters, such as the # character, the value string must be enclosed in single or double quotes.

Resource values can be any string made up of alphanumeric, comma (“,”), underscore (“_”), dash (“-”), colon (“:”), slash (“/”), backslash (“\”), space (“ ”), and equal sign (“=”) characters.

2.47.5.8 Windows Requirements

Under Windows, use double quotes when specifying arguments to `qmgr`. For example:

```
Qmgr: import hook hook1 application/x-python default "\Documents and
Settings\pbsuser1\hook1.py"
```

or

```
qmgr -c 'import hook hook1 application/x-python default "\Documents and
Settings\pbsuser1\hook1.py"'
```

2.47.6 Commands

The `qmgr` commands can be used in two ways. One is to start `qmgr`, then use directives at the `qmgr` prompt. For example, type:

```
qmgr <return>
```

The `qmgr` prompt appears:

```
Qmgr:
```

Now you can enter commands, using directives, for example:

```
Qmgr: print server
```

The other is to call `qmgr` with the `-c` option and a directive in quotes. For example, to enter the same “`print server`” directive:

```
qmgr -c "print server"
```

Commands can be abbreviated to their minimum unambiguous form. Commands apply to all objects unless explicitly limited. The following table lists the commands, briefly tells what they do, and gives a link to a full description:

Table 2-12: `qmgr` Commands

Command	Abbrev	Effect	Description
active	a	Specifies active objects	See section 2.47.6.1, “Making Objects Active” , on page 159
create	c	Creates object	See section 2.47.6.2, “Creating Objects” , on page 162

Table 2-12: qmgr Commands

Command	Abbrev	Effect	Description
delete	d	Deletes object	See section 2.47.6.3, “Deleting Objects” , on page 163
export	e	Exports hook	See section 2.47.6.4, “Exporting Hooks” , on page 163
help	h	Prints usage to std-out	See section 2.47.6.10, “Printing Usage Information” , on page 168
import	i	Imports hook	See section 2.47.6.5, “Importing Hooks” , on page 164
list	l	Lists object attributes and their values	See section 2.47.6.8, “Listing Object Attributes” , on page 166
print	p	Prints creation and configuration commands	See section 2.47.6.9, “Printing Creation and Configuration Commands” , on page 167
quit	q	Exits the qmgr command	
set	s	Sets value of attribute	See section 2.47.6.6, “Setting Attribute Values” , on page 165
unset	u	Unsets value of attribute	See section 2.47.6.7, “Unsetting Attribute Values” , on page 166

2.47.6.1 Making Objects Active

Making objects active is a way to set up a list of objects, all of the same type, on which you can then use a single command. Can be used on any object. For example, if you are going to set the same attribute to the same value on several vnodes, you can make all of the target vnodes active before using a single command to set the attribute value, instead of having to give the command once for each vnode.

When an object is active, it is acted upon when you specify its type but do not specify names. When you specify any object names in a directive, active objects are not operated on unless they are named in the directive.

You can specify a list of active objects for each type of object. You can have active objects of multiple types at the same time. The active objects of one type have no effect on whether objects of another type are active.

Objects are active only until the `qmgr` command is exited, so this feature can be used only at the `qmgr` prompt.

Each time you make any objects active, that list of objects replaces any active objects of the same kind. For example, if you have four queues, and you make Q1 and Q2 active, then later make Q3 and Q4 active, the result is that Q3 and Q4 are the only active queues.

You can make objects be active at different servers simultaneously. For example, you can set vnodes N1 and N2 at the default server, and vnodes N3 and N4 at server Server2 to be active at the same time.

To make all objects inactive, quit `qmgr`. When you quit `qmgr`, any object that was active is no longer active.

2.47.6.1.i Using the **active** Command

active <object type> [*<object name>* [, *<object name>* ...]]

Makes the named object(s) of the specified type active.

Example: To make queue Queue1 active:

Qmgr: active queue Queue1

Example: To make queues Queue1 and Queue2 at the active server be active, then enable them:

Qmgr: active queue Queue1,Queue2

Qmgr: set queue enabled=True

Example: To make queue Queue1 at the default server and queue Queue2 at Server2 be active:

Qmgr: active queue Queue1@default,Queue2@Server2

Example: To make vnodes N1, N2, N3, and N4 active, and then give them all the same value for their max_running attribute:

Qmgr: active node N1,N2,N3,N4

Qmgr: set node max_running = 2

active <object type> @<server> (note space before @ sign)

Makes all object(s) of the specified type at the specified server active.

Example: To make all queues at the default server active:

Qmgr: active queue @default

Example: To make all vnodes at server Server2 active:

Qmgr: active node @Server2

active <object type>

Queries which objects of the specified type are active. The qmgr command prints a list of names of active objects of the specified type to stdout.

2.47.6.2 Creating Objects

create <object type> <object name>[,<object name> ...] [[attribute = value] [,attribute = value] ...]

Creates one new object of the specified type for each name, and gives it the specified name. Can be used only with queues, vnodes, and hooks.

For example, to create a queue named Q1 at the active server:

Qmgr: create queue Q1

For example, to create a vnode named N1 and a vnode named N2:

Qmgr: create node N1,N2

For example, to create queue Queue1 at the default server and queue Queue2 at Server2:

Qmgr: create queue Queue1@default,Queue2@Server2

For example, to create vnodes named N1, N2, N3, and N4 at the active server, and to set their Mom attribute to *Host1* and their max_running attribute to 1:

Qmgr: create node N1,N2,N3,N4 Mom=Host1, max_running = 1

All objects of the same type at a server must have unique names. For example, each queue at server Server1 must have a unique name. Objects at one server can have the same name as objects at another server.

You can create multiple objects of the same type with a single command. You cannot create multiple types of objects in a single command.

2.47.6.3 Deleting Objects

delete <object type> <object name>[,<object name> ...]

Deletes the named object(s). Can be used only with queues, vnodes, and hooks. For example, to delete queue Q1 at the active server:

Qmgr: delete queue Q1

For example, to delete vnodes N1 and N2 at the active server:

Qmgr: delete node N1,N2

For example, to delete queue Queue1 at the default server and queue Queue2 at Server2:

Qmgr: delete queue Queue1@default,Queue2@Server2

delete <object type>

Deletes the active objects of the specified type. For example, to delete the active queues:

Qmgr: delete queue

delete <object type> @<server>

Deletes the active objects of the specified type at the specified server. For example, to delete the active queues at server Server2:

Qmgr: delete queue @Server2

You can delete multiple objects of the same type with a single command. You cannot delete multiple types of objects in a single command.

2.47.6.4 Exporting Hooks

Format for exporting a hook:

export hook <hook name> <content-type> <content-encoding> [<output_file>]

This dumps the script contents of hook <hook_name> into <output_file>, or `stdout` if <output_file> is not specified.

- The resulting <output_file> or `stdout` data is of <content-type> and <content-encoding>.
- The only <content-type> currently supported is “*application/x-python*”.
- The allowed values for <content-encoding> are “*default*” (7bit) and “*base64*”.
- <output_file> must be a path that can be created by `qmgr`.
- Any relative path <output_file> is relative to the directory where `qmgr` was executed.
- If <output_file> already exists it is overwritten. If PBS is unable to overwrite the file due

to ownership or permission problems, then an error message is displayed in `stderr`.

- If the `<output_file>` name contains spaces like the ones used in Windows file names, then `<output_file>` must be enclosed in quotes.

Example 2-6: Dumps hook1's script contents directly into a file "hello.py.out":

```
# qmgr -c 'export hook hook1 application/x-python default hello.py'
# cat hello.py
import pbs
pbs.event().job.comment="Hello, world"
```

Example 2-7: To dump the script contents of a hook 'hook1' into a file in “\My Hooks\hook1.py”:

```
Qmgr: export hook hook1 application/x-python default "\My
Hooks\hook1.py"
```

2.47.6.5 Importing Hooks

To import a hook, you import the contents of a hook script into the hook. You must specify a filename that is locally accessible to `qmgr` and the PBS Server.

Format for importing a hook:

```
import hook <hook name> <content-type> <content-encoding> {<input_file>|-}
```

This uses the contents of `<input_file>` or `stdin (-)` as the contents of hook `<hook_name>`.

- The `<input_file>` or `stdin (-)` data must have a format `<content-type>` and must be encoded with `<content-encoding>`.
- The only `<content-type>` currently supported is “`application/x-python`”.
- The allowed values for `<content-encoding>` are “`default`” (7bit) and “`base64`”.
- If the source of input is `stdin (-)` and `<content-encoding>` is “`default`”, then `qmgr` expects the input data to be terminated by EOF.
- If the source of input is `stdin (-)` and `<content-encoding>` is “`base64`”, then `qmgr` expects input data to be terminated by a blank line.
- `<input_file>` must be locally accessible to both `qmgr` and the requested batch server.
- A relative path `<input_file>` is relative to the directory where `qmgr` was executed.
- If a hook already has a content script, then that is overwritten by this import call.
- If `<input_file>` name contains spaces as are used in Windows filenames, then `<input`

file> must be quoted.

- There is no restriction on the size of the hook script.

Example 2-8: Given a Python script in ASCII text file "hello.py", this makes its contents be the script contents of hook1:

```
#cat hello.py
import pbs
pbs.event().job.comment="Hello, world"
# qmgr -c 'import hook hook1 application/x-python default hello.py'
```

Example 2-9: Given a base64-encoded file "hello.py.b64", qmgr unencodes the file's contents, and then makes this the script contents of hook1:

```
# cat hello.py.b64
cHJpbnQgImhlbGxvLCB3b3JsZC1K
# qmgr -c 'import hook hook1 application/x-python base64 hello.py.b64'
```

2.47.6.6 Setting Attribute Values

You can use the qmgr command to set attributes of any object.

set <object type> <object name>[,<object name> ...] attribute = value [,attribute = value ...]

Sets the value of the specified attribute(s) for the named object(s). Each specified attribute is set for each named object, so if you specify three attributes and two objects, both objects get all three attributes set.

set <object type> attribute = value

Sets the attribute value for all active objects when there are active objects of the type specified.

set <object type> @<server> attribute = value

Sets the attribute value for all active objects at the specified server when there are active objects of the type specified.

You can have spaces between attribute-value pairs.

If a value contains a space or a comma, the value must be enclosed in single or double quotes. For example, to set the value of vnode N1's comment to "Check later, replacing memory":

```
Qmgr: set node N1 comment = "Check later, replacing memory"
```

You can set attribute values for only one type of object in each command.

2.47.6.7 Unsetting Attribute Values

You can use the `qmgr` command to unset attributes of any object.

```
unset <object type> <object name>[, <object name> ...] attribute[, attribute...]
```

Unsets the value of the specified attributes of the named object(s).

```
unset <object type> attribute[, attribute...]
```

Unsets the value of specified attributes of active objects.

```
unset <object type> <object name> attribute[, attribute...]
```

Unsets the value of specified attributes of the named object.

```
unset <object type> @<server> attribute[, attribute...]
```

Unsets the value of specified attributes of active objects at the specified server.

You can have spaces between attribute names.

You can unset attribute values for only one type of object in each command.

2.47.6.8 Listing Object Attributes

You can use the `qmgr` command to list attributes of any object.

```
list <object type> <object name>[, <object name> ...]
```

Lists the attributes, with associated values, of the named object(s).

```
list <object type> <object name> <attribute name>[, <attribute name>]...
```

Lists values of the specified attributes of the named object.

```
list <object type>
```

Lists attributes, with associated values, of active objects of the specified type at the active server.

```
list <object type> @<server>
```

Lists all objects of the specified type at the specified server, with their attributes and the values associated with the attributes.

```
list server
```

Lists attributes of the active server. If no server other than the default server has been made active, lists attributes of the default server (it is the active server).

```
list server <server>
```

Lists attributes of the specified server.

list hook

Lists all hooks, along with their attributes.

list hook <hook name>

Lists attributes of the specified hook.

2.47.6.9 Printing Creation and Configuration Commands

print <object type> <object name>[, <object name> ...]

where *<object name>* follows the name rules in [section 2.47.5.3, “Object Names”, on page 155](#).

Prints out the commands required to do the following:

- Create the named object(s)
- Set object attributes to their current values

print <object type> <object name> [<attribute name>[, <attribute name>]...]

where *<object name>* follows the name rules in [section 2.47.5.3, “Object Names”, on page 155](#).

Prints out the commands required to do the following:

- Create the named object
- Set specified object attributes to their current values

print <object type>

Prints out the commands to create and configure the active objects of the named type.

print <object type> @<server>

Prints out the commands to create and configure all of the objects of the specified type at the specified server.

print server

Prints information for the active server; if there is no active server, prints information for the default server.

Prints out the commands required to do the following for the server and queues, but not hooks:

- Create each queue
- Set the attributes of each queue to their current values

- Set the attributes of the server to their current values

print hook

Prints out the commands to create and configure all hooks.

print hook <hook name>

Prints out the commands to create and configure the specified hook.

2.47.6.10 Printing Usage Information

The `qmgr` built-in help function is invoked using the `help` command. You can request usage information for any of the `qmgr` commands.

help <command>

Prints out usage information for the specified command.

For example, to print usage information for the `set` command:

qmgr

Qmgr: help set

Syntax: set object [name][,name...] attribute[.resource] OP value

2.47.7 Saving and Re-creating Configuration

To save and recreate a configuration, print the configuration information to a file, then read it back in later.

See [section 2.47.6.9, “Printing Creation and Configuration Commands”, on page 167](#).

2.47.7.1 Saving Queue Information

Before re-creating queue and server configuration, use this command to save the configuration/creation commands to a file:

Qmgr: print server > savedsettings

When re-creating the queue and server configuration, read the commands into `qmgr`:

qmgr < savedsettings

2.47.7.2 Saving Hook Information

To save creation and configuration information for all hooks:

qmgr -c "print hook" > hook.qmgr

To re-create all hooks:

```
# qmgr < hook.qmgr
```

2.47.8 Standard Input

The `qmgr` command reads standard input for directives until end-of-file is reached, or the `exit` or `quit` directive is read.

2.47.9 Standard Output

If standard output is connected to a terminal, a command prompt is written to standard output when `qmgr` is ready to read a directive.

If the `-e` option is specified, `qmgr` will echo the directives read from standard input to standard output.

2.47.10 Standard Error

If the `-z` option is not specified, the `qmgr` command writes a diagnostic message to standard error for each error occurrence.

2.47.11 Exit Status

Upon successful processing of all the operands presented to the `qmgr` command, the exit status is zero.

If the `qmgr` command fails to process any operand, the command exits with a value greater than zero.

2.47.12 Caveats

2.47.12.1 Setting Vnode Attributes

Most of a vnode's attributes may be set using `qmgr`. However, some **must** be set on the individual execution host in local vnode definition files, NOT by using `qmgr`. See [section 3.5.2, "Choosing Configuration Method" on page 48 in the PBS Professional Administrator's Guide](#).

2.47.13 Examples

The following are examples of `qmgr` directives:

List serverA's scheduler's attributes

```
list sched @serverA
```

List attributes for default server's scheduler

```
l sched @default
```

List PBS version for default server's scheduler

```
l sched @default pbs_version
```

Set software resource on mynode

```
set node mynode resources_available.software = "myapp=/tmp/foo"
```

Create queue

```
create queue fast priority=10,queue_type=e,enabled = true,max_running=0
```

Increase limit on queue

```
set queue fast max_running +=2
```

Create queue, set resources

```
create queue little  
set queue little resources_max.mem=8mw,resources_max.cput=10
```

Unset limit on queue

```
unset queue fast max_running
```

Set node offline

```
set node state = "offline"
```

Define active list

```
active server s1,s2,s3
```

List a queue

```
list queue @server1
```

Set limit on queue

```
set queue max_running = 10
```

To create a provisioning hook called Provision_Hook, and import the ASCII hook script called "master_provision.py" located in /root/data/:

```
Qmgr: create hook Provision_Hook  
Qmgr: import hook Provision_Hook application/x-python default /root/  
data/master_provision.py
```

2.47.14 See Also

The PBS Professional Administrator's Guide, the PBS Professional Installation and Upgrade Guide, `pbs_queue_attributes(7B)`, `pbs_server_attributes(7B)`, `pbs_node_attributes(7B)`, `pbs_hook_attributes(7B)`, `pbs_sched_attributes(7B)`

2.48 qmove

Moves PBS batch job

2.48.1 Synopsis

qmove destination job_identifier ...

qmove --version

2.48.2 Description

To move a job is to remove the job from the queue in which it resides and place the job in another queue.

The `qmove` command can be used on job arrays, but not on subjobs or ranges of subjobs.

Note that job arrays can only be moved from one server to another if they are in the 'Q', 'H', or 'W' states, and only if there are no running subjobs. The state of the job array is preserved, and the job array will run to completion on the new server.

A job in the *Running*, *Transiting*, or *Exiting* state cannot be moved.

A job in the process of provisioning cannot be moved.

The behavior of the `qmove` command may be affected by any site hooks. Site hooks can modify the job's attributes, change its routing, etc.

2.48.3 Effect of Privilege on Behavior

An unprivileged user can use the `qmove` command to move a job only when the move would not violate queue restrictions. A privileged user (root, Manager, Operator) can use the `qmove` command to move a job under some circumstances where an unprivileged user cannot. The restrictions that apply only to unprivileged users are listed here:

- The queue must be enabled
- Moving the job into the queue must not exceed the queue's limits for jobs or resources
- If the job is an array job, the size of the job array must not exceed the queue's `max_array_size`
- If the queue is accepting jobs only from routing queues, unprivileged users cannot move jobs into it using the `qmove` command

2.48.4 Options

`--version`

The `qmove` command returns its PBS version information and exits. This option can only be used alone.

2.48.5 Operands

The first operand is the new destination for the jobs. It will be accepted in the syntax:

queue

@server

queue@server

See [Chapter 7, "Formats", on page 403](#) for destination identifier information.

If the destination operand describes only a queue, then `qmove` will move jobs into the queue of the specified name at the job's current server.

If the destination operand describes only a batch server, then `qmove` will move jobs into the default queue at that batch server.

If the destination operand describes both a queue and a batch server, then `qmove` will move the jobs into the specified queue at the specified server.

All following operands are *job_identifiers* which specify the jobs to be moved to the new destination. The `qmove` command accepts one or more *job_identifier* operands of the form:

sequence_number[*.server_name*][*@server*]

Note that some shells require that you enclose a job array identifier in double quotes.

2.48.6 Standard Error

The `qmove` command will write a diagnostic messages to standard error for each error occurrence.

2.48.7 Exit Status

Upon successful processing of all the operands presented to the `qmove` command, the exit status will be a value of zero.

If the `qmove` command fails to process any operand, the command exits with a value greater than zero.

2.48.8 See Also

The PBS Professional User's Guide, the PBS Professional Administrator's Guide, `qsub(1B)`, `pbs_movejob(3B)`

2.49 qmsg

Sends message to PBS batch jobs

2.49.1 Synopsis

qmsg [-E] [-O] message_string job_identifier ...

qmsg --version

2.49.2 Description

To send a message to a job is to write a message string into one or more output files of the job. Typically this is done to leave an informative message in the output of the job.

The `qmsg` command writes messages into the files of jobs by sending a Message Job batch request to the batch server that owns the job. The `qmsg` command does not directly write the message into the files of the job.

The `qmsg` command cannot be used on job arrays, subjobs or ranges of subjobs.

2.49.3 Options

- E Specifies that the message is written to the standard error of each job.
- O Specifies that the message is written to the standard output of each job.
- version The `qmsg` command returns its PBS version information and exits. This option can only be used alone.

If no option is specified, the message will be written to the standard error of the job.

2.49.4 Operands

The first operand, *message_string*, is the message to be written. If the string contains blanks, the string must be quoted. If the final character of the string is not a newline, a newline character will be added when written to the job's file.

All following operands are *job_identifiers* which specify the jobs to receive the message string. The `qmsg` command accepts one or more *job_identifier* operands of the form:

sequence_number[*.server_name*][*@server*]

2.49.5 Standard Error

The `qmsg` command will write a diagnostic message to standard error for each error occurrence.

2.49.6 Exit Status

Upon successful processing of all the operands presented to the `qmsg` command, the exit status will be a value of zero.

If the `qmsg` command fails to process any operand, the command exits with a value greater than zero.

2.49.7 See Also

The PBS Professional User's Guide, the PBS Professional Administrator's Guide, `qsub(1B)`, `pbs_msgjob(3B)`

2.50 qorder

Exchanges order of two PBS batch jobs.

2.50.1 Synopsis

qorder job_identifier job_identifier

qorder --version

2.50.2 Description

Allows the exchange of two jobs' positions in the queue or queues in which the jobs reside.

No attribute of the job, e.g. `priority`, is changed. The impact of interchanging the order within or between queues is dependent on local job scheduling policy; contact your systems administrator.

2.50.3 Restrictions

- A job in the running state cannot be reordered.
- The `qorder` command can be used on job arrays, but not on subjobs or ranges of subjobs.
- The two jobs must be located at the same server.

2.50.4 Effect of Privilege on Behavior

For an unprivileged user to reorder jobs, both jobs must be owned by the user. A privileged user (Manager, Operator) can reorder any jobs.

2.50.5 Options

`--version`

The `qorder` command returns its PBS version information and exits. This option can only be used alone.

2.50.6 Operands

Both operands are *job_identifiers* which specify the jobs to be exchanged. The `qorder` command accepts two *job_identifier* operands of the form:

sequence_number[*.server_name*][*@server*]

The server specification for the two jobs must agree as to the current location of the two job ids.

Note that some shells require that you enclose a job array identifier in double quotes.

2.50.7 Standard Error

The `qorder` command will write diagnostic messages to standard error for each error occurrence.

2.50.8 Exit Status

Upon successful processing of all the operands presented to the `qorder` command, the exit status will be a value of zero.

If the `qorder` command fails to process any operand, the command exits with a value greater than zero.

2.50.9 See Also

The PBS Professional User's Guide, the PBS Professional Administrator's Guide, `qsub(1B)`, `qmove(1B)`, `pbs_orderjob(3B)`, `pbs_movejob(3B)`

2.51 qrerun

Reruns a PBS batch job

2.51.1 Synopsis

```
qrerun [-W force] job_identifier [job_identifier ...]
```

```
qrerun --version
```

2.51.2 Description

The `qrerun` command reruns the specified jobs if possible. PBS Manager or Operator privilege is required to use this command.

To rerun a job is to kill it and requeue it in the execution queue from which it was run.

If a job is marked as not rerunnable then `qrerun` will fail. See the `-r` option on the `qsub` and `qalter` commands.

The `qrerun` command can be used on job arrays, subjobs, and ranges of subjobs. It cannot rerun a subjob which is not running.

2.51.3 Options

`-W force`

The job is to be requeued even if the node on which the job is executing is unreachable, or if the job's substate is *provisioning*.

`--version`

The `qrerun` command returns its PBS version information and exits. This option can only be used alone.

2.51.4 Operands

The `qrerun` command accepts one or more `job_identifier` operands of the form:

```
sequence_number[.server_name][@server]
```

Note that some shells require that you enclose a job array identifier in double quotes.

2.51.5 Standard Error

The `qrerun` command will write a diagnostic message to standard error for each error occurrence.

2.51.6 Exit Status

Zero upon successful processing of all operands.

Greater than zero upon failure to process any operand.

2.51.7 See Also

The PBS Professional User's Guide, the PBS Professional Administrator's Guide, `qsub(1B)`, `qalter(1B)`, `pbs_alterjob(3B)`, `pbs_rerunjob(3B)`

2.52 `qrls`

Releases hold on PBS batch jobs

2.52.1 Synopsis

`qrls [-h hold_list] job_identifier ...`

`qrls --version`

2.52.2 Description

The `qrls` command removes or releases holds which exist on batch jobs.

A job may have one or more types of holds which make the job ineligible for execution. The types of holds are *USER*, *OTHER*, *SYSTEM*, and *bad password*. The different types of holds may require that the user issuing the `qrls` command have special privilege. Typically, the owner of the job will be able to remove a *USER* hold, but not an *OTHER* or *SYSTEM* hold. An Attempt to release a hold for which the user does not have the correct privilege is an error and no holds will be released for that job.

If no `-h` option is specified, the *USER* hold will be released.

Only root or admin can set a bad password hold via `qhold -h p`. The owner of the job can `qrls -h p` a hold set with `qhold -h p`.

If the job has no `execution_time` pending, the job will change to the *queued* state. If an `execution_time` is still pending, the job will change to the *waiting* state.

2.52.3 Options

-h hold_list

Defines the types of hold to be released from the jobs. The hold_list option argument is a string consisting of one or more of the letters *u* , *o* , or *s* in any combination, or one or more of the letters *n* or *p*. The hold type associated with each letter is:

Table 2-13: Hold Types

Hold Type	Meaning
<i>u</i>	<i>USER</i>
<i>o</i>	<i>OTHER</i>
<i>s</i>	<i>SYSTEM</i>
<i>n</i>	<i>None</i>
<i>p</i>	<i>Bad password</i>

--version

The `qrls` command returns its PBS version information and exits. This option can only be used alone.

2.52.4 Operands

The `qrls` command accepts one or more job_identifier operands of the form:

sequence_number[*.server_name*][*@server*]

Note that some shells require that you enclose a job array identifier in double quotes.

2.52.5 Standard Error

The `qrls` command will write a diagnostic message to standard error for each error occurrence.

2.52.6 Exit Status

Upon successful processing of all the operands presented to the `qrls` command, the exit status will be a value of zero.

If the `qrls` command fails to process any operand, the command exits with a value greater than zero.

2.52.7 See Also

The PBS Professional User's Guide, the PBS Professional Administrator's Guide, `qsub(1B)`, `qalter(1B)`, `qhold(1B)`, `pbs_alterjob(3B)`, `pbs_holdjob(3B)`, and `pbs_rlsjob(3B)`

2.53 `qrun`

Runs a PBS batch job now

2.53.1 Synopsis

```
qrun [-a] [-H vnode-specification] job_identifier_list  
qrun --version
```

2.53.2 Description

The `qrun` command is used to force a job to run, regardless of scheduling position or resource requirements.

In order to execute `qrun`, the user must have PBS Operator or Manager privilege, and the job must be in the *Queued* state and reside in an execution queue.

The `qrun` command can be used on a subjob or a range of subjobs, but not on a job array. When it is used on a range of subjobs, the non-running subjobs in that range are run.

The `qrun` command cannot be used on a job that is in the process of provisioning.

NOTE: If you use a `-H vnode_specification` option to run a job, but specify insufficient vnodes or resources, the job may not run correctly. Avoid using this option unless you are sure.

When preemption is enabled, the scheduler preempts other jobs in order to run this job. Running a job via `qrun` gives the job higher preemption priority than any other class of job.

2.53.3 Options to `qrun`

`-a`

The `qrun` command exits before the job actually starts execution.

(no `-H` option)

A request is made of the Scheduler to schedule this job. The job is run immediately regardless of scheduling policy as long as the following are true:

- The queue in which the job resides is an execution queue.
- Either the resources required by the job are available, or preemption is enabled and the required resources can be made available by preempting jobs that are running.

The `qrun` command alone overrides the following:

- Limits on resource usage by users, groups, and projects
- Limits on the number of jobs that can be run at a vnode
- Boundaries between primetime and non-primetime, specified in `backfill_prime`
- Whether the job is in a primetime queue: you can run a job in a prime-time queue even when it's not primetime, or vice versa. Primetime boundaries are not honored.
- Dedicated time: you can run a job in a dedicated time queue, even if it's not in a dedicated time queue, and vice versa. However, dedicated time boundaries are still honored.

The `qrun` command alone does not override the following:

- Server and queue resource usage limits

(with `-H` option)

With the `-H` option, all scheduling policies are bypassed and the job is run directly. The job will be run immediately on the named vnodes, regardless of current usage on those vnodes with the exception of vnode state. The job will not be run and the `qrun` request will be rejected if any named vnode is down, offline, already allocated exclusively or would need to be allocated exclusively and another job is already running on the vnode.

If the `qrun -H` command is used on a job that requests an AOE, and that AOE is not instantiated on those vnodes, the vnodes are provisioned with the AOE.

If the job requests an AOE, and that AOE is not available on the specified vnodes, the job is held.

-H *vnode_specification*, without resources

The *vnode_specification* without resources has this format:

(vchunk)[+(vchunk) ...]

where *vchunk* has the format

vnode[+vnode ..]

Example:

```
-H (VnodeA+VnodeB)+(VnodeC)
```

PBS will apply one requested chunk from the job's selection directive in round-robin fashion to each *vchunk* in the list. Each *vchunk* must be sufficient to run the job's corresponding chunk, otherwise the job may not execute correctly.

-H *vnode_specification*, with resources

The *vnode_specification* with resources has this format:

(vchunk)[+(vchunk) ...]

where *vchunk* has the format

vnode:vnode_resources[+vnode:vnode_resources ...]

and where *vnode_resources* has the format

resource=value[:resource=value ...]

Example:

```
-H (VnodeA:mem=100kb:ncpus=1)
  +(VnodeB:mem=100kb:ncpus=2+VnodeC:mem=100kb)
```

PBS creates a new selection directive from the *vnode_specification*, using it instead of the original specification from the user. Any single resource specification will result in the job's original selection directive being ignored. Each *vchunk* must be sufficient to run the job's corresponding chunk, otherwise the job may not execute correctly.

If the job being run requests `-l place=exclhost`, take extra care to satisfy the `exclhost` request. Make sure that if any vnodes are from a multi-vnoded host, all vnodes from that host are allocated. Otherwise those vnodes can be allocated to other jobs.

--version

The `qrun` command returns its PBS version information and exits. This option can only be used alone.

2.53.4 Operands

The `qrun` command accepts a *job_identifier_list* containing one or more *job_identifiers* of the form:

```
sequence_number[.server_name][@server]
```

Note that some shells require that you enclose a job array identifier in double quotes.

2.53.5 Standard Error

The `qrun` command will write a diagnostic message to standard error for each error occurrence.

2.53.6 Exit Status

Zero, on success.

Greater than zero, if the `qrun` command fails to process any operand.

2.53.7 See Also

The PBS Professional Administrator's Guide, `qsub` (1B), `qmgr` (8B), `pbs_runjob` (3B)

2.54 qselect

Selects PBS batch jobs

2.54.1 Synopsis

```
qselect [-a [[op]date_time] [-A account_string] [-c [[op]interval] [-h hold_list] [-H] [-J] [-l
resource_list] [-N name] [-p [[op] priority] [-P project] [-q destination] [-r rerun] [-s
states] [-t suboption [[comparison] specified-time] [-T] [-u user_list] [-x]
```

```
qselect --version
```

2.54.2 Description

The `qselect` command lists those jobs that meet the specified selection criteria. Jobs are selected from a single server.

Each option acts as a filter restricting which jobs are listed. With no options, the `qselect` command will list all jobs at the server which the user is authorized to list (query status of).

When selecting jobs according to their requested resources, this command can be used only on resources in the `Resource_List` job attribute, or on the entire selection directive.

Jobs that are finished or moved are listed only when the `-x` or `-H` options are used. Otherwise, job selection is limited to queued and running jobs.

2.54.3 Relations

When an option is specified with a optional `op` component to the option argument, then `op` specifies a relation between the value of a certain job attribute and the value component of the option argument. If an `op` is allowable on an option, then the description of the option letter will indicate the `op` is allowable. The only acceptable strings for the `op` component, and the relation the string indicates, are shown in the following table of relations:

Table 2-14: Relations

Relation	Meaning
<code>.eq.</code>	the value represented by the attribute of the job is equal to the value represented by the option argument.
<code>.ne.</code>	the value represented by the attribute of the job is not equal to the value represented by the option argument.
<code>.ge.</code>	the value represented by the attribute of the job is greater than or equal to the value represented by the option argument.
<code>.gt.</code>	the value represented by the attribute of the job is greater than the value represented by the option argument.
<code>.le.</code>	the value represented by the attribute of the job is less than or equal to the value represented by the option argument.
<code>.lt.</code>	the value represented by the attribute of the job is less than the value represented by the option argument.

2.54.4 Options to **qselect**

-a [op]date_time

Deprecated. Restricts selection to a specific execution time, or a range of execution times.

The **qselect** command selects only jobs for which the value of the **Execution_Time** attribute is related to the **date_time** argument by the optional **op** operator.

The **date_time** argument has the format:

[[CC]YY]MMDDhhmm[.SS]

where the **MM** is the two digits for the month, **DD** is the day of the month, **hh** is the hour, **mm** is the minute, and the optional **SS** is the seconds. **CC** is the century and **YY** the year.

If **op** is not specified, jobs will be selected for which the **Execution_Time** and **date_time** values are equal. If **op** is specified, jobs will be selected according to the definitions given in Relations above.

-A account_string

Restricts selection to jobs whose **Account_Name** attribute matches the specified **account_string**.

-c [op]interval

Restricts selection to jobs whose **Checkpoint** interval attribute matches the specified relationship.

The values of the **Checkpoint** attribute are defined to have the following ordered relationship:

n > s > c=minutes > c > u

If the optional **op** is not specified, jobs will be selected whose **Checkpoint** attribute is equal to the interval argument. If **op** is specified, jobs will be selected according to the rules in Relations above.

For an interval value of “*u*”, only “*.eq.*” and “*.ne.*” are valid.

-h hold_list

Restricts the selection of jobs to those with a specific set of hold types. Only those jobs will be selected whose **Hold_Types** attribute exactly match the value of the **hold_list** argument.

The **hold_list** argument is a string consisting of the single letter **n**, or one or more of the letters **u**, **o**, **p**, or **s** in any combination. If letters are duplicated,

they are treated as if they occurred once. The letters represent the hold types:

Table 2-15: Hold Types

Hold Type	Meaning
<i>n</i>	none
<i>u</i>	user
<i>o</i>	other
<i>p</i>	bad password
<i>s</i>	system

-H

Restricts selection to finished and moved jobs.

-J

Limits the selection to job arrays only.

-l resource_list

Restricts selection of jobs to those with specified resource amounts. Users without operator or manager privilege cannot specify custom resources which were created to be invisible to users.

The resource_list is in the following format:

resource_name op value[,resource_name op val,...]

The relation operator **op** must be present.

For job-wide resources, all operators are useful. However, resource specifications for chunks using the select statement, or placement using the place statement are stored as strings. Therefore the only useful operators for these are *.eq.* and *.ne.*

The definitions given in Relations above are used when comparing the values of resources.

-N name

Restricts selection of jobs to those with a specific name.

-p [op]priority

Restricts selection of jobs to those with a priority that matches the specified relationship. If op is not specified, jobs are selected for which the job **Priority** attribute is equal to the **priority**

If the **op** is specified, the relationship is defined in Relations above.

-P project

Restrict selection of jobs to those matching the specified project.

Format: String.

Project name can contain any characters except for the following: Slash ("/"), left bracket ("["), right bracket ("]"), double quote ("\""), semicolon (";"), colon (":"), vertical bar ("|"), left angle bracket ("<"), right angle bracket (">"), plus ("+"), comma (","), question mark ("?"), and asterisk ("*").

-q destination

Restricts selection to those jobs residing at the specified destination.

The destination may be of one of the following three forms:

queue

@server

queue@server

If the **-q** option is not specified, jobs will be selected from the default server.

If the destination describes only a queue, only jobs in that queue on the default batch server will be selected.

If the destination describes only a server, then jobs in all queues on that server will be selected.

If the destination describes both a queue and a server, then only jobs in the named queue on the named server will be selected.

-r rerun

Restricts selection of jobs to those with the specified **Rerunnable** attribute.

The option argument must be a single character. The following two characters are supported by PBS: *y* and *n*.

-s states

Restricts job selection to those in the specified states.

The states argument is a character string which consists of any combination of the characters: *B*, *E*, *F*, *H*, *M*, *Q*, *R*, *S*, *T*, *U*, *W*, and *X*. (A repeated character will be accepted, but no additional meaning is assigned to it.)

Table 2-16: Job States

State	Meaning
<i>B</i>	Job array has started execution.
<i>E</i>	The <i>Exiting</i> state.
<i>F</i>	The <i>Finished</i> state.
<i>H</i>	The <i>Held</i> state.
<i>M</i>	The <i>Moved</i> state.
<i>Q</i>	The <i>Queued</i> state.
<i>R</i>	The <i>Running</i> state.
<i>S</i>	The <i>Suspended</i> state.
<i>T</i>	The <i>Transiting</i> state.
<i>U</i>	Job suspended due to workstation user activity.
<i>W</i>	The <i>Waiting</i> state.
<i>X</i>	The <i>eXited</i> state. Subjobs only.

Jobs will be selected which are in any of the specified states. Since array jobs are never in states *R*, *S*, *T*, or *U*, if those states are specified, no array job will be selected. Subjobs of the array in those states may be selected if -T is specified.

-t suboption [comparison] specified-time

Allows jobs to be selected according to their time attributes. The suboption is one of:

Table 2-17: Suboptions to the -t Option

Sub option	Time Attribute Selected	Meaning
a	Execution_Time	Time the job began execution
c	ctime	Job creation time, seconds since epoch
e	etime	Time the job became eligible to run
g	eligible_time	Amount of eligible time job accrued waiting to run
m	mtime	Modification time
q	qtime	Job queued time
s	stime	Job start time
t	estimated.start_time	Job's estimated start time

The comparison is one of the relations listed above in Relations.

The specified-time is in datetime format. See [Chapter 7, "Formats", on page 403](#).

A time period can be bracketed by using the -t option twice. For example, to select jobs using stime between noon and 3 p.m.:

```
qselect -ts.gt.09251200 -ts.lt.09251500
```

-T

Limits selection to job and subjob identifiers.

-u user_list

Restricts selection to jobs owned by the specified user names.

This provides a means of limiting the selection to jobs owned by one or more users.

The syntax of the user_list is:

```
user_name[@host][,user_name[@host],...]
```

Host names may be wild carded on the left end, e.g. “*.nasa.gov”.

User_name without a “@host” is equivalent to “user_name@*”, that is at any host. Jobs will be selected which are owned by the listed users at the corresponding hosts.

-x

Allows selection of finished and moved jobs in addition to queued and running jobs.

-X

Allows selection of completed or deleted subjobs (subjobs in *X* state).

--version

The `qselect` command returns its PBS version information and exits. This option can only be used alone.

2.54.5 Standard Output

The list of job identifiers of selected jobs is written to standard output. Each job identifier is separated by white space. Each job identifier is of the form:

```
sequence_number.server_name@server
```

Where *sequence_number.server* is the identifier assigned at submission time; see `qsub`. *@server* identifies the server which currently owns the job.

2.54.6 Standard Error

The `qselect` command will write a diagnostic message to standard error for each error occurrence.

2.54.7 Exit Status

Upon successful processing of all options presented to the `qselect` command, the exit status will be a value of zero.

If the `qselect` command fails to process any option, the command exits with a value greater than zero.

2.54.8 See Also

The PBS Professional User's Guide, the PBS Professional Administrator's Guide, `qalter(1B)`, `qdel(1B)`, `qhold(1B)`, `qmove(1B)`, `qrls(1B)`, `qstat(1B)`, `qsub(1B)`, `pbs_job_attributes(7B)`, `pbs_resources(7B)`

2.55 **qsig**

Send signal to PBS batch job

2.55.1 Synopsis

qsig [-s signal] job_identifier ...

qsig --version

2.55.2 Description

The `qsig` command requests that a signal be sent to the specified executing batch jobs. The signal is sent to the session leader of the job.

If the `-s` option is not specified, `'SIGTERM'` is sent.

- The request to signal a batch job is rejected if:
- The user is not authorized to signal the job.
- The job is not in the running state.
- The requested signal is not supported by the system upon which the job is executing.
- The job is in the process of provisioning

The `qsig` command sends a Signal Job batch request to the server which owns the job.

The `qsig` command can be used for job arrays, ranges of subjobs, and subjobs. If it is used on a range of subjobs, the subjobs in the range which are running will be signaled.

2.55.3 Options

-s signal

Declares which signal is sent to the job.

The signal argument is either a signal name, e.g. `SIGKILL`, the signal name without the SIG prefix, e.g. `KILL`, or an unsigned signal number, e.g. `9`. The signal name `SIGNULL` is allowed; the server will send the signal 0 to the job which will have no effect. Not all signal names will be recognized by `qsig` signal name; try issuing the signal number instead.

Two special signal names, “*suspend*” and “*resume*”, [note, all lower case], are used to suspend and resume jobs. When suspended, a job continues to occupy system resources but is not executing and is not charged for wall-time. Manager or operator privilege is required to suspend or resume a job.

If `qsig -s resume` is used on a job that was suspended using `qsig -s suspend`, the job will be resumed when there are sufficient resources.

--version

The `qsig` command returns its PBS version information and exits. This option can only be used alone.

2.55.4 Operands

The `qsig` command accepts one or more *job_identifier* operands. For a job, this has the form:

```
sequence_number[.server_name][@server]
```

and for a job array, it is:

```
sequence_number[][.server_name][@server]
```

Note that some shells require that you enclose a job array identifier in double quotes.

2.55.5 Standard Error

The `qsig` command will write a diagnostic messages to standard error for each error occurrence.

2.55.6 Exit Status

Upon successful processing of all the operands presented to the `qsig` command, the exit status will be a value of zero.

If the `qsig` command fails to process any operand, the command exits with a value greater than zero.

2.55.7 See Also

The PBS Professional User's Guide, the PBS Professional Administrator's Guide, `qsub(1B)`, `pbs_sigjob(3B)`, `pbs_resources(7B)`

2.56 qstart

Allows PBS jobs to be run from a queue

2.56.1 Synopsis

qstart destination ...

qstart --version

2.56.2 Description

The `qstart` command directs that a destination queue should process batch jobs. If the destination is an execution queue, the scheduler will begin to schedule jobs that reside in the queue for execution. If the destination is a routing queue, the server will begin to route jobs from that queue.

In order to execute `qstart`, the user must have PBS Operator or Manager privilege.

2.56.3 Options

`--version`

The `qstart` command returns its PBS version information and exits. This option can only be used alone.

2.56.4 Operands

The `qstart` command accepts one or more destination operands. The operands are one of three forms:

queue

@server

queue@server

If *queue* is specified, the request is to start that queue at the default server. If the *@server* form is given, the request is to start all queues at that server. If a full destination identifier, *queue@server*, is given, the request is to start the named queue at the named server.

2.56.5 Standard Error

The `qstart` command will write a diagnostic message to standard error for each error occurrence.

2.56.6 Exit Status

Upon successful processing of all the operands presented to the `qstart` command, the exit status will be a value of zero.

If the `qstart` command fails to process any operand, the command exits with a value greater than zero.

2.56.7 See Also

The PBS Professional Administrator's Guide and the following manual pages: `pbs_server(8B)`, `qstop(8B)`, and `qmgr(8B)`

2.57 qstat

Displays status of PBS batch jobs, queues, or servers

2.57.1 Synopsis

2.57.1.1 Displaying Job Status

Default format:

```
qstat [-p] [-J] [-t] [-x] [ [job_identifier | destination] ...]
```

Long format:

```
qstat -f [-p] [-J] [-t] [-x] [ [job_identifier | destination] ...]
```

Alternate format:

```
qstat [-a [-w] | -H | -i | -r ] [-G | -M] [-J] [-n [-l][-w]] [-s [-l][-w]] [-t] [-u user_list] [
    [job_identifier | destination] ...]
```

2.57.1.2 Displaying Queue Status

Default format:

```
qstat -Q [destination ...]
```

Long format:

```
qstat -Q -f [destination ...]
```

Alternate format:

```
qstat -q [-G | -M] [destination ...]
```

2.57.1.3 Displaying Server Status

Default format:

```
qstat -B [server_name ...]
```

Long format:

```
qstat -B -f [server_name ...]
```

2.57.1.4 Displaying Version Information

```
qstat --version
```

2.57.2 Description

The `qstat` command is used to display the status of jobs, queues, and batch servers. The status information is written to standard output.

Status information can be displayed in a default format, an alternate format, or a long format, depending upon the options given. Default and alternate formats display all status information for a job, queue or server on one line, in columns. Long formats display status information one attribute to a line.

Status information for finished and moved jobs can be displayed using the `-x` and `-H` options.

When displaying job status information, the `qstat` command will display status information about all *job_identifiers* and destinations specified.

If your job has been moved to another server through peer scheduling, give the job ID as an argument to `qstat`. If you only give the `qstat` command, your job will not appear to exist. For example, your job 123.ServerA is moved to ServerB. In this case, use

```
qstat 123
```

or

```
qstat 123.ServerA
```

To list all jobs at ServerB, you can use:

```
qstat @ServerB
```

If your default server is ServerB, and your job started at ServerA but was moved to ServerB, to see the job, you must use:

```
qstat 123@ServerA
```

Users without manager or operator privilege cannot view resources or attributes that are invisible to unprivileged users.

2.57.3 Displaying Job Status

2.57.3.1 Job Status in Default Format

The `qstat` command will display job status in default format when the options given are among `-p`, `-J`, `-t` or `-x`, regardless of operands. Jobs are displayed one to a line, with these column headers:

```
Job id   Name           User           Time Use S Queue
-----
```

Description of columns:

Table 2-18: Description of Default Job Status Columns

Column	Description
Job id	The job_identifier assigned by PB
Name	Job name assigned by submitter.
User	Username of job owner.

Table 2-18: Description of Default Job Status Columns

Column	Description	
Time Use	<p>The CPU time or walltime used by the job, depending on which is specified or inherited. Before the application has actually started running, for example during stage-in, this field is "0". At the point where the application starts accumulating <code>cput</code> or <code>walltime</code>, this field changes to "00:00:00". After that, every time the MOM polls for resource usage, the field is updated.</p> <p>The MOM on the each execution host polls for the usage of all processes belonging to the job on her host. Usage is summed. The polling interval is short when a job first starts running and lengthens to a maximum 2 minutes. See section 3.6.1, "Configuring MOM's Polling Cycle" on page 53 in the PBS Professional Administrator's Guide.</p>	
S	The job's state:	
	<i>B</i>	Array job has at least one subjob running.
	<i>E</i>	Job is exiting after having run.
	<i>F</i>	Job is finished.
	<i>H</i>	Job is held.
	<i>M</i>	Job was moved to another server.
	<i>Q</i>	Job is queued.
	<i>R</i>	Job is running.
	<i>S</i>	Job is suspended.
	<i>T</i>	Job is being moved to new location.
	<i>U</i>	Cycle-harvesting job is suspended due to keyboard activity.
	<i>W</i>	Job is waiting for its submitter-assigned start time to be reached.
	<i>X</i>	Subjob has completed execution or has been deleted.
Queue	The queue in which the job resides.	

2.57.3.2 Job Status in Long Format

If the `-f` (full) option is given, full job status information for each job is displayed in this order:

- The job ID
- Each job attribute, one to a line
- The job's submission arguments
- The job's executable, in JSDL format
- The executable's argument list, in JSDL format

The job attributes are listed as *name = value* pairs. This includes the `exec_host` string and the `exec_vnode` string. The full output can be very large.

The `exec_host` string has the format:

*hosta/J1+hostb/J2*P+...*

where *J1* and *J2* are an index of the job on the named host and *P* is the number of processors allocated from that host to this job. *P* does not appear if it is *1*.

The `exec_vnode` string has the format:

(vnodeA:ncpus=N1:mem=M1)+(vnodeB:ncpus=N2:mem=M2)+...

where *N1* and *N2* are the number of CPUs allocated to that job on that vnode, and *M1* and *M2* are the amount of memory allocated to that job on that vnode.

2.57.3.3 Job Status in Alternate Format

The `qstat` command will display job status in the alternate format if any of the `-a`, `-i`, `-G`, `-H`, `-M`, `-n`, `-r`, `-s`, or `-u user_list` options is given. Jobs are displayed one to a line. If jobs are running and the `-n` option is specified, or if jobs are finished or moved and the `-H` and `-n` options are specified, there is a second line for the `exec_host` string.

2.57.3.3.i Output:

The output contains the following columns:

								Req'd	Req'd	Elap
Job	ID	Username	Queue	Jobname	SessID	NDS	TSK	Memory	Time	S Time
-----	-----	-----	-----	-----	-----	---	---	-----	-----	- ----

If `-n` is specified, the column output is followed by the `exec_host` string.

Description of columns:

Table 2-19: Description of Alternate Format Job Columns

Column	Description
Job ID	The <i>job_identifier</i> assigned by PBS.
Username	Username of job owner.
Queue	Queue in which the job resides.
Jobname	Job name assigned by submitter.
SessID	Session ID. Only appears if the job is running.
NDS	Number of chunks or nodes requested by the job.
TSK	Number of CPUs requested by the job.
Req'd Memory	Amount of memory requested by the job.
Req'd Time	If CPU time is requested, this shows CPU time. Otherwise, shows walltime.
S	The job's state. (See listing above.)
Elap Time	If CPU time is requested, this shows CPU time. Otherwise, shows walltime.

2.57.4 Displaying Queue Status

2.57.4.1 Queue Status in Default Format

The `qstat` command will display queue status in the default format if the only option is `-Q`, regardless of operands. Queue status is displayed one queue to a line, with these column headers:

```
Queue      Max Tot  Ena  Str Que  Run  Hld  Wat  Trn  Ext  Type
-----
```

Description of columns:

Table 2-20: Description of Default Queue Status Columns

Column	Description
Queue	Queue name.
Max	Maximum number of jobs allowed to run concurrently in the queue.
Tot	Total number of jobs in the queue.
Ena	Whether the queue is enabled or disabled.
Str	Whether the queue is started or stopped.
Que	Number of queued jobs.
Run	Number of running jobs.
Hld	Number of held jobs.
Wat	Number of waiting jobs.
Trn	Number of jobs being moved (transiting.)
Ext	Number of exiting jobs.
Type	Type of queue: execution or routing.

2.57.4.2 Queue Status in Long Format

If the `-f` (full) option is given, full queue status information for each queue is displayed starting with the queue name, followed by each attribute, one to a line, as *name = value* pairs.

2.57.4.2.i Queue Status: Alternate Format

The `qstat` command will display queue status in the alternate format if any of the `-q`, `-G` or `-M` options is given. Queue status is displayed one queue to a line, with these column headers:

```
Queue   Memory CPU Time Walltime Node Run Que Im State
-----
```

Description of columns:

Table 2-21: Description of Queue Alternate Status Columns

Column	Description
Queue	Queue name.
Memory	Maximum amount of memory that can be requested by a job in the queue.
CPU Time	Maximum amount of CPU time that can be requested by a job in the queue.
Walltime	Maximum amount of wall time that can be requested by a job in the queue.
Node	Maximum number of nodes that can be requested by a job in the queue.
Run	Number of running jobs. Lowest row is total number of running jobs in all the queues shown.
Que	Number of queued jobs. Lowest row is total number of queued jobs in all the queues shown.
Lm	Maximum number of jobs allowed to run concurrently in the queue.
State	State of the queue: <i>E</i> (enabled) or <i>D</i> (disabled), and <i>R</i> (running) or <i>S</i> (stopped).

2.57.5 Displaying Server Status

2.57.5.1 Server Status in Default Format:

The `qstat` command will display server status if the only option given is `-B`, regardless of operands.

Column headers for default server status:

```

Server  Max   Tot   Que   Run   Hld   Wat   Trn   Ext   Status
-----
```

Description of columns:

Table 2-22: Description of Server Status Default Display Columns

Column	Description
Server	Name of the server.
Max	Maximum number of jobs allowed concurrently running on the server.
Tot	Total number of jobs currently managed by the server.
Que	Number of queued jobs.
Run	Number of running jobs.
Hld	Number of held jobs.
Wat	Number of waiting jobs.
Trn	Number of transiting jobs.
Ext	Number of exiting jobs.
Status	Status of the server.
Server	Status in Long Format:

2.57.5.2 Server Status in Long Format

If the `-f` (full) option is given, full server status information is displayed starting with the server name, followed by each attribute, one to a line, as *name = value* pairs. PBS version information is listed.

2.57.6 Options to `qstat`

2.57.6.1 Default Job Status Options

`-J`

Limits status information to job arrays.

`-t`

Displays status information for jobs, job arrays, and subjobs. When used with `-J` option, limits status information to subjobs.

- p
The Time Use column is replaced with the percentage completed for the job. For an array job this is the percentage of subjobs completed. For a normal job, it is the larger of percentage used walltime or percentage used CPU time. Default format used.
- x
Displays status information for finished and moved jobs in addition to running and queued jobs.

2.57.6.2 Alternate Job Status Options

The following options will cause the alternate job status format to be used:

- a
All queued and running jobs are displayed. If a destination is given, information for all jobs at that destination is displayed. If a *job_identifier* is given, information about that job is displayed. Always specify this option before the -n or -s options, otherwise they will not take effect.
- H
Without a job identifier, displays information for all finished or moved jobs. If a job identifier is given, displays information for that job regardless of its state.
- i
If a destination is given, information for queued, held or waiting jobs at that destination is displayed. If a *job_identifier* is given, information about that job is displayed regardless of its state.
- r
If a destination is given, information for running or suspended jobs at that destination is displayed. If a *job_identifier* is given, information about that job is displayed regardless of its state.
- T
Displays estimated start time for queued jobs, replacing the *Elap Time* field with the *Est Start* field. Jobs with earlier estimated start times are displayed before those with later estimated start times.

Running jobs are displayed before other jobs. Running jobs are sorted by their *stime* attribute (start time).

Queued jobs whose estimated start times are unset (*estimated.start_time* = *unset*) are displayed after those with estimated start times, with estimated start time shown as a double dash (“--”). Queued jobs with estimated start times in the past are treated as if their estimated start times are unset.

Time displayed is local to the `qstat` command. Current week begins on Sunday.

The following table shows the format used without the `-w` option:

Table 2-23: Format for Estimated Start Time Field without `-w` Option

Format	Job's Estimated Start Time	Example
<i>HH:MM</i>	Today	15:34
<i><2-letter weekday> HH</i>	Within 7 days, but after today	We 15
<i><3-letter month name></i>	This calendar year, but after this week	Feb
<i>YYYY</i>	Less than or equal to 5 years from today, after this year	2012
<i>">5yrs"</i>	More than 5 years from today	>5yrs

The following table shows the format used with the `-w` option:

Table 2-24: Format for Estimated Start Time Field with `-w` Option

Format	Job's Estimated Start Time	Example
<i>Today HH:MM</i>	Today	Today 13:34

Table 2-24: Format for Estimated Start Time Field with -w Option

Format	Job's Estimated Start Time	Example
<i>Day HH:MM</i>	This week, but after today	Wed 15:34
<i>Day Mon Daynum HH:MM</i>	This year, but after this week	Wed Feb 10 15:34
<i>Day Mon Daynum Year HH:MM</i>	After this year	Wed Feb 10 2011 15:34

When used with the -f option, prints the full timezone-qualified start time.

If a job's estimated start time cannot be calculated, the start time is shown as a question mark ("?").

Estimated start time information can be made unavailable to unprivileged users; in this case, the estimated start time appears to be unset.

-u user_list

If a destination is given, status for jobs at that destination owned by users in **user_list** is displayed. If a *job_identifier* is given, status information for that job is displayed regardless of the job's ownership.

Hostnames may be wildcarded, but not domain names. When no hostname is specified, username is for any host.

Format: *username*[*@host*] in comma-separated list.

-n

The **exec_host** string is listed on the line below the basic information. If the -1 option is given, the **exec_host** string is listed on the end of the same line. If using the -a option, always specify the -n option after -a otherwise the -n option will not take effect.

-s

Any comment added by the administrator or scheduler is shown on the line below the basic information. If the -1 option is given, the comment string is listed on the end of the same line. If using the -a option, always specify the -s option after -a otherwise the -s option will not take effect.

-w

Allows display of wider fields. User name, Queue and Job name can be up to 15 characters wide. Session ID can be up to 8 characters wide and NDS can be up to 4 characters wide. Can only be used with **-a**, **-n** or **-s**.

-1

Reformats `qstat` output to a single line. Can only be used in conjunction with the **-n** and/or **-s** options.

2.57.6.3 Queue Status Options

-Q

Display queue status in default format. Operands must be destinations.

-q

Display queue status in alternate format. Operands must be destinations.

2.57.6.4 Server Status Options

-B

Display server status. Operands must be names of servers.

2.57.6.5 Job, Queue, and Server Status Options

-f

Full display. Job, queue or server attributes displayed one to a line.

-G

Show size in gigabytes. Alternate format is used.

-M

Show size in megawords. A word is considered to be 8 bytes. Alternate format is used.

2.57.6.6 Version Information

--version

The `qstat` command returns its PBS version information and exits. This option can only be used alone.

2.57.7 Operands

2.57.7.1 Job Identifier Operands

job_identifier

Job identifier assigned by PBS at submission. Only used with job status requests. Status information for this job is displayed.

2.57.7.2 Destination Operands

Name of queue, name of queue at a specific server, or specification of server.

Name of queue:

<queue name>

Name of queue at server:

<queue name>@<server>

All queues at a server:

@<server>

When displaying job status:

- If *<queue name>* is given, status is displayed for all jobs in the named queue at the default server.
- If *<queue name>@<server>* is given, status is displayed for all jobs in queue_name at server.
- If *@<server>* is given, status is displayed for all jobs at all queues at that server.

When displaying queue status:

- If *<queue name>* is given, status is displayed for that queue at the default server.
- If *<queue name>@<server>* is given, status is displayed for the named queue at the named server.
- If *@<server>* is given, status is displayed for all queues at that server.

2.57.7.3 Server Name Operands

server_name

Name of server. Used with the -B option to display status for that server.

2.57.8 Standard Error

The `qstat` command writes a diagnostic message to standard error for each error occurrence.

2.57.9 Exit Status

- Zero upon successful processing of all the operands.
- Greater than zero if any operands could not be processed.
- Non-zero if `x` option is not provided when querying finished jobs.

2.57.10 See Also

The PBS Professional User's Guide, the PBS Professional Administrator's Guide, `qalter(1B)`, `qsub(1B)`, `pbs_alterjob(3B)`, `pbs_statjob(3B)`, `pbs_statque(3B)`, `pbs_statserver(3B)`, `pbs_submit(3B)`, `pbs_job_attributes(7B)`, `pbs_queue_attributes(7B)`, `pbs_server_attributes(7B)`, `pbs_resources(7B)`

2.58 `qstop`

Prevents PBS jobs from running from the specified queue

2.58.1 Synopsis

qstop destination ...

qstop --version

2.58.2 Description

The `qstop` command directs that a destination queue should stop processing batch jobs. If the destination is an execution queue, the server will cease scheduling jobs that reside in the queue for execution. If the destination is a routing queue, the server will cease routing jobs from that queue.

In order to execute `qstop`, the user must have PBS Operator or Manager privilege.

2.58.3 Options

--version

The `qstop` command returns its PBS version information and exits. This option can only be used alone

2.58.4 Operands

The `qstop` command accepts one or more destination operands. The operands are one of three forms:

`<queue>`

`@<server>`

`<queue>@<server>`

If `<queue>` is specified, the request is to stop that queue at the default server. If the `@<server>` form is given, the request is to stop all the queues at that server. If a full destination identifier, `<queue>@<server>`, is given, the request is to stop the named queue at the named server.

2.58.5 Standard Error

The `qstop` command will write a diagnostic message to standard error for each error occurrence.

2.58.6 Exit Status

Upon successful processing of all the operands presented to the `qstop` command, the exit status will be a value of zero.

If the `qstop` command fails to process any operand, the command exits with a value greater than zero.

2.58.7 See Also

The PBS Professional Administrator's Guide and the following manual pages: `pbs_server(8B)`, `qstart(8B)`, and `qmgr(8B)`

2.59 qsub

Submits PBS job

2.59.1 Synopsis

```
qsub [-a date_time] [-A account_string] [-c interval] [-C directive_prefix] [-e path] [-h] [-I]
[-j join] [-J range] [-k keep] [-l resource_list] [-m mail_events] [-M user_list] [-N name]
[-o path] [-p priority] [-P project] [-q destination] [-r c] [-S path_list] [-u user_list] [-v
variable_list] [-V] [-W additional_attributes] [-X] [-z] [script | -- executable [arglist for
executable]]
```

```
qsub --version
```

2.59.2 Description

The **qsub** command is used to submit a batch job to PBS. Submitting a PBS job specifies a task, requests resources and sets job attributes.

The **qsub** command can read from a job script, from standard input, or from the command line. When the user has submitted the job, PBS returns the job identifier for that job. For a job, this is of the form:

```
sequence_number.servername
```

For an array job, this is of the form:

```
sequence_number[.servername]
```

During execution, jobs can be interactive or non-interactive.

2.59.2.1 Where PBS Puts Job Files

By default, PBS copies the **stdout** and **stderr** files from the job back to the current working directory where the **qsub** command is executed. See the **-o** and **-e** options.

2.59.2.2 Submitting Jobs By Using Scripts

To submit a PBS job script, the user types

```
qsub [options] scriptname
```

Scripts can be written in Python, UNIX shells such as `csh` and `sh`, the Windows command batch language, Perl, etc. The same Python script can be run under UNIX/Linux or under Windows. A PBS job script consists of the following:

- Optional shell specification
- Any PBS directives
- The user's tasks: programs, commands or applications

Example 2-10: A Python job script named “myjob.py” for a job named “HelloJob” that prints “Hello” under UNIX/Linux or Windows:

```
#PBS -l select=1:ncpus=3:mem=1gb
#PBS -N HelloJob
print "Hello"
```

To run a Python job script under UNIX/Linux:

```
qsub -S $PBS_EXEC/bin/pbs_python <script name>
```

To run a Python job script under Windows:

```
qsub -S %PBS_EXEC%\bin\pbs_python.exe <script name>
```

Example 2-11: A script named “weatherscript” for a job named “Weather1” which runs the executable “weathersim” on UNIX/Linux:

```
#!/bin/sh
#PBS -N Weather1
#PBS -l walltime=1:00:00
/usr/local/weathersim
```

To submit the job, the user types:

```
qsub weatherscript <return>
```

Example 2-12: A script named “weather.exe” for a job named “Weather1” which runs under Windows:

```
#PBS -N Weather1
#PBS -l walltime=1:00:00
weathersim.exe
```

To submit the job, the user types:

```
qsub weather.exe <return>
```

Scripts can contain comments. Under Windows, comments can contain only ASCII characters. See the PBS Professional User's Guide.

2.59.2.3 Submitting Jobs From Standard Input

To submit a PBS job by typing job specifications at the command line, the user types

```
qsub [options] <return>
```

then types any directives, then any tasks, followed by

- UNIX: CTRL-D on a line by itself
- Windows: CTRL-Z <return>

to terminate the input.

2.59.2.4 Submitting a Job From the qsub Command Line

To submit a job from the command line, the user types

```
qsub [options] -- executable [arguments to executable] <return>
```

Example 2-13: To run `myprog` with the arguments `a` and `b`:

```
qsub -- myprog a b <return>
```

Example 2-14: To run `myprog` with the arguments `a` and `b`, naming the job `JobA`:

```
qsub -N JobA -- myprog a b <return>
```

2.59.2.5 Requesting Resources and Placing Jobs

Requesting resources includes setting limits on resource usage and controlling how the job is placed on nodes.

Resources are requested by using the `-l` option, either in chunks inside of selection statements, or in job-wide requests using `resource_name=value` pairs. See the `pbs_resources(7B)` man page. The selection statement is of the form:

```
-l select=[N:]chunk[+[N:]chunk ...]
```

where *N* specifies how many of that chunk, and a chunk is of the form:

```
resource_name=value[:resource_name=value ...]
```

Job-wide `resource_name=value` requests are of the form:

```
-l resource_name=value[,resource_name=value ...]
```

The place statement has this form:

```
-l place=[ arrangement ][: sharing ][: grouping]
```

where

arrangement

one of *free* | *pack* | *scatter* | *vscatter*

sharing

one of *excl* | *shared* | *exclhost*

grouping

can have only one instance of *group=resource*

and where

free

Place job on any vnode(s).

pack

All chunks will be taken from one host.

scatter

Only one chunk with any MPI processes will be taken from a host. A chunk with no MPI processes may be taken from the same node as another chunk.

vscatter

Only one chunk is taken from any vnode. Each chunk must fit on a vnode.

excl

Only this job uses the vnodes chosen.

shared

This job can share the vnodes chosen.

exclhost

The entire host is allocated to the job.

group=resource

Chunks will be grouped according to a resource. All nodes in the group must have a common value for the resource, which can be either the built-in resource host or a site-defined node-level resource.

Resource must be a string or a string array.

Note that nodes can have sharing attributes that override job placement requests. See the `pbs_node_attributes(7B)` man page.

For more on resource requests, usage limits and job placement, see `pbs_resources(7B)`.

2.59.2.6 Caveats

Do not mix old style resource or node specifications with the new *select* and *place* statements. Do not use one in a job script and the other on the command line. Mixing the two will result in an error.

You cannot submit a job requesting a custom resource which has been created to be invisible or read-only for users, regardless of your privilege. A manager or operator can use the `qalter` command to change a job's request for this kind of custom resource.

2.59.2.7 Setting Attributes

The user sets job attributes by giving options to the `qsub` command or by using PBS directives. Each `qsub` option except `-C`, `-q`, and `-z` sets a job attribute, and has a corresponding PBS directive with the same syntax as the option. Attributes set via command-line options take precedence over those set using PBS directives. See the PBS Professional User's Guide, or [section 6.11, "Job Attributes", on page 375](#).

2.59.2.8 Changing `qsub` Behavior

The behavior of the `qsub` command may be affected by the server's `default_qsub_arguments` attribute. This attribute can set the default for any job attribute. The `default_qsub_arguments` server attribute is settable by the administrator, and is overridden by command-line arguments and script directives. See [section 6.6, "Server Attributes", on page 314](#).

The behavior of the `qsub` command may also be affected by any site hooks. Site hooks can modify the job's attributes, change its routing, etc.

2.59.3 Options to `qsub`

`-a date_time`

Point in time after which the job is eligible for execution. Given in pairs of digits. Sets job's `Execution_Time` attribute to `date_time`.

Format: `datetime`:

`[[[CC]YY]MM]DD]hhmm[.SS]`

where *CC* is the century, *YY* is the year, *MM* is the month, *DD* is the day of the month, *hh* is the hour, *mm* is the minute, and *SS* is the seconds.

Each portion of the date defaults to the current date, as long as the next-smaller portion is in the future. For example, if today is the 3rd of the

month and the specified day *DD* is the 5th, the month *MM* is set to the current month.

If a specified portion has already passed, the next-larger portion is set to one after the current date. For example, if the day *DD* is not specified, but the hour *hh* is specified to be 10:00 a.m. and the current time is 11:00 a.m., the day *DD* is set to tomorrow.

-A account_string

Accounting string associated with the job. Used for labeling accounting data. Sets job's **Account_Name** attribute to **account_string**.

Format: string.

-c checkpoint_spec

Determines when the job will be checkpointed. Sets job's **Checkpoint** attribute. An **\$action** script is required to checkpoint the job.

See the **pbs_mom(8B)** man page.

The argument **checkpoint_spec** can take on one of the following values:

c

Checkpoint at intervals, measured in CPU time, set on job's execution queue. If no interval set at queue, job is not checkpointed

c=<minutes of CPU time>

Checkpoint at intervals of specified number of minutes of job CPU time. This value must be > 0. If interval specified is less than that set on job's execution queue, queue's interval is used.

Format: Integer

w

Checkpoint at intervals, measured in walltime, set on job's execution queue. If no interval set at queue, job is not checkpointed.

w=<minutes of walltime>

Checkpoint at intervals of the specified number of minutes of job walltime. This value must be greater than zero. If the interval specified is less than that set on the execution queue in which the job resides, the queue's interval is used.

Format: Integer

n

No checkpointing.

s

Checkpoint only when the server is shut down.

u

Unset. Defaults to behavior when interval argument is set to s.

Default: u.

Format: String.

-C directive_prefix

Defines the prefix identifying a PBS directive. Default prefix is “*#PBS*”.

If the *directive_prefix* argument is a null string, *qsub* does not scan the script file for directives. Overrides the *PBS_DPREFIX* environment variable and the default. Cannot be used as a PBS directive.

-e path

Path to be used for the job’s standard error stream. Sets job’s *Error_Path* attribute to *path*. The path argument is of the form:

[hostname:]path_name

The path is interpreted as follows:

path_name

If *path_name* is a relative path, then it is taken to be relative to the current working directory of the *qsub* command, where it is executing on the current host.

If *path_name* is an absolute path, then it is taken to be an absolute path on the current host where the *qsub* command is executing.

hostname:path_name

If *path_name* is a relative path, then it is taken to be relative to the user’s home directory on the host named *hostname*.

If *path_name* is an absolute path, then it is the absolute path on the host named *hostname*.

If *path_name* does not include a filename, the default filename is

jobid.ER

If the *-e* option is not specified, PBS copies the standard error to the current working directory where the *qsub* command was executed. The default filename for the standard error stream is used. It has this form:

job_name.e<sequence number>

-h

Applies a user hold to the job. Sets the job’s *Hold_Types* attribute to “*u*”.

-I

Job is to be run interactively. Sets job’s *interactive* attribute to *True*. The job is queued and scheduled as any PBS batch job, but when executed, the

standard input, output, and error streams of the job are connected to the terminal session in which `qsub` is running. If a job script is given, only its directives are processed. When the job begins execution, all input to the job is taken from the terminal session. See the PBS Professional User's Guide for additional information on interactive jobs.

Interactive jobs are not rerunnable.

Job arrays cannot be interactive.

When used with `-Wblock=true`, no exit status is returned.

Not supported on Windows.

-j join

Whether and how to join the job's standard error and standard output streams. Sets job's `Join_Path` attribute to `join`.

Possible values of `join`:

Table 2-25: Suboptions to Join Option

Suboption	Meaning
<i>oe</i>	Standard error and standard output are merged into standard output.
<i>eo</i>	Standard error and standard output are merged into standard error.
<i>n</i>	Standard error and standard output are not merged.

Default: not merged.

-J range

Declares that this job is an array job. Sets job's `array` attribute to *True*.

The argument `range` identifies the integers greater than or equal to zero that are associated with the subjobs of the array. *range* is specified in the form *X-Y[:Z]* where *X* is the first index, *Y* is the upper bound on the indices and *Z* is the stepping factor. For example, `2-7:2` will produce indices of 2, 4, and 6. If *Z* is not specified, it is taken to be 1.

-k keep

Specifies whether and which of the standard output and standard error streams is retained on the execution host. Overrides default path names for these streams. Sets the job's `Keep_Files` attribute to `keep`.

Default: neither is retained.

In the case where output and/or error is retained on the execution host in a job-specific staging and execution directory created by PBS, these files are deleted when PBS deletes the directory.

The **keep** argument can take on the following values:

Table 2-26: Suboptions to keep Option

Suboption	Meaning
<i>e</i>	The standard error stream is retained on the execution host, in the job's staging and execution directory. The filename is <i>job_name.e<sequence number></i>
<i>o</i>	The standard output stream is retained on the execution host, in the job's staging and execution directory. The filename is <i>job_name.o<sequence number></i>
<i>eo, oe</i>	Both standard output and standard error streams are retained on the execution host, in the job's staging and execution directory.
<i>n</i>	Neither stream is retained.

-l resource_list

Allows the user to request resources and specify job placement. Sets job's **Resource_list** attribute to **resource_list**. Requesting a resource places a limit on its usage.

Requesting resources in chunks:

-l select=[N:]chunk[+[N:]chunk ...]

where N specifies how many of that chunk, and a chunk is:

resource_name=value[:resource_name=value ...]

Requesting job-wide resources:

-l resource_name=value[,resource_name=value ...]

Specifying placement of jobs:

-l place=modifier[:modifier]

where modifier is any combination of *group*, *excl*, and/or one of *free*|*pack*|*scatter*.

For more on resource requests, usage limits and job placement, see `pbs_resources(7B)`.

-m mail_events

The set of conditions under which mail about the job is sent. Sets job's `Mail_Points` attribute to `mail_events`. The `mail_events` argument can be either “*n*” or any combination of *a* , *b* , and *e* .

Table 2-27: Suboptions to m Option

Suboption	Meaning
<i>n</i>	No mail will be sent.
<i>a</i>	Mail is sent when the job is aborted by the batch system.
<i>b</i>	Mail is sent when the job begins execution.
<i>e</i>	Mail is sent when the job terminates.

Format: string.

Default value: “a”.

-M user_list

List of users to whom mail about the job is sent. Sets job's `Mail_Users` attribute to `user_list`.

The `user_list` argument is of the form:

user[@host][,user[@host],...]

Default: job owner.

-N name

Sets job's `Job_Name` attribute and name to `name`.

Format: string, up to 15 characters in length. It must consist of an alphabetic or numeric character followed by printable, nonwhite-space characters.

Default: if a script is used to submit the job, the job's name is the name of the script. If no script is used, the job's name is “*STDIN*”.

-o path

Path to be used for the job's standard output stream. Sets job's `Output_Path` attribute to `path`. The `path` argument is of the form:

[hostname:]path_name

The path is interpreted as follows:

path_name

If *path_name* is a relative path, then it is taken to be relative to the current working directory of the command, where it is executing on the current host.

If *path_name* is an absolute path, then it is taken to be an absolute path on the current host where the command is executing.

hostname:path_name

If *path_name* is a relative path, then it is taken to be relative to the user's home directory on the host named *hostname*.

If *path_name* is an absolute path, then it is the absolute path on the host named *hostname*.

If *path_name* does not include a filename, the default filename is

jobid.OU

If the **-o** option is not specified, PBS copies the standard output to the current working directory where the **qsub** command was executed. The default filename for the standard output stream is used. It has this form:

job_name.o<*sequence number*>

-p priority

Priority of the job. Sets job's Priority attribute to priority.

Format: host-dependent integer.

Range: [-1024, +1023] inclusive.

Default: *zero*.

-P project

Specifies a project for the job. Sets job's **project** attribute to specified value.

Format: String.

Project name can contain any characters except for the following: Slash ("/"), left bracket ("["), right bracket ("]"), double quote ("\""), semicolon (";"), colon (":"), vertical bar ("|"), left angle bracket ("<"), right angle bracket (">"), plus ("+"), comma (","), question mark ("?"), and asterisk ("*").

Default value: "*_pbs_project_default*".

-q destination

Where the job is sent upon submission.

Specifies a queue, a server, or a queue at a server. The destination argument can have one of these formats:

<*queue*>

Job is submitted to the named queue at the default server.

@<server>

Job is submitted to the default queue at the named server.

<queue>@<server>

Job is submitted to the named queue at the named server.

Default: default queue at default server.

-r y|n

Declares whether the job is rerunnable. See the `qrerun (1B)` command.

Sets job's `Rerunnable` attribute to the argument.

Format: single character, "y" or "n".

Table 2-28: Suboptions to r Option

Suboption	Meaning
<i>y</i>	Job is rerunnable.
<i>n</i>	Job is not rerunnable.

Default: "n".

Interactive jobs are not rerunnable.

-S path_list

Specifies the interpreter or shell path for the job script. Sets job's `Shell_Path_List` attribute to `path_list`.

The `path_list` argument is the full path to the interpreter or shell including the executable name.

Only one path may be specified without a host name. Only one path may be specified per named host. The path selected is the one whose host name is that of the server on which the job resides.

Format:

path[@host][.path@host ...]

Default: user's login shell on execution node.

Example of using `bash` via a directive:

#PBS -S /bin/bash@mars,/usr/bin/bash@jupiter

Example of running a Python script from the command line on UNIX/Linux:

qsub -S \$PBS_EXEC/bin/pbs_python <script name>

Example of running a Python script from the command line on Windows:

```
qsub -S %PBS_EXEC%\bin\pbs_python.exe <script name>
```

-u user_list

List of usernames. Job is run under a username from this list. Sets job's `User_List` attribute to `user_list`.

Only one username may be specified without a host name. Only one username may be specified per named host. The server on which the job resides will select first the username whose host name is the same as the server name. Failing that, the next selection will be the username with no specified hostname. The usernames on the server and execution hosts must be the same. The job owner must have authorization to run as the specified user.

Format of `user_list`:

```
user[@host]/.user@host ...]
```

Default: job owner (username on submit host.)

-v variable_list

Lists environment variables to be exported to the job. This is the list of environment variables which is added to those already automatically exported. These variables exist in the user's login environment from which `qsub` is run. The job's `Variable_List` attribute is appended with the variables in `variable_list` and their values. See [section 2.59.7, "Environment Variables", on page 227](#).

Format: comma-separated list of strings in the form:

variable

or

variable=value

If a *variable=value* pair contains any commas, the value must be enclosed in single or double quotes, and the *variable=value* pair must be enclosed in the kind of quotes not used to enclose the value. For example:

```
qsub -v "var1='A,B,C,D'" job.sh
```

```
qsub -v a=10, "var2='A,B'", c=20, HOME=/home/zzz job.sh
```

Default: no environment variables are added to job's variable list.

-V

Declares that all environment variables in the user's login environment where `qsub` is run are to be exported to the job. The job's `Variable_List` attribute is appended with all of these environment variables and their values.

-W additional_attributes

The -W option allows specification of any job attribute. Some job attributes must be specified using this option. Those attributes are listed below. Format:

-W attribute_name=value[,attribute_name=value...]

If white space occurs within the additional_attributes argument, or the equal sign “=” occurs within an attribute_value string, then that must be enclosed with single- or doublequotes.

The following attributes must be set using the -W option:

depend=dependency_list

Defines dependencies between this and other jobs. Sets the job’s depend attribute to dependency_list. The dependency_list has the form:

type:arg_list[,type:arg_list ...]

where except for the *on* type, the *arg_list* is one or more PBS job IDs in the form:

jobid[:jobid ...]

The type can be:

after: arg_list

This job may be scheduled for execution at any point after all jobs in *arg_list* have started execution.

afterok: arg_list

This job may be scheduled for execution only after all jobs in *arg_list* have terminated with no errors. See “Warning about exit status with csh” in Exit Status.

afternotok: arg_list

This job may be scheduled for execution only after all jobs in *arg_list* have terminated with errors. See [section 2.59.8.1, “Warning About Exit Status with csh”, on page 229](#).

afterany: arg_list

This job may be scheduled for execution after all jobs in *arg_list* have finished execution, with any exit status (with or without errors.) This job will not run if a job in the *arg_list* was killed.

before: arg_list

Jobs in *arg_list* may begin execution once this job has begun execution.

beforeok: arg_list

Jobs in *arg_list* may begin execution once this job terminates without errors. See “Warning about exit status with csh” in Exit Status.

beforenotok: arg_list

If this job terminates execution with errors, then jobs in *arg_list* may begin. See [section 2.59.8.1, “Warning About Exit Status with csh”, on page 229](#).

beforeany: arg_list

Jobs in *arg_list* may begin execution once this job terminates execution, with or without errors.

on: count

This job may be scheduled for execution after *count* dependencies on other jobs have been satisfied. This type is used in conjunction with one of the *before* types listed. *count* is an integer greater than 0.

Job IDs in the *arg_list* of *before* types must have been submitted with a type of *on*.

To use the *before* types, the user must have the authority to alter the jobs in *arg_list*. Otherwise, the dependency is rejected and the new job aborted.

Error processing of the existence, state, or condition of the job on which the newly submitted job is a deferred service, i.e. the check is performed after the job is queued. If an error is detected, the new job will be deleted by the server. Mail will be sent to the job submitter stating the error.

Dependency examples:

```
qsub -W depend=afterok:123.host1.domain.com /tmp/script
```

```
qsub -W depend=before:234.host1.com:235.host1.com /tmp/script
```

group_list=g_list

List of group names. Job is run under a group name from this list. Sets job's *group_List* attribute to *g_list*.

Only one group name may be specified without a host name. Only one group name may be specified per named host. The server on which the job resides will select first the group name whose host name is the same as the server name. Failing that, the next selection will be the group

name with no specified hostname. The group names on the server and execution hosts must be the same.

Under Windows, the primary group is the first group found for the user by PBS when it queries the accounts database.

Format of `g_list`:

group[@host][,group@host ...]

Default: login group name of job owner.

`pwd`

`pwd=`

`pwd=`

These forms prompt the user for a password. A space between `W` and `pwd` is optional. Spaces between the quotes are optional. Examples:

```
qsub ... -Wpwd <return>
```

```
qsub ... -W pwd=' ' <return>
```

```
qsub ... -W pwd=" " <return>
```

Available on Windows and supported Linux x86 and x86_64 platforms only.

`block=true`

Specifies that `qsub` waits for the job to terminate, then returns the job's exit value. Sets job's `block` attribute to *TRUE*. When used with X11 forwarding or interactive jobs, no exit value is returned. See [section 2.59.8, "Exit Status", on page 229](#).

`sandbox=<value>`

Determines which directory PBS uses for the job's staging and execution. If value is *PRIVATE*, PBS creates a job-specific directory for staging and execution. If value is *HOME* or is unset, PBS uses the user's home directory for staging and execution.

`stagein=path_list`

`stageout=path_list`

Specifies files or directories to be staged-in before execution or staged-out after execution is complete. Sets the job's `stagein` and `stageout` attributes to the specified `path_lists`. On completion of the job, all staged-in and staged-out files and directories are removed from the execution host(s). The `path_list` has the form:

filespec[.filespec]

where *filespec* is

local_path@hostname:remotepath

regardless of the direction of the copy. The name *local_path* is the name of the file or directory on the primary execution host. It can be relative to the staging and execution directory on the execution host, or it can be an absolute path.

The “@” character separates *local_path* from *remote_path*.

The name *remote_path* is the path on *hostname*. The name can be relative to the staging and execution directory on the primary execution host, or it can be an absolute path.

If *path_list* has more than one *filespec*, i.e. it contains commas, it must be enclosed in double-quotes.

umask=NNNN

The umask with which the job is started. Sets job’s umask attribute to NNNN. Controls umask of job’s standard output and standard error.

The following example allows group and world read on the job’s output:

```
-W umask=33
```

Can be used with one to four digits; typically two.

Default value: *077*

-X

Allows user to receive X output from interactive job.

DISPLAY variable in submission environment must be set to desired display.

Can be used with interactive jobs only: must be used with **-I** or **-W interactive=true**.

Cannot be used with **-v DISPLAY**.

When used with **-Wblock=true**, no exit status is returned.

Can be used with **-V** option.

Not available under Windows.

-Z

Job identifier is not written to standard output.

--version

The **qsub** command returns its PBS version information and exits. This option can only be used alone.

2.59.4 Operands

The `qsub` command accepts as operands one of the following:

(script)

Path to script. Can be absolute or relative to current directory where `qsub` is run.

-

(a dash)

Any PBS directives and user tasks are read from the command line. Same as for no operands.

-- executable [arguments to executable]

a single executable (preceded by two dashes) and its arguments

The executable, and any arguments to the executable, are given on the `qsub` command line. The executable is preceded by two dashes, "--".

If a script or executable is specified, it must be the last argument to `qsub`. The arguments to an executable must follow the name of the executable.

2.59.5 Standard Output

Unless the `-Z` option is set, the job identifier assigned to the job is written to standard output if the job is successfully created.

2.59.6 Standard Error

The `qsub` command writes a diagnostic message to standard error for each error occurrence.

2.59.7 Environment Variables

The `qsub` command uses the following environment variables:

PBS_DEFAULT

Name of default server.

PBS_DPREFIX

Prefix string which identifies PBS directives.

Environment variables beginning with "*PBS_O_*" are created by `qsub`. PBS automatically exports the following environment variables to the job, and the job's `Variable_List` attribute is set to this list:

PBS_ENVIRONMENT

Set to *PBS_BATCH* for a batch job. Set to *PBS_INTERACTIVE* for an interactive job. Created upon execution.

PBS_JOBDIR

Pathname of job's staging and execution directory on the primary execution host.

PBS_JOBID

Job identifier given by PBS when the job is submitted. Created upon execution.

PBS_JOBNAME

Job name given by user. Created upon execution.

PBS_NODEFILE

Name of file containing the list of nodes assigned to the job. Created upon execution.

PBS_O_HOME

User's home directory. Value of **HOME** taken from user's submission environment.

PBS_O_HOST

Name of submit host. Value taken from user's submission environment.

PBS_O_LANG

Value of **LANG** taken from user's submission environment.

PBS_O_LOGNAME

User's login name. Value of **LOGNAME** taken from user's submission environment.

PBS_O_MAIL

Value of **MAIL** taken from user's submission environment.

PBS_O_PATH

User's **PATH**. Value of **PATH** taken from user's submission environment.

PBS_O_QUEUE

Name of the queue to which the job was submitted. Value taken from user's submission environment.

PBS_O_SHELL

Value taken from user's submission environment.

PBS_O_SYSTEM

Operating system, from `uname -s`, on submit host. Value taken from user's submission environment.

PBS_O_TZ

Value taken from user's submission environment.

PBS_O_WORKDIR

Absolute path to directory where `qsub` is run. Value taken from user's submission environment.

PBS_QUEUE

Name of the queue from which the job is executed. Created upon execution.

TMPDIR

Pathname of job's scratch directory.

2.59.8 Exit Status

Zero upon successful processing of input. Exit value is greater than zero upon failure of `qsub`.

For blocking jobs, `qsub` exits and returns the exit value of the job. If the job is deleted without being run, `qsub` returns an exit value of 3.

2.59.8.1 Warning About Exit Status with `cs`

If a job is run in `cs` and a `.logout` file exists in the home directory in which the job executes, the exit status of the job is that of the `.logout` script, not the job script. This may impact any inter-job dependencies.

2.59.9 See Also

The PBS Professional User's Guide, the PBS Professional Administrator's Guide, `pbs_job_attributes(7B)`, `pbs_server_attributes(7B)`, `pbs_resources(7B)`, `qalter(1B)`, `qhold(1B)`, `qmove(1B)`, `qmsg(1B)`, `qrerun(1B)`, `qrls(1B)`, `qselect(1B)`, `qstat(1B)`

2.60 `qterm`

Terminates a PBS server

2.60.1 Synopsis

```
qterm [ -f | -F | -i ] [ -m ] [ -s ] [ -t type ] [ server[ server ...]]
qterm --version
```

2.60.2 Description

The `qterm` command terminates a PBS batch server.

Once the server is terminating, no new jobs are accepted by the server, and no jobs are allowed to begin execution. The impact on running jobs depends on the way the server is shut down.

The `qterm` command does not exit until the server has completed its shutdown procedure.

If the complex is configured for failover, and the primary server is shut down, the normal behavior for the secondary server is to become active. The `qterm` command provides options to manage the behavior of the secondary server; it can be shut down, forced to remain idle, or shut down in place of the primary server.

In order to run the `qterm` command, the user must have PBS Operator or Manager privilege.

2.60.3 Options

The following table lists the options to the `qterm` command.

Table 2-29: `qterm` Options

(no option)	The <code>qterm</code> command defaults to <code>-t quick</code> .
<code>-f</code>	<p>If the complex is configured for failover, both the primary and secondary servers are shut down.</p> <p>Without the <code>-f</code> option, the primary server is shut down and the secondary server becomes active.</p> <p>The <code>-f</code> option cannot be used with the <code>-i</code> or <code>-F</code> options.</p>
<code>-F</code>	<p>If the complex is configured for failover, only the secondary server is shut down, and the primary server remains active.</p> <p>The <code>-F</code> option cannot be used with the <code>-f</code> or <code>-i</code> options.</p>

Table 2-29: qterm Options

-i	<p>If the complex is configured for failover, the secondary server remains idle when the primary server is shut down.</p> <p>The -i option cannot be used with the -f or -F options.</p>	
-m	<p>All MOMs (pbs_mom) are shut down. This option does not cause jobs or subjobs to be killed. Jobs are left running subject to other options to the qterm command.</p>	
-s	<p>The scheduler (pbs_sched) is shut down.</p>	
-t <type>	immediate	<p>All running jobs immediately stop execution. Any running jobs that can be checkpointed are checkpointed, terminated, and requeued. Jobs that cannot be checkpointed are terminated and requeued if they are rerunnable, otherwise they are killed.</p> <p>If any job cannot be terminated, for example the server cannot contact the MOM of a running job, the server continues to execute and the job is listed as running. The server can be terminated by a second qterm -t immediate command.</p> <p>While terminating, the server is in the <i>Terminating</i> state.</p>
	delay	<p>The server waits to terminate until all non-checkpointable, non-rerunnable jobs are finished executing. Any running jobs that can be checkpointed are checkpointed, terminated, and requeued. Jobs that cannot be checkpointed are terminated and requeued if they are rerunnable, otherwise they are allowed to continue to run.</p> <p>While terminating, the server is in the <i>Terminating-Delayed</i> state.</p>
	quick	<p>Running jobs are left running. Running subjobs are requeued.</p> <p>This is the default behavior when no options are given to the qterm command.</p> <p>While terminating, the server is in the <i>Terminating</i> state.</p>
--version	<p>The qterm command returns its PBS version information and exits. This option can only be used alone.</p>	

2.60.4 Operands

The server list operand specifies which servers are to shut down. It is a space-separated list of server names. If no servers are specified, then the default server is shut down.

2.60.4.1 Standard Error

The `qterm` command writes a diagnostic message to standard error for each error occurrence.

2.60.4.2 Exit Status

Zero upon successful processing of all the operands presented to the `qterm` command.
Greater than zero if the `qterm` command fails to process any operand.

2.60.4.3 See Also

The PBS Professional Administrator's Guide, `pbs_server(8B)`, `pbs_mom(8B)`, `pbs_sched(8B)`

2.61 `tracejob`

Prints log messages for a PBS job

2.61.1 Synopsis

```
tracejob [-a] [-c count] [-f filter] [-l] [-m] [-n days] [-p path] [-s] [-v] [-w cols] [-z] jobid  
tracejob --version
```

2.61.2 Description

The `tracejob` command extracts log messages for a given *jobid* and prints them in chronological order.

Log messages contain server, scheduler, accounting and MOM information. Server logs contain information such as when a job was queued or modified. Scheduler logs contain clues as to why a job is not running. Accounting logs contain accounting records for when a job was queued, started, ended or deleted. MOM logs contain information about what happened to a job while it was running.

To get MOM log messages for a job, `tracejob` must be run on the machine on which the job ran.

All users have access to server, scheduler and MOM information. Only Administrator or root can access accounting information.

Some log messages appear many times. In order to make the output of `tracejob` more readable, messages that appear over a certain number of times (see option `-c` below) are restricted to only the most recent message.

If `tracejob` is run on a job array, the information returned will be about the job array itself, and not its subjobs. Job arrays do not have associated MOM log messages. If `tracejob` is run on a subjob, the same types of log messages will be available as for a job. Certain log messages that occur for a regular job will not occur for a subjob.

Note that some shells require that you enclose a job array identifier in double quotes.

2.61.3 Options to `tracejob`

- `-a` Do not report accounting information.
- `-c <count>` Set excessive message limit to `count`. If a message is logged at least `count` times, only the most recent message is printed.
The default for `count` is 15.
- `-f <filter>` Do not include log events of type `filter`. The `-f` option can be used more than once on the command line.
Filter values:
Filter values can be any of: *error*, *system*, *admin*, *job*, *job_usage*, *security*, *sched*, *debug*, *debug2*, *resv*, *debug3*, or *debug4*, or their equiva-

lent in hexadecimal. The following table shows the hex value and category for each filter.

Table 2-30: Filters

Filter	Hex Value	Message Category
<i>error</i>	<i>0x0001</i>	Internal errors
<i>system</i>	<i>0x0002</i>	System errors
<i>admin</i>	<i>0x0004</i>	Administrative events
<i>job</i>	<i>0x0008</i>	Job-related events
<i>job_usage</i>	<i>0x0010</i>	Job accounting info
<i>security</i>	<i>0x0020</i>	Security violations
<i>sched</i>	<i>0x0040</i>	Scheduler events
<i>debug</i>	<i>0x0080</i>	Common debug messages
<i>debug2</i>	<i>0x0100</i>	Uncommon debug messages
<i>resv</i>	<i>0x0200</i>	Reservation debug messages
<i>debug3</i>	<i>0x0400</i>	Less common than debug2
<i>debug4</i>	<i>0x0800</i>	Less common than debug3

-l

Do not report scheduler information.

-m

Do not report MOM information.

-n <days>

Report information from up to days days in the past.

Default is *1* = today.

-p <path>

Use *path* as path to **PBS_HOME** on machine being queried.

-s

Do not report server information.

-
- w <cols>**
Width of current terminal. If not specified by the user, `tracejob` queries OS to get terminal width. If OS doesn't return anything, default is `80`.
 - v**
Verbose. Report more of `tracejob`'s errors than default.
 - Z**
Suppresses printing of duplicate messages.
 - version**
The `tracejob` command returns its PBS version information and exits. This option can only be used alone.

2.61.4 Exit Status

Zero upon successful processing of all options.

Exit value is greater than zero if `tracejob` is unable to process any options.

2.61.5 See Also

The PBS Professional Administrator's Guide

`pbs_server(8B)`, `pbs_sched(8B)`, `pbs_mom(8B)`

2.62 xpbs

GUI front end to PBS commands

2.62.1 Synopsis

xpbs [-admin]

xpbs --version

2.62.2 Description

The `xpbs` command provides a user-friendly point-and-click interface to PBS commands. Please see the sections below for a tour and tutorials. Also, within every dialog box, a Help button can be found for assistance.

2.62.3 Options

-admin

A mode where additional buttons are made available for terminating PBS servers, starting/stopping/disabling/enabling queues, and running/rerunning jobs.

--version

The `xpbs` command returns its PBS version information and exits. This option can only be used alone.

2.62.4 Getting Started

Running `xpbs` will initialize the X resource database from various sources in the following order:

1. The `RESOURCE_MANAGER` property on the root window (updated via `xrdb`) with settings usually defined in the `.Xdefaults` file
2. Preference settings defined by the system administrator in the global `xpbsrc` file
3. User's `~/ .xpbsrc` file - this file defines various X resources like fonts, colors, list of PBS hosts to query, criteria for listing queues and jobs, and various view states. See [section 2.62.13, “Setting Preferences”, on page 245](#) below for a list of resources that can be set.

2.62.5 Running `xpbs`

To run `xpbs` as a regular, non-privileged user, type:

```
setenv DISPLAY <display_host>:0
xpbs
```

To run `xpbs` with the additional purpose of terminating PBS servers, stopping and starting queues, or running/rerunning jobs, then run:

```
xpbs -admin
```

NOTE: Be sure to appropriately set `~/ .rhosts` file if you're planning to submit jobs to some remote server, and expecting output files to be returned to the local host (where `xpbs` was run). Usually, adding the PBS hostname running the server to your `.rhosts` file locally, and adding the name of the local machine to the `.rhosts` file at remote host, should be sufficient.

Also, be sure that the PBS client commands are in the default `PATH` because `xpbs` will call these commands.

2.62.6 The **xpbs** Display

This section describes the main parts of the **xpbs** display. The main window is composed of 5 distinct areas (subwindows) arranged vertically (one on top of another) in the following order:

1. Menu
2. Hosts
3. Queues
4. Jobs
5. Info

2.62.6.1 Menu

The Menu area is composed of a row of command buttons that signal some action with a click of the left mouse button. The buttons are:

Manual Update

to update the information on hosts, queues, and jobs.

Auto Update

same as Manual Update except updating is done automatically every <some specified> number of minutes.

Track Job

for periodically checking for returned output files of jobs.

Preferences

for setting certain parameters such as the list of server host(s) to query.

Help

contains some help information.

About

tells of the author and who to send comments, bugs, suggestions to.

Close

for exiting **xpbs** plus saving the current setup information (if anything had changed) in the user's `$HOME/.xpbsrc` file. Information saved include the selected host(s), queue(s), job(s), the different jobs listing criteria, the view states (i.e. minimized/maximized) of the Hosts, Queues, Jobs, and INFO regions, and anything in the Preferences section.

2.62.6.2 Hosts

The Hosts area is composed of a leading horizontal HOSTS bar, a listbox, and a set of command buttons. The HOSTS bar contains a minimize/maximize button, identified by a dot or a rectangular image, for displaying or iconifying the Hosts region. The listbox displays information about favorite server host(s), and each entry is meant to be selected via a single left mouse button click, shift key + mouse button 1 click for contiguous selection, or cntrl key + mouse button 1 click for non-contiguous selection. The command buttons represent actions on selected host(s), and commonly found buttons are:

detail

for obtaining detailed information about selected server host(s). This functionality can also be achieved by double clicking on an entry in the Hosts listbox.

Submit

for submitting a job to any of the queues managed by the selected host(s).

terminate

for terminating PBS servers on selected host(s). (-admin only)

The server hosts can be chosen by specifying in the ~/.xpbsrc file (or .Xdefaults) the resource:

```
*serverHosts: hostname1 hostname2 ...
```

Another way of specifying the host is to click on the Preferences button in the Menu region, and manipulate the server Hosts entry widget from the preferences dialog box.

2.62.6.3 Queues

The Queues area is composed of a leading horizontal QUEUES bar, a listbox, and a set of command buttons. The QUEUES bar lists the hosts that are consulted when listing queues; the bar also contains a minimize/maximize button for displaying or iconifying the Queues region. The listbox displays information about queues managed by the server host(s) selected from the Hosts listbox; each listbox entry is meant to be selected (highlighted) via a single left mouse button click, shift key + mouse button 1 click for contiguous selection, or cntrl key + mouse button 1 click for non-contiguous selection. The command buttons represent actions for operating on selected queue(s), and commonly found buttons are:

detail

for obtaining detailed information about selected queue(s). This functionality can also be achieved by double clicking on a Queues listbox entry.

stop

for stopping the selected queue(s). (-admin only)

start	for starting the selected queue(s). (-admin only)
disable	for disabling the selected queue(s). (-admin only)
enable	for enabling the selected queue(s). (-admin only)

2.62.6.4 Jobs

The Jobs area is composed of a leading horizontal JOBS bar, a listbox, and a set of command buttons. The JOBS bar lists the queues that are consulted when listing jobs; the bar also contains a minimize/maximize button for displaying or iconifying the Jobs region. The listbox displays information about jobs that are found in the queue(s) selected from the Queues listbox; each listbox entry is meant to be selected (highlighted) via a single left mouse button click, shift key + mouse button 1 click for contiguous selection, or ctrl key + mouse button 1 click for non-contiguous selection. The region just above the Jobs listbox shows a collection of command buttons whose labels describe criteria used for filtering the Jobs listbox contents. The list of jobs can be selected according to the owner of jobs (**Job_Owner**), job state (**Job_State**), name of the job (**Job_Name**), type of hold placed on the job (**Hold_Types**), the account name associated with the job (**Account_Name**), checkpoint attribute (**Checkpoint**), time the job is eligible for queueing/execution (**Queue_Time**), resources requested by the job (**Resource_List**), priority attached to the job (**Priority**), and whether or not the job is rerunnable (**Rerunnable**). The selection criteria can be modified by clicking on any of the appropriate command buttons to bring up a selection box. The criteria command buttons are accompanied by a Select Jobs button, which when clicked, will update the contents of the Jobs listbox based on the new selection criteria. Please see `qselect (1B)` for more details on how the jobs are filtered.

Finally, to the right of the listbox, the Jobs region is accompanied by the following command buttons, for operating on selected job(s):

detail	for obtaining detailed information about selected job(s). This functionality can also be achieved by double clicking on a Jobs listbox entry.
modify	for modifying attributes of the selected job(s).
delete	for deleting the selected job(s).
hold	for placing some type of hold on selected job(s).

release	for releasing held job(s).
signal	for sending signals to selected job(s) that are running.
msg	for writing a message string into the output streams of the selected job(s).
move	for moving selected job(s) into some specified destination queue.
order	for exchanging order of two selected jobs in a queue.
run	for running selected job(s). (-admin only)
rerun	for requeueing selected job(s) that are running. (-admin only)

2.62.6.5 Info

The Info Area shows the progress of the commands' executed by **xpbs**. Look into this box for errors. The INFO bar also contains a minimize/maximize button for displaying or iconifying the Info region.

2.62.7 Widgets Used in **xpbs**

Some of the widgets used in **xpbs** and how they are manipulated are described in the following:

2.62.7.1 listbox

can be multi-selectable (a number of entries can be selected/highlighted using a mouse click) or single-selectable (one entry can be highlighted at a time). For a multi-selectable listbox, the following operations are allowed:

1. single click with mouse button 1 to select/highlight an entry.
2. shift key + mouse button 1 to contiguously select more than one entry.
3. cntrl key + mouse button 1 to non-contiguously select more than one entry. NOTE: For systems running Tk < 4.0, the newly selected item is reshuffled to appear next to already selected items.
4. click the Select All/Deselect All button to select all entries or deselect all entries at once.
5. double clicking an entry usually activates some action that uses the selected entry as a parameter.

2.62.7.2 scrollbar

usually appears either vertically or horizontally and contains 5 distinct areas that are mouse clicked to achieve different effects:

top arrow

Causes the view in the associated widget to shift up by one unit (i.e. the object appears to move down one unit in its window). If the button is held down the action will auto-repeat.

top gap

Causes the view in the associated window to shift up by one less than the number of units in the window (i.e. the portion of the object that used to appear at the very top of the window will now appear at the very bottom). If the button is held down the action will auto-repeat.

slider

Pressing button 1 in this area has no immediate effect except to cause the slider to appear sunken rather than raised. However, if the mouse is moved with the button down then the slider will be dragged, adjusting the view as the mouse is moved.

bottom gap

Causes the view in the associated window to shift down by one less than the number of units in the window (i.e. the portion of the object that used to appear at the very bottom of the window will now appear at the very top). If the button is held down the action will auto-repeat.

bottom arrow

Causes the view in the associated window to shift down by one unit (i.e. the object appears to move up one unit in its window). If the button is held down the action will auto-repeat.

2.62.7.3 entry

brought into focus with a click of the left mouse button. To manipulate this widget, simply type in the text value. Use of arrow keys, mouse selection of text for deletion or overwrite, copying and pasting with sole use of mouse buttons are permitted. This widget is usually accompanied by a scrollbar for horizontally scanning a long text entry string.

2.62.7.4 matrix of entry boxes

usually shown as several rows of entry widgets where a number of entries (called fields) can be found per row. The matrix is accompanied by up/down arrow buttons for paging through the rows of data, and each group of fields gets one scrollbar for horizontally scanning long entry strings. Moving from field to field can be done using the <Tab>, <Cntrl-f>, or <Cntrl-b> (move backwards) keys.

2.62.7.5 spinbox

a combination of an entry widget and a horizontal scrollbar. The entry widget will only accept values that fall within a defined list of valid values, and incrementing through the valid values is done by clicking on the up/down arrows.

2.62.7.6 button

a rectangular region appearing either raised or pressed that invokes an action when clicked with the left mouse button. When the button appears pressed, then hitting the <RETURN> key will automatically select the button.

2.62.7.7 text

an editor like widget. This widget is brought into focus with a click of the left mouse button. To manipulate this widget, simply type in the text. Use of arrow keys, backspace/delete key, mouse selection of text for deletion or overwrite, copying and pasting with sole use of mouse buttons are permitted. This widget is usually accompanied by a scrollbar for vertically scanning a long entry.

2.62.8 Submitting Jobs

Submitting a PBS job requires only to manipulate the widgets found in the Submit window. The submit dialog box is composed of 4 distinct regions:

1. Job Script
2. Options
3. OTHER Options
4. Command Buttons

The Job Script file region is at the upper left, the Options region containing various widgets for setting job attributes is scattered all over the dialog box, the OTHER Options is located just below the Job Script file region, and Command Buttons region is at the bottom.

The job script region is composed of a header box, the text box, FILE entry box, and a couple of buttons labeled load and save. If you have a script file containing PBS options and executable lines, then type the name of the file on the FILE entry box, and then click on the load button. The various widgets in the Submit window will get loaded with values found in the script file. The script file text box will only be loaded with executable lines (non-PBS) found in the script. The job script header box has a Prefix entry box that can be modified to specify the PBS directive to look for when parsing a script file for PBS options. If you don't have a script file, you can start typing the executable lines of the job in the file text box.

To submit a job, perform the following steps:

1. Select a host from the HOSTS listbox in the main `xpbs` display.
2. Click on the Submit button located in the Menu bar.
3. Specify the script file containing the job execution lines and job resource and attribute values, or simply type in the execution lines in the FILE textbox.
4. Start manipulating the various widgets in the Submit window. Particularly, pay close attention to the Destination listbox. This box lists all the queues found in the host that you selected. A special entry called “@host” refers to the default queue at host. Select appropriately the destination queue of the job. More options can be found by clicking the OTHER Options buttons.
5. At the bottom of the Submit window, click confirm submit . You can also click on interactive to run the job interactively. Running a job interactively will open an xterm window to your display host containing the session.

NOTE: The script FILE entry box is accompanied by a save button that you click to save the current widget values to the specified file in a form that can later be read by `xpbs` or by the `qsub` command.

2.62.9 Modifying Attributes of Jobs

Modifying a PBS job requires only to manipulate the widgets found in the Modify window. To modify a job or jobs, do the following steps:

1. Select one or more jobs from the JOBS listbox in the main `xpbs` display.
2. Click on the modify button located to the right of the listbox.
3. The Modify window is structured similarly to the Submit window. Simply manipulate the widgets to specify replacement or additional values of job attributes.
4. Click on the confirm modify button located at the bottom of the dialog box.

2.62.10 Deleting Jobs

Deleting a PBS job requires only to manipulate the widgets found in the Delete window. To delete a job or jobs, do the following steps:

1. Select one or more jobs from the JOBS listbox in the main `xpbs` display.
2. Click on the delete button located to the right of the listbox.
3. Manipulate the spinbox widget to set the kill delay signal interval.
4. Click on the delete button located at the bottom of the dialog box.

2.62.11 Tracking Returned Output Files

If you want to be informed of returned output files of current jobs, and be able to quickly see the contents of those files, then enable the “track job” feature as follows:

1. Submit all the jobs that you want monitored.
2. Click on the Track Job button located in the Menu bar to bring up the Track Job dialog box.
3. Specify the list of user names, whose jobs are to be monitored for returned output files, in the matrix located at the upper left of the dialog box.
4. Manipulate the minutes spinbox, located just below the user names matrix, to specify the interval value when output files will be periodically checked.
5. Specify the location of job output files (whether locally or remotely) by clicking on one of the radio buttons located at the upper right of the dialog box. Returned locally means the output files will be returned back to the host where `xpbs` was run. If the output files are returned to some remote host, then `xpbs` will execute an RSH

`<remote_host> test -f <output_files>` to test the existence of the files. RSH is whatever you set the remote shell command to in the corresponding entry box.

NOTE: Be sure the files are accessible from the host where `xpbs` was run (i.e. `.rhosts` appropriately set).

6. Click start/reset tracking button located at the bottom of the dialog box to:
 - cancel any previous tracking
 - build a new list of jobs to be monitored for returned output files based on currently queued jobs.
 - start periodic tracking.
7. Click on close window button.

When an output file for a job being monitored is found, then the Track Job button (the one that originally invoked the Track Job dialog box) will turn into a different color, and the Jobs Found Completed listbox, located in the Track Job dialog box, is then loaded with the corresponding job id(s). Then double click on a job id to see the contents of the output file and the error file. Click stop tracking if you want to cancel tracking.

2.62.12 Leaving `xpbs`

Click on the Close button located in the Menu bar to leave `xpbs`. If anything had changed, it will bring up a dialog box asking for a confirmation in regards to saving state information like the view states (minimize/maximize) of the HOSTS, QUEUES, JOBS, and INFO subwindows, and various criteria for listing queues and jobs. The information is saved in `~/ .xpbs-src` file.

2.62.13 Setting Preferences

The resources that can be set in the X resources file, `~/ .xpbsrc`, are:

***serverHosts**

list of server hosts (space separated) to query by `xpbs` keyword
PBS_DEFAULT_SERVER can be used which will be used as a placeholder for the value obtained from **defServerFile*.

***defServerFile**

the file containing the name of the default server host. The content of this will be substituted for the *PBS_DEFAULT_SERVER* keyword in **serverHosts* value.

- *timeoutSecs**
specify the number of seconds before timing out waiting for a connection to a PBS host.
- *xtermCmd**
the xterm command to run driving an interactive PBS session.
- *labelFont**
font applied to text appearing in labels.
- *fixlabelFont**
font applied to text that label fixed-width widgets such as listbox labels. This must be a fixed-width font.
- *textFont**
font applied to a text widget. Keep this as fixed-width font.
- *backgroundColor**
the color applied to background of frames, buttons, entries, scrollbar handles.
- *foregroundColor**
the color applied to text in any context (under selection, insertion, etc...).
- *activeColor**
the color applied to the background of a selection, a selected command button, or a selected scroll bar handle.
- *disabledColor**
color applied to a disabled widget.
- *signalColor**
color applied to buttons that signal something to the user about a change of state. For example, the color of the Track Job button when returned output files are detected.
- *shadingColor**
a color shading applied to some of the frames to emphasize focus as well as decoration.
- *selectorColor**
the color applied to the selector box of a radio button or check-button.
- *selectHosts**
list of hosts (space separated) to automatically select/highlight in the HOSTS listbox.

***selectQueues**

list of queues (space separated) to automatically select/highlight in the QUEUES listbox.

***selectJobs**

list of jobs (space separated) to automatically select/highlight in the JOBS listbox.

***selectOwners**

list of owners checked when limiting the jobs appearing on the Jobs listbox in the main `xpbs` window. Specify value as “*Owners: <list_of_owners>*”. See -u option in `qselect (1B)` for format of *<list_of_owners>*.

***selectStates**

list of job states to look for (do not space separate) when limiting the jobs appearing on the Jobs listbox in the main `xpbs` window. Specify value as “*Job_States: <states_string>*”. See -s option in `qselect (1B)` for format of *<states_string>*.

***selectRes**

list of resource amounts (space separated) to consult when limiting the jobs appearing on the Jobs listbox in the main `xpbs` window. Specify value as “*Resources: <res_string>*”. See -l option in `qselect (1B)` for format of *<res_string>*.

***selectExecTime**

the Execution Time attribute to consult when limiting the list of jobs appearing on the Jobs listbox in the main `xpbs` window. Specify value as “*Queue_Time: <exec_time>*”. See -a option in `qselect (1B)` for format of *<exec_time>*.

***selectAcctName**

the name of the account that will be checked when limiting the jobs appearing on the Jobs listbox in the main `xpbs` window. Specify value as “*Account_Name: <account_name>*”. See -A option in `qselect (1B)` for format of *<account_name>*.

***selectCheckpoint**

the checkpoint attribute relationship (including the logical operator) to consult when limiting the list of jobs appearing on the Jobs listbox in the main `xpbs` window. Specify value as “*Checkpoint: <checkpoint_arg>*”. See -c option in `qselect (1B)` for format of *<checkpoint_arg>*.

***selectHold**

the hold types string to look for in a job when limiting the jobs appearing on the Jobs listbox in the main `xpbs` window. Specify value as “*Hold_Types:*

<hold_string>”. See `-h` option in `qselect(1B)` for format of *<hold_string>*.

***selectPriority**

the priority relationship (including the logical operator) to consult when limiting the list of jobs appearing on the Jobs listbox in the main `xpbs` window. Specify value as “*Priority: <priority_value>*”. See `-p` option in `qselect(1B)` for format of *<priority_value>*.

***selectRerun**

the rerunnable attribute to consult when limiting the list of jobs appearing on the Jobs listbox in the main `xpbs` window. Specify value as “*Rerunable: <rerun_val>*”. See `-r` option in `qselect(1B)` for format of *<rerun_val>*.

***selectJobName**

name of the job that will be checked when limiting the jobs appearing on the Jobs listbox in the main `xpbs` window. Specify value as “*Job_Name: <jobname>*”. See `-N` option in `qselect(1B)` for format of *<jobname>*.

***iconizeHostsView**

a boolean value (*True* or *False*) indicating whether or not to iconize the HOSTS region.

***iconizeQueuesView**

a boolean value (*True* or *False*) indicating whether or not to iconize the QUEUES region.

***iconizeJobsView**

a boolean value (*True* or *False*) indicating whether or not to iconize the JOBS region.

***iconizeInfoView**

a boolean value (*True* or *False*) indicating whether or not to iconize the INFO region.

***jobResourceList**

a curly-braced list of resource names as according to architecture known to `xpbs`. The format is as follows:

```
{ <arch-type1> resname1 resname2 ... resnameN }
{ <arch-type2> resname1 resname2 ... resnameN }
. . .
{ <arch-typeN> resname1 resname2 ... resnameN }
```

2.62.14 xpbs and PBS Commands

xpbs calls PBS commands as follows:

Table 2-31: xpbs and PBS Commands

Command Button	PBS Command
detail (Hosts)	qstat -B -f <selected server_host(s)>
terminate	qterm <selected server_host(s)>
detail (Queues)	qstat -Q -f <selected queue(s)>
stop	qstop <selected queue(s)>
start	qstart <selected queue(s)>
enable	qenable <selected queue(s)>
disable	qdisable <selected queue(s)>
detail (Jobs)	qstat -f <selected job(s)>
modify	qalter <selected job(s)>
delete	qdel <selected job(s)>
hold	qhold <selected job(s)>
release	qrls <selected job(s)>
run	qrun <selected job(s)>
rerun	qrerun <selected job(s)>
signal	qsig <selected job(s)>
msg	qmsg <selected job(s)>
move	qmove <selected job(s)>
order	qorder <selected job(s)>

2.62.15 Exit Status

Upon successful processing, the `xpbs` exit status will be a value of zero.

If the `xpbs` command fails, the command exits with a value greater than zero.

2.62.16 See Also

The PBS Professional User's Guide, the PBS Professional Administrator's Guide, `qalter(1B)`, `qdel(1B)`, `qhold(1B)`, `qmove(1B)`, `qmsg(1B)`, `qre-run(1B)`, `qrls(1B)`, `qselect(1B)`, `qsig(1B)`, `qstat(1B)`, `qorder(1B)`, `qsub(1B)`, `qdisable(8B)`, `qenable(8B)`, `qrun(8B)`, `qstart(8B)`, `qstop(8B)`, `qterm(8B)`

2.63 xpbsmon

GUI for displaying, monitoring execution hosts under PBS

2.63.1 Synopsis

xpbsmon

xpbsmon --version

2.63.2 Description

The `xpbsmon` command provides a way to graphically display the various nodes that run jobs. A node or execution host can be running a `pbs_mom` daemon, or not running the daemon. For the latter case, it could just be a nodename that appears in a nodes file that is managed by a main `pbs_server` running on another host. This utility also provides the ability to monitor values of certain system resources by posting queries to the `pbs_mom` of a node. With this utility, you can see what job is running on what node, who owns the job, how many nodes assigned to a job, status of each node (color-coded and the colors are user-modifiable), how many nodes are available, free, down, reserved, offline, of unknown status, in use running multiple jobs or executing only 1 job. Please see the sections below for a tour and tutorials of `xpbsmon`. Also, within every dialog box, a Help button can be found for assistance.

2.63.3 Getting Started

Running `xpbsmon` will initialize the X resource database from various sources in the following order:

1. The `RESOURCE_MANAGER` property on the root window (updated via `xrdb`) with settings usually defined in the `.Xdefaults` file
2. Preference settings defined by the system administrator in the global `xpbsmonrc` file
3. User's `~/ .xpbsmonrc` file - this file defines various X resources like fonts, colors, list of colors to use to represent the various status of the nodes, list of PBS sites to query, list of server hosts on each site, list of nodes/execution hosts on each server host, list of system resource queries to send to the nodes' `pbs_mom`, and various view states. See [section 2.63.10, "Setting Preferences", on page 257](#) below for a list of resources that can be set.

2.63.4 Running xpbsmon

`xpbsmon` can be run either as a regular user or superuser. If you run it with less privilege, you may not be able to see all the information for a node. If it is executed as a regular user, you should still be able to see what jobs are running on what nodes, possibly state, as this information are obtained by `xpbsmon` talking directly to the specified server. If you want other system resource values, it may require special privilege since `xpbsmon` will have to talk directly to the `pbs_mom` of a node. In addition, the host where `xpbsmon` was running must also have been given explicit access permission by the MOM (unless the GUI is running on the same host where MOM is running). This is done done by updating the `$clienthost` and/or the `$restricted` parameter on the MOM's configuration file.

To run `xpbsmon`, type:

```
setenv DISPLAY <display_host>:0
xpbsmon
```

If you are running the GUI and only interested in jobs data, then be sure to set all the nodes' type to `NOMOM` in the Pref dialog box.

2.63.5 Options

`--version`

The `xpbsmon` command returns its PBS version information and exits. This option can only be used alone.

2.63.6 The **xpbsmon** Display

This section describes the main parts of the **xpbsmon** display. The main window is composed of 3 distinct areas (subwindows) arranged vertically (one on top of another) in the following order:

1. Menu
2. Site Information
3. Info

2.63.6.1 Menu

The Menu area is composed of a row of command buttons that signal some action with a click of the left mouse button. The buttons are:

Site..

displays a popup menu containing the list of PBS sites that have been added using the Sites Preferences window. Simply drag your mouse and release to the site name whose servers/nodes information you would like to see.

Pref..

brings up various dialog boxes for specifying the list of sites, servers on each site, nodes that are known to a server, and the system resource queries to be sent to a node's **pbs_mom** daemon.

Auto Update..

brings up another window for specifying whether or not to do auto updates of nodes information, and also for specifying the interval number of minutes between updates.

Help

contains some help information.

About

tells who the author is and who to send comments, bugs, suggestions to.

Close

for exiting **xpbsmon** plus saving the current setup information (if anything had changed) in the user's `$HOME/.xpbsmonrc` file. Information saved include the specified list of sites, servers on each site, nodes known to each server, and system resource queries to send to node's **pbs_mom**.

Minimize Button

shows the iconized view of Site Information where nodes are represented as tiny boxes, where each box is colored according to status. In order to get

more information about a node, you need to double click on the colored box.

Maximize button

shows the full view of Site Information where nodes are represented in bigger boxes, still colored depending on the status, and some information on it is displayed.

2.63.6.2 Site Information

Only one site at a time can be displayed. This area (shown as one huge box referred to as the site box) can be further subdivided into 3 areas: the site name label at the top, server boxes in the middle, and the color status bar at the bottom. The site name label shows the name of the site as specified in the Pref.. window. At the middle of the site box shows a row of big boxes housing smaller boxes.

The big box is an abstraction of a server host (called a server box), showing its server display label at the top of the box, a grid of smaller boxes representing the nodes that the server knows about (where jobs are run), and summary status for the nodes under the server. Status information will show counters for the number of nodes used, available, reserved, offline, or of unknown status and even # of cpus assigned. For a cleaner display, some counters with a value of zero are not displayed. The server boxes are placed in a grid, with a new row being started when either *siteBoxMaxNumServerBoxesPerRow or *siteBoxMaxWidth limit has been reached.

The smaller boxes represent the nodes/execution hosts where jobs are run (referred to as node boxes). Each node box shows the name at the top, and a sub-box (a smaller square) that is colored according to the status of the node that it represents, and if the view type is FULL, it will display some node information according to the system resource queries specified on the Pref.. window. Clicking on the sub-box will show a much bigger box (called the MIRROR view) with bigger fonts containing nodes information. Another view is called ICON and this shows a tiny box with a colored area. The node boxes are arranged in a grid, where a new row is created if either the *serverBoxMaxNumNodeBoxesPerRow or *serverBoxMaxWidth limit has been reached. ICON view of the node boxes will be constrained by the *nodeBoxIconMaxHeight and *nodeBoxIconMaxWidth pixel values; FULL view of the node boxes will be bounded by *nodeBoxFullMaxWidth and *nodeBoxFullMaxHeight; the mirror view of the node boxes has its size be *nodeBoxMirrorMaxWidth, and *nodeBoxMirrorMaxHeight.

Horizontal and vertical scrollbars for the site box, server box, and node box will be displayed as needed.

Finally, the color bar information shows a color chart displaying what the various colors mean in terms of node status. The color-to-status mapping can be modified by setting the X resources: `*nodeColorNOINFO`, `*nodeColorFREE`, `*nodeColorINUSEshared`, `*nodeColorINUSEexclusive`, `*nodeColorDOWN`, `*nodeColorRSVD`, `*nodeColorOFFL`, `*nodeColor-BUSY`.

2.63.6.3 Info

The Info Area shows the progress of some of the background actions performed by `xpbsmon`. Look into this box for errors.

2.63.7 Widgets Used in `xpbsmon`

Some of the widgets used in `xpbsmon` and how they are manipulated are described in the following:

2.63.7.1 `listbox`

the ones found in this GUI are only single-selectable (one entry can be highlighted/selected at a time via a mouse click).

2.63.7.2 `scrollbar`

usually appears either vertically or horizontally and contains 5 distinct areas that are mouse clicked to achieve different effects:

top arrow

Causes the view in the associated widget to shift up by one unit (i.e. the object appears to move down one unit in its window). If the button is held down the action will auto-repeat.

top gap

Causes the view in the associated window to shift up by one less than the number of units in the window (i.e. the portion of the object that used to appear at the very top of the window will now appear at the very bottom). If the button is held down the action will auto-repeat.

slider

Pressing button 1 in this area has no immediate effect except to cause the slider to appear sunken rather than raised. However, if the mouse is moved with the button down then the slider will be dragged, adjusting the view as the mouse is moved.

bottom gap

Causes the view in the associated window to shift down by one less than the number of units in the window (i.e. the portion of the object that used to appear at the very bottom of the window will now appear at the very top). If the button is held down the action will auto-repeat.

bottom arrow

Causes the view in the associated window to shift down by one unit (i.e. the object appears to move up one unit in its window). If the button is held down the action will auto-repeat.

2.63.7.3 entry

brought into focus with a click of the left mouse button. To manipulate this widget, simply type in the text value. Use of arrow keys, mouse selection of text for deletion or overwrite, copying and pasting with sole use of mouse buttons are permitted. This widget is usually accompanied by a scrollbar for horizontally scanning a long text entry string.

2.63.7.4 box

made up of 1 or more listboxes displayed adjacent to each other giving the effect of a “matrix”. Each row from the listboxes makes up an element of the box. In order to add items to the box, you need to manipulate the accompanying entry widgets, one for each listbox, and then clicking the add button. Removing items from the box is done by selecting an element, and then clicking delete.

2.63.7.5 spinbox

a combination of an entry widget and a horizontal scrollbar. The entry widget will only accept values that fall within a defined list of valid values, and incrementing through the valid values is done by clicking on the up/down arrows.

2.63.7.6 button

a rectangular region appearing either raised or pressed that invokes an action when clicked with the left mouse button. When the button appears pressed, then hitting the <RETURN> key will automatically select the button.

2.63.8 Updating Preferences

2.63.8.1 Time Sharing

Suppose you have a time-sharing environment where the front-end is called bower and you have 4 nodes: bower1, bower2, bower3, bower4. bower is the host that runs the server; jobs are submitted to host bower where it enqueues it for future execution. Also, a `pbs_mom` daemon is running on each of the execution hosts. If the server bower also maintains a nodes list containing information like state for the 4 nodes, then this will also be reported. Then to setup `xpbsmon`, do the following:

1. Click the Pref.. button on the Menu section.
2. On the Sites Preference dialog, enter any arbitrary site name, for example “Local”. Then click the add button.
3. On the Server_Host entry box, enter “bower”, and on the DisplayLabel entry box, put an arbitrary label (as it would appear on the header of the server box) like “Bower”, and then click add.
4. Click the nodes.. button that is accompanying the Servers box. This would bring up the Server Preference dialog.
5. Now add the entries “bower1”, “bower2”, “bower3”, “bower4” specifying type MOM for each on the Nodes box.
6. If you need to monitor certain system resource parameters for each of the nodes, you need to specify query expressions containing resource queries to be sent to the individual PBS moms. For example, if you want to obtain memory usage, then select a node from the Nodes list, click on the query.. button that accompanies the Nodes list, and this would bring up the Query Table dialog. Specify the following input:

Query_Expr: (availmem/totmem) * 100

Display_Info: Memory Usage:

Display_Type: SCALE

The above says to display the result of the “Query_Expr” in a scale widget calibrated over 100. The queries “availmem” and “totmem” will be sent to the PBS mom, and the

expression is evaluated upon receiving all results from the mom. If you want to display the result of another query, say “loadave”, directly, then specify the following:

Query_Expr: loadave

Display_Info: Load Average:

Display_Type: TEXT

NOTE: For a list of queries that can be sent to a pbs_mom, please click on the Help button on the Query table window.

2.63.8.2 Jobs Exclusive Environment

Supposing you have a “space non-sharing” environment where the server maintains a list of nodes that it runs jobs on exclusively (one job at a time outstanding per node). Let’s call this server b1. Simply update Preferences information as follows:

1. Click the Pref.. button on the Menu section.
2. On the Sites Preference dialog, enter a site name, for example “B System”. Then click the add button.
3. On the Server_Host entry box, enter “b1”, DisplayLabel entry box type “B1” (or whatever label that you would like to appear on the header of the server box), and then click add.

2.63.8.3 Hybrid Time Sharing/Space Sharing Environment

A cluster of heterogeneous machines, time-sharing or jobs exclusive, could easily be represented in xpbsmon by combining steps in CASE 1 and CASE 2.

2.63.9 Leaving xpbsmon

Click on the Close button located in the Menu bar to leave xpbsmon. If anything had changed, it will bring up a dialog box asking for a confirmation in regards to saving preferences information about list of sites, their view types, list of servers on each site, the list of nodes known to each server, and the list of queries to be sent to the pbs_mom of each node. The information is saved in ~/.xpbsmonrc file.

2.63.10 Setting Preferences

The resources that can be set in the X resources file, ~/.xpbsmonrc, are described in the following:

2.63.10.1 Node Box Properties

Resource names beginning with “*small” or “*node” apply to the properties of the node boxes. A node box is made of an outer frame where the node label sits on top, the canvas (smaller box) is on the middle, and possibly some horizontal/ vertical scrollbars.

nodeColorNOINFO

color of node box when information for the node it represents could not be obtained.

***nodeColorFREE**

color of canvas when node it represents is up.

***nodeColorINUSEshared**

color when node it represents has more than 1 job running on it, or when node has been marked by the server that manages it as “job-sharing”.

***nodeColorINUSEexclusive**

list of colors to assign to a node box when host it represents is running only 1 job, or when node has been marked by the server that manages it as “time-sharing”. `xpbsmon` will use this list to assign 1 distinct color per job unless all the colors have been exhausted, in which case, colors will start getting assigned more than once in a round-robin fashion.

***nodeColorDOWN**

color when node it represents is down.

***nodeColorRSVD**

color when node it represents is reserved.

***nodeColorOFFL**

color when node it represents is offline.

***nodeColorBUSY**

color when node it represents is busy (high load average).

***smallForeground**

applies to the color of text inside the canvas.

***smallBackground**

applies to the color of the frame.

***smallBorderWidth**

distance (in pixels) from other node boxes.

***smallRelief**

how node box will visually appear (style).

-
- *smallScrollBorderWidth**
significant only in FULL mode, this is the distance of the horizontal/vertical scrollbars from the canvas and lower edge of the frame.
 - *smallScrollBackground**
background color of the scrollbars
 - *smallScrollRelief**
how scrollbars would visually appear (style).
 - *smallCanvasBackground**
color of the canvas (later overridden depending on status of the node it represents)
 - *smallCanvasBorderWidth**
distance of the canvas from the frame and possibly the scrollbars.
 - *smallCanvasRelief**
how the canvas is visually represented (style).
 - *smallLabelBorderWidth**
the distance of the node label from the canvas and the topmost edge of the frame.
 - *smallLabelBackground**
the background of the area of the node label that is not filled.
 - *smallLabelRelief**
how the label would appear visually (style).
 - *smallLabelForeground**
the color of node label text.
 - *smallLabelFont**
the font to use for the node label text.
 - *smallLabelFontWidth**
font width (in pixels) of *smallLabelFont
 - *smallLabelFontHeight**
font height (in pixels) of *smallLabelFont
 - *smallTextFont**
font to use for the text that appear inside a canvas.
 - *smallTextFontWidth**
font width (in pixels) of *smallTextFont.
 - *smallTextFontHeight**
font height (in pixels) of *smallTextFont.

-
- *nodeColorTrough**
color of trough part (the /100 portion) of a canvas scale item.
 - *nodeColorSlider**
color of slider part (value portion) of a canvas scale item.
 - *nodeColorExtendedTrough**
color of extended trough (over 100 portion when value exceeds max) of a canvas scale item.
 - *nodeScaleFactor**
tells how much bigger you want the scale item on the canvas to appear. (1 means to keep size as is)
 - *nodeBoxFullMaxWidth**
 - *nodeBoxFullMaxHeight**
maximum width and height (in pixels) of a node box in FULL mode.
 - *nodeBoxIconMaxWidth**
 - *nodeBoxIconMaxHeight**
maximum width and height (in pixels) of a node box in ICON mode.
 - *nodeBoxMirrorMaxWidth**
 - *nodeBoxMirrorMaxHeight**
maximum width and height (in pixels) of a node box displayed on a separate window (after it has been clicked with the mouse to obtain a bigger view)
 - *nodeBoxMirrorScaleFactor**
tells how much bigger you want the scale item on the canvas to appear while the node box is displayed on a separate window (1 means to keep size as is)

2.63.10.2 Server Box Properties

Resource names beginning with “*medium” apply to the properties of the server boxes. A server box is made of an outer frame where the server display label sits on top, a canvas filled with node boxes is on the middle, possibly some horizontal/vertical scrollbars, and a status label at the bottom.

- *mediumLabelForeground**
color of text applied to the server display label and status label.
- *mediumLabelBackground**
background color of the unfilled portions of the server display label and status label.

-
- *mediumLabelBorderWidth**
distance of the server display label and status label from other parts of the server box.
 - *mediumLabelRelief**
how the server display label and status label appear visually (style).
 - *mediumLabelFont**
font used for the text of the server display label and status label.
 - *mediumLabelFontWidth**
font width (in pixels) of *mediumLabelFont.
 - *mediumLabelFontHeight**
font height (in pixels) of *mediumLabelFont.
 - *mediumCanvasBorderWidth**
the distance of the server box's canvas from the label widgets.
 - *mediumCanvasBackground**
the background color of the canvas.
 - *mediumCanvasRelief**
how the canvas appear visually (style).
 - *mediumScrollBorderWidth**
distance of the scrollbars from the other parts of the server box.
 - *mediumScrollBackground**
the background color of the scrollbars
 - *mediumScrollRelief**
how the scrollbars appear visually.
 - *mediumBackground**
the color of the server box frame.
 - *mediumBorderWidth**
the distance of the server box from other boxes.
 - *mediumRelief**
how the server box appears visually (style).
 - *serverBoxMaxWidth**
 - *serverBoxMaxHeight**
maximum width and height (in pixels) of a server box.
 - *serverBoxMaxNumNodeBoxesPerRow**
maximum # of node boxes to appear in a row within a canvas.

2.63.10.3 Miscellaneous Properties

Resource names beginning with “*big” apply to the properties of a site box, as well as to widgets found outside of the server box and node box. This includes the dialog boxes that appear when the menu buttons of the main window are manipulated. The site box is the one that appears on the main region of `xpbsmon`.

- *bigBackground**
background color of the outer layer of the main window.
- *bigForeground**
color applied to regular text that appear outside of the node box and server box.
- *bigBorderWidth**
distance of the site box from the menu area and the color information area.
- *bigRelief**
how the site box is visually represented (style)
- *bigActiveColor**
the color applied to the background of a selection, a selected command button, or a selected scroll bar handle.
- *bigShadingColor**
a color shading applied to some of the frames to emphasize focus as well as decoration.
- *bigSelectorColor**
the color applied to the selector box of a radiobutton or checkbutton.
- *bigDisabledColor**
color applied to a disabled widget.
- *bigLabelBackground**
color applied to the unfilled portions of label widgets.
- *bigLabelBorderWidth**
distance from other widgets of a label widget.
- *bigLabelRelief**
how label widgets appear visually (style)
- *bigLabelFont**
font to use for labels.
- *bigLabelFontWidth**
font width (in pixels) of *bigLabelFont.
- *bigLabelFontHeight**
font height (in pixels) of *bigLabelFont.

-
- *bigLabelForeground**
color applied to text that function as labels.
 - *bigCanvasBackground**
the color of the main region.
 - *bigCanvasRelief**
how the main region looks like visually (style)
 - *bigCanvasBorderWidth:**
distance of the main region from the menu and info regions.
 - *bigScrollBorderWidth**
if the main region has a scrollbar, this is its distance from other widgets appearing on the region.
 - *bigScrollBackground**
background color of the scrollbar appearing outside a server box and node box.
 - *bigScrollRelief**
how the scrollbar that appears outside a server box and node box looks like visually (style)
 - *bigTextFontWidth**
the font width (in pixels) of ***bigTextFont**
 - *bigTextFontHeight**
the font height (in pixels) of ***bigTextFont**
 - *siteBoxMaxWidth**
maximum width (in pixels) of the site box.
 - *siteBoxMaxHeight**
maximum height (in pixels) of the site box.
 - *siteBoxMaxNumServerBoxesPerRow**
maximum number of server boxes to appear in a row inside the site box.
 - *autoUpdate**
if set to *True*, then information about nodes is periodically gathered.
 - *autoUpdateMins**
the # of minutes between polling for data regarding nodes when ***autoUpdate** is set.
 - *siteInView**
the name of the site that should be in view
 - *rcSiteInfoDelimiterChar**
the separator character for each input within a curly-bracketed line of input of ***siteInfo**.

***sitesInfo**

```
{<site1name><sep><site1-display-type><sep> <server-name><sep>
<server-display-label><sep><nodename><sep> <nodetype><sep>
<node-query-expr>}
. . .
{<site2name><sep><site2-display-type> <sep><server-name><sep>
<server-display-label><sep><nodename> <sep><nodetype><sep>
<node-query-expr>}
```

Information about a site where *<site1-display-type>* can be either *{FULL,ICON}*, *<nodetype>* can be *{MOM, NOMOM}*, and *<nodequery-expr>* has the format:

```
{ {<expr>} {expr-label} <output-format>}
```

where *<output-format>* could be *{TEXT, SCALE}*. It's probably better to use the Pref dialog boxes in order to specify a value for this.

Example:

```
*rcSiteInfoDelimiterChar ;
*sitesInfo: {NAS;ICON;newton;Newton; newton3; NOMOM;} {Langley;
FULL; db;DB; db.nas.nasa.gov; MOM; {{ ( availmem / totmem ) *
100} {Memory Usage:} SCALE} {{ ( loadave / ncpus ) * 100}
{Cpu Usage:} SCALE} {ncpus {Number of Cpus:} TEXT} {physmem
{Physical Memory:} TEXT} {idletime {Idle Time (s):} TEXT}
{loadave {Load Avg:} TEXT}} {NAS;ICON;newton;Newton;newton4;
NOMOM;} {NAS;ICON;newton; Newton; newton1;NOMOM;} {NAS;
ICON;newton;Newton; newton2;NOMOM;} {NAS;ICON;b0101;DB;aspa-
sia.nas.nasa.gov; MOM;{{ ( availmem / totmem ) * 100} {Memory
Usage:} SCALE} {{ ( loadave / ncpus ) * 100} {Cpu Usage:}
SCALE} {ncpus {Number of Cpus:} TEXT} {physmem {Physical Mem-
ory:} TEXT} {idletime {Idle Time (s):} TEXT} {loadave {Load
Avg:} TEXT}} {NAS;ICON;newton;Newton;newton7;NOMOM;}
```

2.63.11 Exit Status

Upon successful processing, the `xpbsmon` exit status will be a value of zero.

If the `xpbsmon` command fails, the command exits with a value greater than zero.

If `xpbsmon` is querying a host running a server with an incompatible version, you may see the following messages:

```
Internal error: pbsstatnode: End of File (15031)
```

The above message can be safely ignored.

2.63.12 See Also

The PBS Professional User's Guide, the PBS Professional Administrator's Guide,
`pbs_sched(8B)`, `pbs_mom(8B)`, `pbs_tclapi(3B)`

Chapter 3

MOM Parameters

This chapter describes the configuration files used by MOM and lists the MOM configuration parameters that are found in the Version 1 MOM configuration file, `PBS_HOME/mom_priv/config`.

3.1 Syntax of MOM Configuration File

The Version 1 MOM configuration file contains parameter settings for the MOM on the local host.

Version 1 configuration files list local resources and initialization values for MOM. Local resources are either static, listed by name and value, or externally-provided, listed by name and command path. Local static resources are for use only by the scheduler. They do not appear in a `pbsnodes -a` query. See the `-c` option to the `pbs_mom` command. Do not change the syntax of the Version 1 configuration file.

Each configuration item is listed on a single line, with its parts separated by white space. Comments begin with a hashmark ("`#`").

3.1.1 Externally-provided Resources

Externally-provided resources, for example dynamic resources such as scratch space, use a shell escape to run a command. These resources are described with a name and value, where the first character of the value is an exclamation mark ("`!`"). The remainder of the value is the path and command to execute.

Parameters in the command beginning with a percent sign ("%") can be replaced when the command is executed. For example, this line in a configuration file describes a resource named "escape":

```
escape    !echo %xxx %yyy
```

If a query for the "escape" resource is sent with no parameter replacements, the command executed is "echo %xxx %yyy". If one parameter replacement is sent, "escape[xxx=hi there]", the command executed is "echo hi there %yyy". If two parameter replacements are sent, "escape[xxx=hi][yyy=there]", the command executed is "echo hi there". If a parameter replacement is sent with no matching token in the command line, "escape[zzz=snafu]", an error is reported.

3.1.2 Windows Notes

If the argument to a MOM option is a pathname containing a space, enclose it in double quotes as in the following:

```
hostn !"Program Files\PBS Pro\exec\bin\hostn" host
```

When you edit any PBS configuration file, make sure that you put a newline at the end of the file. The Notepad application does not automatically add a newline at the end of a file; you must explicitly add the newline.

3.2 Contents of MOM Configuration File

Initialization value directives have names beginning with a dollar sign ("\$"). They are listed here:

```
$action <default_action> <timeout> <new_action>
```

Replaces the *default_action* for an event with the site-specified *new_action*. *timeout* is the time allowed for *new_action* to run. *new_action* is the site-

supplied script that replaces *default_action*. This is the complete list of values for *default_action*:

Table 3-1: How *\$action* is Used

default_action	Result
checkpoint	Run <i>new_action</i> in place of the periodic job checkpoint, after which the job continues to run.
checkpoint_abort	Run <i>new_action</i> to checkpoint the job, after which the job must be terminated by the script.
multinodebusy <timeout> <i>requeue</i>	Used with cycle harvesting and multi-vnode jobs. Changes default behavior when a vnode becomes busy. Instead of allowing the job to run, the job is requeued. Timeout is ignored. The only <i>new_action</i> is <i>requeue</i> .
restart	Runs <i>new_action</i> in place of restart.
terminate	Runs <i>new_action</i> in place of SIGTERM or SIGKILL when MOM terminates a job.

***\$aix_largepagemode* <value>**

Controls whether large page mode is available for PBS jobs on AIX. Settable by root only.

Format: Boolean.

Default: *False*.

***\$alps_client* <path>**

Cray only. Path to the Cray *apbasil* command. Must be full path to command.

Format: path to command

Default: None

***\$alps_release_timeout* <timeout>**

Cray only. Specifies the amount of time that PBS tries to release an ALPS reservation before giving up. After this amount of time has passed, PBS stops trying to release the ALPS reservation, the job exits, and the job's resources are released. PBS sends a HUP to the MoM so that she rereads the ALPS inventory to get the current available ALPS resources.

We recommend that the value for this parameter be twice the value for `suspectbegin`.

Format: Seconds, specified as positive integer.

Default: *600* (10 minutes)

\$checkpoint_path <path>

MOM passes this parameter to the checkpoint and restart scripts. This path can be absolute or relative to `PBS_HOME/mom_priv`. Overrides default. Overridden by path specified in the `pbs_mom -C` option and by `PBS_CHECKPOINT_PATH` environment variable. Not supported in the HPCBP MOM. See [section 10.3.6.5, "Specifying Checkpoint Path" on page 720 in the PBS Professional Administrator's Guide](#).

\$clienthost <hostname>

`hostname` is added to the list of hosts which will be allowed to connect to MOM as long as they are using a privileged port. For example, this will allow the hosts "fred" and "wilma" to connect to MOM:

```
$clienthost fred
```

```
$clienthost wilma
```

The following hostnames are added to `$clienthost` automatically: the server, the localhost, and if configured, the secondary server. The server sends each MOM a list of the hosts in the nodes file, and these are added internally to `$clienthost`. None of these hostnames need to be listed in the configuration file.

Two hostnames are always allowed to connect to `pbs_mom`, "localhost" and the name returned to MOM by the system call `gethostname()`. These hostnames do not need to be listed in the configuration file.

The hosts listed as "clienthosts" make up a "sisterhood" of machines. Any one of the sisterhood will accept connections from within the sisterhood. The sisterhood must all use the same port number.

\$cpuset_error_action

When using a cpuset-enabled MOM, specifies the action taken when a cpuset creation error occurs. Can take one of the following values:

continue

The error is logged and the job is killed and requeued.

offline

The vnodes on this host for this job are marked *offline*, and the job is requeued.

Format: *String*

Allowable values: *continue*, *offline*

Default: *offline*

\$cputmult <factor>

This sets a factor used to adjust CPU time used by each job. This allows adjustment of time charged and limits enforced where jobs run on a system with different CPU performance. If MOM's system is faster than the reference system, set factor to a decimal value greater than 1.0. For example:

```
$cputmult 1.5
```

If MOM's system is slower, set factor to a value between 1.0 and 0.0. For example:

```
$cputmult 0.75
```

\$dce_refresh_delta <delta>

Defines the number of seconds between successive refreshings of a job's DCE login context. For example:

```
$dce_refresh_delta 18000
```

Not supported in the HPCBP MOM.

\$enforce <limit>

MOM will enforce the given limit. Some limits have associated values, and appear in the configuration file like this:

```
$enforce variable_name value
```

See The PBS Professional Administrator's Guide. Not supported in the HPCBP MOM.

\$enforce mem

MOM will enforce each job's memory limit.

\$enforce cpuaverage

MOM will enforce ncpus when the average CPU usage over a job's lifetime usage is greater than the job's limit.

\$enforce average_trialperiod <seconds>

Modifies *cpuaverage*. Minimum number of seconds of job walltime before enforcement begins.

Format: Integer

Default: 120

\$enforce average_percent_over <percentage>

Modifies *cpuaverage*. Gives percentage by which a job may exceed its ncpus limit.

Format: Integer

Default: *50*

\$enforce average_cpu*factor* <*factor*>

Modifies *cpuaverage*. The *ncpus* limit is multiplied by *factor* to produce actual limit.

Format: Float

Default: *1.025*

\$enforce cpuburst

MOM will enforce the *ncpus* limit when CPU burst usage exceeds the job's limit.

\$enforce delta_percent_over <*percentage*>

Modifies *cpuburst*. Gives percentage over limit to be allowed.

Format: Integer

Default: *50*

\$enforce delta_cpu*factor* <*factor*>

Modifies *cpuburst*. The *ncpus* limit is multiplied by *factor* to produce actual limit.

Format: Float

Default: *1.5*

\$enforce delta_weightup <*factor*>

Modifies *cpuburst*. Weighting factor for smoothing burst usage when average is increasing.

Format: Float

Default: *0.4*

\$enforce delta_weightdown <*factor*>

Modifies *cpuburst*. Weighting factor for smoothing burst usage when average is decreasing.

Format: Float

Default: *0.4*

\$ideal_load <*load*>

Defines the load below which the *vnode* is not considered to be busy. Used with the *\$max_load* directive. Not supported in the HPCBP MOM.

Example:

\$ideal_load 1.8

Format: Float

No default

`$jobdir_root <stage_directory_root>`

Directory under which PBS creates job-specific staging and execution directories. PBS creates a job's staging and execution directory when the job's `sandbox` attribute is set to *PRIVATE*. If `$jobdir_root` is unset, it defaults to the job owner's home directory. In this case the user's home directory must exist. If `stage_directory_root` does not exist when MOM starts up, MOM will abort. If `stage_directory_root` does not exist when MOM tries to run a job, MOM will kill the job. Path must be owned by root, and permissions must be *1777*. On Windows, this directory should have *Full Control Permission* for the local Administrators group.

Example:

```
$jobdir_root /scratch/foo
```

`$kbd_idle <idle_wait> <min_use> <poll_interval>`

Declares that the vnode will be used for batch jobs during periods when the keyboard and mouse are not in use. Not supported in the HPCBP MOM.

idle_wait

Time, in seconds, that the workstation keyboard and mouse must be idle before being considered available for batch jobs.

Must be set to non-zero value for cycle harvesting to be enabled.

Format: Integer

No default

min_use

Time, in seconds, during which the workstation keyboard or mouse must continue to be in use before the workstation is determined to be unavailable for batch jobs.

Format: Integer

Default: *10*

poll_interval

Interval, in seconds, at which MOM checks for keyboard and mouse activity.

Format: Integer

Default: *1*

Example:

```
$kbd_idle 1800 10 5
```

\$logevent <mask>

Sets the mask that determines which event types are logged by pbs_mom. To include all debug events, use *0xffffffff*.

Log events:

Table 3-2: Event Classes

Name	Hex Value	Message Category
ERROR	<i>0001</i>	Internal errors
SYSTEM	<i>0002</i>	System errors
ADMIN	<i>0004</i>	Administrative events
JOB	<i>0008</i>	Job-related events
JOB_USAGE	<i>0010</i>	Job accounting info
SECURITY	<i>0020</i>	Security violations
SCHED	<i>0040</i>	Scheduler events
DEBUG	<i>0080</i>	Common debug messages
DEBUG2	<i>0100</i>	Uncommon debug messages
RESV	<i>0200</i>	Reservation-related info
DEBUG3	<i>0400</i>	Rare debug messages
DEBUG4	<i>0800</i>	Limit-related messages

\$max_check_poll <seconds>

Maximum time between polling cycles, in seconds. See [section 3.6.1, "Configuring MOM's Polling Cycle" on page 53 in the PBS Professional Administrator's Guide](#). Minimum recommended value: 30 seconds.

Minimum value: *1 second*

Default: *120 seconds*

Format: Integer

\$max_load <load> [suspend]

Defines the load above which the vnode is considered to be *busy*. Used with the \$ideal_load directive. No new jobs are started on a *busy* vnode.

The optional suspend directive tells PBS to suspend jobs running on the node if the load average exceeds the `max_load` number, regardless of the source of the load (PBS and/or logged-in users). Without this directive, PBS will not suspend jobs due to load.

We recommend setting this to a slightly higher value than the number of CPUs, for example `.25 + ncpus`.

Not supported in the HPCBP MOM.

Example:

```
$max_load 3.5
```

Format: Float

Default: number of CPUs on machine

\$min_check_poll <seconds>

Minimum time between polling cycles, in seconds. Must be greater than zero and less than `$max_check_poll`. See [section 3.6.1, "Configuring MOM's Polling Cycle" on page 53 in the PBS Professional Administrator's Guide](#). Minimum recommended value: 10 seconds.

Minimum value: *1 second*

Default: *10 seconds*

Format: Integer

\$prologalarm <timeout>

Defines the maximum number of seconds the prologue and epilogue may run before timing out.

Example:

```
$prologalarm 30
```

Format: Integer

Default: *30*

\$reject_root_scripts <True|False>

When set to *True*, MoM won't acquire any new hook scripts, and MoM won't run job scripts that would execute as root or Admin.

Format: *Boolean*

Default: *False*

\$restart_background <value>

Controls how MOM runs a restart script after checkpointing a job. When this option is set to *True*, MOM forks a child which runs the restart script. The child returns when all restarts for all the local tasks of the job are done.

MOM does not block on the restart. When this option is set to *False*, MOM runs the restart script and waits for the result. Not supported in the HPCBP MOM.

Format: Boolean

Default: *False*

\$restart_transmogrify <value>

Controls how MOM runs a restart script after checkpointing a job.

When this option is set to *True*, MOM runs the restart script, replacing the session ID of the original task's top process with the session ID of the script.

When this option is set to *False*, MOM runs the restart script and waits for the result. The restart script must restore the original session ID for all the processes of each task so that MOM can continue to track the job.

When this option is set to *False* and the restart uses an external command, the configuration parameter `restart_background` is ignored and treated as if it were set to *True*, preventing MOM from blocking on the restart.

Not supported in the HPCBP MOM.

Format: Boolean

Default: *False*

\$restrict_user <value>

Controls whether users not submitting jobs have access to this machine. If value is *True*, restrictions are applied.

See `$restrict_user_exceptions` and `$restrict_user_maxsysid`.

Not supported on Windows.

Format: Boolean

Default: *False*

\$restrict_user_exceptions <user_list>

Comma-separated list of users who are exempt from access restrictions applied by `$restrict_user`. Leading spaces within each entry are allowed. Maximum of 10 names.

\$restrict_user_maxsysid <value>

Any user with a numeric user ID less than or equal to value is exempt from restrictions applied by `$restrict_user`.

If `$restrict_user` is on and no value exists for `$restrict_user_maxsysid`, PBS looks in `/etc/login.defs`, if it exists, for the value. Otherwise the default is used.

Format: Integer

Default: 999

\$restricted <hostname>

The hostname is added to the list of hosts which will be allowed to connect to MOM without being required to use a privileged port. Queries from the hosts in the restricted list are only allowed access to information internal to this host, such as load average, memory available, etc. They may not run shell commands.

Hostnames can be wildcarded. For example, to allow queries from any host from the domain “xyz.com”:

```
$restricted *.xyz.com
```

\$suspendsig <suspend_signal> [resume_signal]

Alternate signal `suspend_signal` is used to suspend jobs instead of SIG-STOP. Optional `resume_signal` is used to resume jobs instead of SIG-CONT. Not supported in the HPCBP MOM.

\$tmpdir <directory>

Location where each job’s scratch directory will be created.

PBS creates a temporary directory for use by the job, not by PBS. PBS creates the directory before the job is run and removes the directory and its contents when the job is finished. It is scratch space for use by the job. Permission must be 1777 on UNIX/Linux, writable by Everyone on Windows.

Not supported in the HPCBP MOM.

Example:

```
$tmpdir /memfs
```

Default on UNIX: /tmp.

Default on Windows: value of the TMP environment variable.

\$usecp <hostname:source_prefix> <destination_prefix>

MOM will use `/bin/cp` to deliver output files when the destination is a network mounted file system, or when the source and destination are both on the local host, or when the `source_prefix` can be replaced with the `destination_prefix` on hostname. Both `source_prefix` and `destination_prefix` are absolute pathnames of directories, not files.

Overrides PBS_RCP and PBS_SCP.

Not supported in the HPCBP MOM.

Use trailing slashes on both the source and destination. For example:

```
$usecp HostA:/users/work/myproj/ /sharedwork/proj_results/
```

\$vnodedef_additive

Specifies whether MOM considers a vnode that appeared previously either in the inventory or in a vnode definition file, but that does not appear now, to be in her list of vnodes.

When `$vnodedef_additive` is *True*, MOM treats missing vnodes as if they are still present, and continues to report them as if they are present. This means that the server does not mark missing vnodes as *stale*.

When `$vnodedef_additive` is *False*, MOM does not list missing vnodes, the server's information is brought up to date with the inventory and vnode definition files, and the server marks missing vnodes as *stale*.

Visible in configuration file on Cray only.

Format: *Boolean*

Default for MOM on Cray login node: *False*

\$wallmult <factor>

Each job's `walltime` usage is multiplied by this factor. For example:

```
$wallmult 1.5
```

3.2.1 Cray-only Initialization Values**pbs_accounting_workload_mgmt <value>**

Controls whether CSA accounting is enabled. Name does not start with dollar sign. If set to “1”, “on”, or “true”, CSA accounting is enabled. If set to “0”, “off”, or “false”, accounting is disabled.

Default: “true”; enabled.

3.2.2 SGI-only Initialization Values**cpuset_create_flags <flags>**

Lists the flags for when MOM does a `cpusetCreate(3)` for each job. `flags` is an or-ed list of flags. The flags are:

Altix, supported versions of ProPack, Performance Suite

```
CPUSET_CPU_EXCLUSIVE|0
```

Default: *CPUSET_CPU_EXCLUSIVE*

ICE, supported versions of ProPack, Performance Suite

```
CPUSET_CPU_EXCLUSIVE|0
```

Default: *0*

cpuset_destroy_delay <delay>

MOM will wait **delay** seconds before issuing a `cpusetDestroy(3)` on the `cpuset` of a just-completed job. This allows processes time to finish.

Example:

```
cpuset_destroy_delay 10
```

Format: Integer

Default for Altix: *0*

memreserved <megabytes>

Deprecated. The amount of per-vnode memory reserved for system overhead. This much memory is deducted from the value of `resources_available.mem` for each vnode managed by this MOM.

For example,

```
memreserved 16
```

Default: *0MB*

3.2.3 Static MOM Resources

Static resources local to the vnode are described one resource to a line, with a name and value separated by white space. For example, tape drives of different types could be specified by:

```
tape3480 4
```

```
tape3420 2
```

```
tapedat 1
```

```
tape8mm 1
```


Chapter 4

Scheduler Parameters

This chapter lists the scheduler's configuration parameters. These parameters are found in the scheduler's configuration file, `PBS_HOME/sched_priv/sched_config`.

4.1 Format of Scheduler's Configuration File

4.1.1 Parameters with Separate Primetime and Non-primetime Specification

If a scheduler parameter can be specified separately for primetime and non-primetime, the format for the parameter is the following:

name: value [prime | non_prime | all | none]

- The *name* field cannot contain any whitespace.
- The *value* field may contain whitespace if the string is double-quoted. *value* can be: *True* | *False* | <number> | <string>. "*True*" and "*False*" are not case-sensitive.
- The third field allows you to specify that the setting is to apply during primetime, non-primetime, all the time, or none of the time. A blank third field is equivalent to "*all*" which means that it applies to both prime- and non-primetime.

Acceptable values: "*all*", "*ALL*", "*none*", "*NONE*", "*prime*", "*PRIME*", "*non_prime*", "*NON_PRIME*"

4.1.2 Parameters without Separate Primetime and Non-primetime Specification

If a scheduler parameter cannot be specified separately for primetime and non-primetime, the format for the parameter is the same as the above, except that there is no third field.

4.1.3 Format Details

- Each entry must be a single, unbroken line.
- Entries must be quoted if they contain whitespace.
- Any line starting with a “#” is a comment, and is ignored.

4.1.4 Editing Configuration Files Under Windows

When you edit any PBS configuration file, make sure that you put a newline at the end of the file. The Notepad application does not automatically add a newline at the end of a file; you must explicitly add the newline.

4.2 Configuration Parameters

backfill

Toggle that controls whether PBS uses backfilling. If this is set to *True*, the scheduler attempts to schedule smaller jobs around higher-priority jobs when using *strict_ordering*, as long as running the smaller jobs won't change the start time of the jobs they were scheduled around. The scheduler chooses jobs in the standard order, so other high-priority jobs will be considered first in the set to fit around the highest-priority job.

When this parameter is *True*, the scheduler backfills around starving jobs when *help_starving_jobs* is *True*.

Can be used with *strict_ordering* and *help_starving_jobs*

Format: Boolean

Default: *True all*

backfill_prime

The Scheduler will not run jobs which would overlap the boundary between primetime and non-primetime. This assures that jobs restricted to running in either primetime or non-primetime can start as soon as the time boundary happens.

See also `prime_spill`, `prime_exempt_anytime_queues`.

Format: Boolean

Default: *False all*

`by_queue`

If set to *True*, all jobs that can be run from the highest-priority queue are run, then any jobs that can be run from the next queue are run, and so on. If `sort_queues` is set to *True*, queues are ordered highest-priority first. If `by_queue` is set to *False*, all jobs are treated as if they are in one large queue. The `by_queue` parameter is overridden by the `round_robin` parameter when `round_robin` is set to *True*.

See [section 4.8.4, "Examining Jobs Queue by Queue" on page 124 in the PBS Professional Administrator's Guide](#).

Format: Boolean

Default: *True all*

`cpus_per_ssinode`

Deprecated. Such configuration now occurs automatically.

`dedicated_prefix`

Queue names with this prefix are treated as dedicated queues, meaning jobs in that queue will only be considered for execution if the system is in dedicated time as specified in the configuration file `PBS_HOME/sched_priv/dedicated_time`.

See [section 4.8.10, "Dedicated Time" on page 148 in the PBS Professional Administrator's Guide](#).

Format: String

Default: *ded*

`fair_share`

Enables the fairshare algorithm, and turns on usage collecting. Jobs will be selected based on a function of their recent usage and priority (shares).

See [section 4.8.18, "Using Fairshare" on page 165 in the PBS Professional Administrator's Guide](#).

Format: Boolean

Default: *False all*

`fairshare_entity`

Specifies the entity for which fairshare usage data will be collected. Can be one of *"euser"*, *"egroup"*, *"Account_Name"*, *"queue"*, or *"egroup:euser"*.

Format: String

Default: *euser*

fairshare_enforce_no_shares

If this option is enabled, jobs whose entity has zero shares will never run. Requires **fair_share** to be enabled.

Format: Boolean

Default: *False*

fairshare_usage_res

Specifies the resource to collect and use in fairshare calculations and can be any valid PBS resource, including user-defined resources.

A special case resource is the exact string “*ncpus*walltime*”. The number of CPUs used is multiplied by the walltime in seconds used by the job to determine the usage.

See [section 4.8.18.5, "Tracking Resource Usage" on page 170 in the PBS Professional Administrator's Guide](#).

Format: String

Default: *cpus*

half_life

The half-life for fairshare usage; after the amount of time specified, the fairshare usage is halved. Requires that **fair_share** be enabled.

See [section 4.8.18, "Using Fairshare" on page 165 in the PBS Professional Administrator's Guide](#).

Format: Duration

Default: *24:00:00*

help_starving_jobs

Setting this option enables starving job support. Once jobs have waited for the amount of time given by **max_starve** they are considered starving. If a job is considered starving, then no lower-priority jobs will run until the starving job can be run, unless backfilling is also specified. To use this option, the **max_starve** configuration parameter needs to be set as well. See also **backfill**, **max_starve**, and the server's **eligible_time_enable** attribute.

At each scheduler iteration, PBS calculates **estimated.start_time** and **estimated.exec_vnode** for starving jobs being backfilled around.

Format: Boolean

Default: *True all*

job_sort_key

Selects how jobs should be sorted. `job_sort_key` can be used to sort using either (a) resources or (b) special case sorting routines. Multiple `job_sort_key` entries can be used, one to a line, in which case the first entry will be the primary sort key, the second will be used to sort equivalent items from the first sort, etc. This attribute is overridden by the `job_sort_formula` attribute. If both are set, `job_sort_key` is ignored and an error message is printed.

Syntax:

job_sort_key: "PBS_resource HIGH|LOW"

job_sort_key: "fair_share_perc HIGH|LOW"

job_sort_key: "job_priority HIGH|LOW"

job_sort_key: "preempt_priority HIGH|LOW"

Options: One of the following is required.

HIGH

Specifies descending sort.

LOW

Specifies ascending sort.

There are three special case sorting routines, which can be used instead of a specific PBS resource:

Table 4-1: Special Sorting in `job_sort_key`

Special Sort	Description
<i>fair_share_perc</i> <i>HIGH</i>	Sort based on the values in the <code>resource_group</code> file. If user A has more priority than user B, all of user A's jobs will always be run first. Past history is not used. This should only be used if entity share (strict priority) sorting is needed. Do not enable <code>fair_share_perc</code> sorting if using the <code>fair_share</code> scheduling option. (This option was previously named “ <code>fair_share</code> ” in the deprecated <code>sort_by</code> parameter). See section 4.8.14, "Sorting Jobs by Entity Shares (Was Strict Priority)" on page 155 in the <i>PBS Professional Administrator's Guide</i>
<i>job_priority</i> <i>HIGH LOW</i>	Sort jobs by the job priority attribute regardless of job owner.
<i>preempt_priority</i> <i>HIGH</i>	Sort jobs by preemption priority. Recommended that this be used when soft user limits are used. Also recommended that this be the primary sort key.
<i>sort_priority</i> <i>HIGH LOW</i>	Deprecated. See <code>job_priority</code> above.

The following example shows how to sort jobs so that those with high CPU count come first:

```
job_sort_key: "ncpus HIGH" all
```

The following example shows how to sort jobs so that those with lower memory come first:

```
job_sort_key: "mem LOW" prime
```

Format: Quoted string

Default: Not in force

key

Deprecated. Use `job_sort_key`.

load_balancing

When set to *True*, the scheduler takes into account the load average on vnodes as well as the resources listed in the `resources` line in `sched_config`. Load balancing can result in overloaded CPUs.

See [section 4.8.27, "Using Load Balancing" on page 187 in the PBS Professional Administrator's Guide](#).

Format: Boolean

Default: *False all*

load_balancing_rr

Deprecated. To duplicate this setting, enable `load_balancing` and set `smp_cluster_dist` to `round_robin`.

See [section 4.8.27, "Using Load Balancing" on page 187 in the PBS Professional Administrator's Guide](#).

log_filter

Defines which event types to keep out of the scheduler's logfile. The value should be set to the bitwise **OR** of the event classes which should be filtered. A value of 0 specifies maximum logging.

See [section 13.4.4.1.iii, "Specifying Scheduler Log Events" on page 860 in the PBS Professional Administrator's Guide](#).

Format: Integer

Default: 3328

max_starve

The amount of time before a job is considered starving. This variable is used only if `help_starving_jobs` is set.

Upper limit: None

Format: Duration

Default: 24:00:00

mem_per_ssinode

Deprecated. Such configuration now occurs automatically.

mom_resources

This option is used to query the MOMs to set the value of `resources_available.RES` where RES is a site-defined resource. Each MOM is queried with the resource name and the return value is used to replace `resources_available.RES` on that vnode. On a multi-vnoded

machine with a natural vnode, all vnodes will share anything set in `mom_resources`.

Format: String

Default: Unset

`node_sort_key`

Defines sorting on resource or priority values on vnodes. Resource must be numerical, for example, *long* or *float*. Up to 20 `node_sort_key` entries can be used, in which case the first entry will be the primary sort key, the second will be used to sort equivalent items from the first sort, etc.

Syntax:

`node_sort_key: <resource>|sort_priority HIGH|LOW`

`node_sort_key: <resource> HIGH|LOW
total|assigned|unused`

total

Use the `resources_available` value. This is the default setting when sorting on a resource.

assigned

Use the `resources_assigned` value.

unused

Use the value given by `resources_available` - `resources_assigned`.

sort_priority

Sort vnodes by the value of the vnode priority attribute.

See [section 4.8.48, "Sorting Vnodes on a Key" on page 277 in the PBS Professional Administrator's Guide](#).

Format: String

Default: `node_sort_key: sort_priority HIGH all`

`nonprimetime_prefix`

Queue names which start with this prefix will be treated as non-primetime queues. Jobs within these queues will only run during non-primetime. Primetime and non-primetime are defined in the `holidays` file.

See [section 4.8.34, "Using Primetime and Holidays" on page 235 in the PBS Professional Administrator's Guide](#).

Format: String

Default: `np_`

peer_queue

Defines the mapping of a remote queue to a local queue for Peer Scheduling. Maximum number is 50 peer queues per scheduler.

See [section 4.8.31, "Peer Scheduling" on page 199 in the PBS Professional Administrator's Guide](#).

Format: String

Default: unset

preemptive_sched

Enables job preemption.

See `preempt_order` and [section 4.8.33, "Using Preemption" on page 222 in the PBS Professional Administrator's Guide](#) for details.

Format: String

Default: *True all*

preempt_checkpoint

Deprecated. Add "C" to `preempt_order` parameter.

preempt_fairshare

Deprecated. Add "fairshare" to `preempt_prio` parameter.

preempt_order

Defines the order of preemption methods which the Scheduler will use on jobs. This order can change depending on the percentage of time remaining on the job. The ordering can be any combination of *S* *C* and *R*:

Table 4-2: Preemption Order Symbols

Symbol	Action
S	suspend
C	checkpoint
R	Requeue

Usage: an ordering (*SCR*) optionally followed by a percentage of time remaining and another ordering.

Must be a quoted list("").

Example:

`preempt_order: "SR"`

This example specifies that PBS should first attempt to use suspension to preempt a job, and if that is unsuccessful, then requeue the job.

Example:

```
preempt_order: "SCR 80 SC 50 S"
```

This example says if the job has between 100-81% of requested time remaining, first try to suspend the job, then try checkpoint then requeue. If the job has between 80-51% of requested time remaining, then attempt suspend then checkpoint; and between 50% and 0% time remaining just attempt to suspend the job.

Format: Quoted list

Default: *SCR*

preempt_prio

Specifies the ordering of priority for different preemption levels. Two or more job types may be combined at the same priority level with a plus sign (“+”) between them, using no whitespace. Comma-separated preemption levels are evaluated left to right, with higher priority to the left. The table below lists the six preemption levels. Note that any level not specified in the `preempt_prio` list is ignored.

Table 4-3: Preemption Levels

Level	Description
<code>express_queue</code>	Jobs in the express queues preempt other jobs. See <code>preempt_queue_prio</code> . Does not require <code>by_queue</code> or <code>sort_queues</code> to be <i>True</i> .
<code>starving_jobs</code>	When a job becomes starving it can preempt other jobs.
<code>fairshare</code>	When the entity owning a job exceeds its fair-share limit.
<code>queue_softlimits</code>	Jobs which are over their queue soft limits
<code>server_softlimits</code>	Jobs which are over their server soft limits
<code>normal_jobs</code>	The preemption level into which a job falls if it does not fit into any other specified level.

Example:

```
preempt_prio: "starving_jobs, normal_jobs, fairshare"
```

In this example, the first line states that starving jobs have the highest priority, then normal jobs, and jobs whose entities are over their fairshare limit are third highest.

Example:

```
preempt_prio: "normal_jobs, starving_jobs+fairshare"
```

This example shows that starving jobs whose entities are also over their fairshare limit are lower priority than normal jobs.

Format: Quoted list

Default: *express_queue, normal_jobs*

preempt_queue_prio

Specifies the minimum queue priority required for a queue to be classified as an express queue. Express queues do not require *by_queue* or *sort_queues* to be *True*.

Format: Integer

Default: *150*

preempt_requeue

Deprecated. Add an “*R*” to *preempt_order* parameter.

preempt_sort

Whether jobs most eligible for preemption will be sorted according to their start times.

If set to “*min_time_since_start*”, first job preempted will be that with most recent start time.

If not set, preempted job will be that with longest running time.

Must be commented out in order to be unset; default scheduler configuration file has this parameter set to *min_time_since_start*.

Allowable values: “*min_time_since_start*”, or no *preempt_sort* setting.

See [section 4.8.33.7, "Sorting Within Preemption Level" on page 229 in the PBS Professional Administrator's Guide](#).

Format: String

Default: *min_time_since_start*

preempt_starving

Deprecated. Add “*starving_jobs*” to *preempt_prio* parameter.

preempt_suspend

Deprecated. Add an “S” to preempt_order parameter.

primetime_prefix

Queue names starting with this prefix are treated as primetime queues. Jobs will only run in these queues during primetime. Primetime and non-primetime are defined in the holidays file.

See [section 4.8.34, "Using Primetime and Holidays" on page 235 in the PBS Professional Administrator's Guide](#).

Format: String

Default: *p_*

prime_exempt_anytime_queues

Determines whether *anytime* queues are controlled by backfill_prime.

If set to *True*, jobs in an anytime queue will not be prevented from running across a primetime/non-primetime or non-primetime/primetime boundary.

If set to *False*, the jobs in an anytime queue may not cross this boundary, except for the amount specified by their prime_spill setting.

See also backfill_prime, prime_spill.

Format: Boolean.

Default: *False*

prime_spill

Specifies the amount of time a job can spill over from non-primetime into primetime or from primetime into non-primetime. This option can be separately specified for prime- and non-primetime. This option is only meaningful if backfill_prime is *True*.

See also backfill_prime, prime_exempt_anytime_queues.

For example, the first setting below means that non-primetime jobs can spill into primetime by 1 hour. However the second setting means that jobs in either prime/non-prime can spill into the other by 1 hour.

```
prime_spill: 1:00:00 prime
```

```
prime_spill: 1:00:00 all
```

Format: Duration

Default: *00:00:00*

provision_policy

Specifies how vnodes are selected for provisioning. Can be set by Manager only; readable by all. Can be set to one of the following:

avoid_provision

PBS first tries to satisfy the job's request from free vnodes that already have the requested AOE instantiated. PBS uses `node_sort_key` to sort these vnodes.

If it cannot satisfy the job's request using vnodes that already have the requested AOE instantiated, it does the following:

PBS uses the server's `node_sort_key` to select the free vnodes that must be provisioned in order to run the job, choosing from any free vnodes, regardless of which AOE is instantiated on them.

Of the selected vnodes, PBS provisions any that do not have the requested AOE instantiated on them.

aggressive_provision

PBS selects vnodes to be provisioned without considering which AOE is currently instantiated.

PBS uses the server's `node_sort_key` to select the vnodes on which to run the job, choosing from any free vnodes, regardless of which AOE is instantiated on them. Of the selected vnodes, PBS provisions any that do not have the requested AOE instantiated on them.

Format: String

Default: *aggressive_provision*

resources

Specifies those resources which are not to be over-allocated when scheduling jobs. Vnode-level boolean resources are automatically honored and do not need to be listed here. Limits are set by setting `resources_available.resourceName` on vnodes, queues, and the server. The Scheduler will consider numeric (integer or float) items as consumable resources and ensure that no more are assigned than are available (e.g. `ncpus` or `mem`). Any string resources will be compared using string comparisons. If “`host`” is not added to the resources line, then when the user submits a job requesting a specific vnode in the following syntax:

```
qsub -l select=host=vnodeName
```

the job will run on any host.

Format: String.

Default: *ncpus, mem, arch, host, vnode, aoe, netwins*

resource_unset_infinite

Resources in this list are treated as infinite if they are unset. Cannot be set differently for primetime and non-primetime.

Example:

```
resource_unset_infinite: "vmem, foo_licenses"
```

Format: Comma-delimited list of resources

Default: Empty list

round_robin

If set to *True*, the scheduler will consider one job from the first queue, then one job from the second queue, and so on in a circular fashion. If *sort_queues* is set to *True*, the queues are ordered with the highest priority queue first. Each scheduling cycle starts with the same highest-priority queue, which will therefore get preferential treatment.

If *round_robin* is set to *False*, the scheduler will consider jobs according to the setting of the *by_queue* parameter.

When *True*, overrides the *by_queue* parameter.

Format: Boolean

Default: *False all*

server_dyn_res

Directs the Scheduler to replace the Server's *resources_available* values with new values returned by a site-specific external program.

See [section 5.14.4.1, "Dynamic Server-level Resources" on page 327 in the PBS Professional Administrator's Guide](#) for details of usage.

Format: String

Default: Unset.

smp_cluster_dist

Specifies how single-host jobs should be distributed to all hosts of the complex.

Options:

pack

Keep putting jobs onto one host until it is full and then move on to the next.

round_robin

Put one job on each vnode in turn before cycling back to the first one.

lowest_load

Put the job on the lowest-loaded host.

See [section 4.8.42, "SMP Cluster Distribution" on page 267 in the PBS Professional Administrator's Guide](#) and [section 4.8.27, "Using Load Balancing" on page 187 in the PBS Professional Administrator's Guide](#).

Format: String

Default: *pack all*

sort_by

Deprecated. Use `job_sort_key`.

sort_queues

If set to *True* queues are sorted so that the highest priority queues are considered first. Queues are sorted by each queue's priority attribute. The queues are sorted in a descending fashion, that is, a queue with priority 6 comes before a queue with priority 3.

When set to *False*, queues are not sorted.

This is a prime option, which means it can be selectively applied to prime-time or non-prime-time.

Note that the sorted order of queues is not taken into consideration unless `by_queue` is set to *True*.

See [section 4.8.45, "Sorting Queues into Priority Order" on page 272 in the PBS Professional Administrator's Guide](#).

Format: Boolean

Default: *True ALL*

strict_fifo

Deprecated. Use `strict_ordering`.

strict_ordering

Specifies that jobs must be run in the order determined by whatever sorting parameters are being used. This means that a job cannot be skipped due to resources required not being available. If a job due to run next cannot run, no job will run, unless backfilling is used, in which case jobs can be back-filled around the job that is due to run next.

See [section 4.8.19, "FIFO Scheduling" on page 175 in the PBS Professional Administrator's Guide](#).

Example line in `PBS_HOME/sched_priv/sched_config`:

```
strict_ordering: True ALL
```

Format: Boolean.

Default: *False all*

sync_time

Deprecated. The amount of time between writing the fairshare usage data to disk. Requires **fair_share** to be enabled.

Format: Duration

Default: *1:00:00*

unknown_shares

The number of shares for the *unknown* group. These shares determine the portion of a resource to be allotted to that group via fairshare. Requires **fair_share** to be enabled.

See [section 4.8.18, "Using Fairshare" on page 165 in the PBS Professional Administrator's Guide](#).

Format: Integer

Default: The unknown group gets 0 shares unless set.

Chapter 5

Resources

This chapter describes the resources provided by PBS Professional.

5.1 Resource Data Types

Data types for built-in and custom resource are described in [section 7.1, “List of Formats”, on page 403](#).

5.2 Advice on Using Resources

Resource names are case-insensitive.

The following advice will help you use resources.

5.2.1 Using boolean Resources

See ["Boolean" on page 404](#).

Non-consumable.

5.2.2 Using duration Resources

See ["Duration" on page 405](#).

Specifies a maximum time period the resource can be used.

Non-consumable.

5.2.3 Using float Resources

See ["Float" on page 405](#).

Consumable.

5.2.4 Using long Resources

See ["long" on page 407](#).

Consumable.

5.2.5 Using size Resources

See ["Size" on page 408](#).

Consumable.

5.2.6 Using string Resources

See ["String \(resource value\)" on page 408](#).

Non-consumable.

We do not recommend using non-printing characters.

When using `qsub -l <string resource>=<string value>`, you must escape string values for both `qsub` and the shell. Example:

```
qsub -lteststring='\"abc def\"'
```

The final quote should be single, not double.

Values are case-sensitive.

5.2.7 Using string_array Resources

See ["string_array" on page 408](#).

Non-consumable. Resource request will succeed if request matches one of the values.

Resource request can contain only one string.

A string array resource with one value works exactly like a string resource.

The value of `resources_default.<string array resource>` can only be one string.

5.3 Custom Resource Formats

The names of custom numeric resources must be alphanumeric with a leading alphabetic: `[a-zA-Z][a-zA-Z0-9_]*`. Allowable values for float and long resources are the same as for built-in resources. Custom boolean, time, size, string or string array resources must have the same format as built-in resources.

5.4 Built-in Resources

Different resources are available on different systems, often depending on the architecture of the computer itself. The table below lists the available resources that can be requested by PBS jobs on any system.

Table 5-1: Built-in Resources

Resource	Description
accelerator	Indicates whether this vnode is associated with an accelerator. Host-level. Can be requested only inside of a select statement. On Cray, this resource exists only when there is at least one associated accelerator. On Cray, this is set to <i>True</i> when there is at least one associated accelerator whose state is <i>UP</i> . On Cray, set to <i>False</i> when all associated accelerators are in state <i>DOWN</i> . Used for requesting accelerators. Format: Boolean. Python type: bool
accelerator_memory	Indicates amount of memory for accelerator(s) associated with this vnode. Host-level. Can be requested only inside of a select statement. On Cray, PBS sets this resource only on vnodes with at least one accelerator with state = <i>UP</i> . For Cray, PBS sets this resource on the 0th NUMA node (the vnode with <code>PBScrayseg=0</code>), and the resource is shared by other vnodes on the compute node. For example, on <code>vnodeA_2_0</code> : <pre>resources_available.accelerator_memory=4196mb</pre> On <code>vnodeA_2_1</code> : <pre>resources_available.accelerator_memory=@vnodeA_2_0</pre> Consumable. Format: size. Python type: pbs.size

Table 5-1: Built-in Resources

Resource	Description
accelerator_model	Indicates model of the accelerator(s) associated with this vnode. Host-level. On Cray, PBS sets this resource only on vnodes with at least one accelerator with state = <i>UP</i> . Can be requested only inside of a select statement. Non-consumable. Format: String. Python type: str
aoe	List of AOE (Application Operating Environments) that can be instantiated on a vnode. Case-sensitive. An AOE is the environment that results from provisioning a vnode. Each job can request at most one AOE. Cannot be set on Server's host. Allowable values are site-dependent. Settable by Manager and Operator; visible to all. Non-consumable. Type: string array. Python type: str
arch	System architecture. One architecture can be defined for a vnode. One architecture can be requested per vnode. Allowable values and effect on job placement are site-dependent. Can be requested only inside of a select statement. Non-consumable. Type: string. Python type: str
cput	Amount of CPU time used by the job for all processes on all vnodes. Establishes a job resource limit. Can be requested only outside of a select statement. Non-consumable. Type: duration. Python type: pbs.duration
exec_vnode	Read-only. The vnodes that PBS estimates this job will use. Cannot be requested for a job; used for reporting only. Type: string. Python type: str
file	Size of any single file that may be created by the job. Can be requested only outside of a select statement. Type: size. Python type: pbs.size
host	Name of execution host. Can be requested only inside of a select statement. Automatically set to the short form of the hostname in the Mom attribute. On Cray compute node, set to <i><mpp_host>_<nid></i> . On CLE 2.2, value is set to “ <i>default</i> ”. Cannot be changed. Site-dependent. Type: string. Python type: str

Table 5-1: Built-in Resources

Resource	Description
<code>max_walltime</code>	Maximum walltime allowed for a shrink-to-fit job. Job's actual walltime is between <code>max_walltime</code> and <code>min_walltime</code> . PBS sets walltime for a shrink-to-fit job. If this resource is specified, <code>min_walltime</code> must also be specified. Must be greater than or equal to <code>min_walltime</code> . Cannot be used for <code>resources_min</code> or <code>resources_max</code> . Cannot be set on job arrays or reservations. If not specified, PBS uses 5 years as the maximum time slot. Can be requested only outside of a select statement. Non-consumable. Default: None. Type: duration. Python type: <code>pbs.duration</code>
<code>mem</code>	Amount of physical memory i.e. workingset allocated to the job, either job-wide or vnode-level. Can be requested only inside of a select statement. Consumable. Type: size. Python type: <code>pbs.size</code>
<code>min_walltime</code>	Minimum walltime allowed for a shrink-to-fit job. When this resource is specified, job is a shrink-to-fit job. If this attribute is set, PBS sets the job's walltime. Job's actual walltime is between <code>max_walltime</code> and <code>min_walltime</code> . Must be less than or equal to <code>max_walltime</code> . Cannot be used for <code>resources_min</code> or <code>resources_max</code> . Cannot be set on job arrays or reservations. Can be requested only outside of a select statement. Non-consumable. Default: None. Type: duration. Python type: <code>pbs.duration</code>
<code>mpiprocs</code>	Number of MPI processes for this chunk. Defaults to 1 if <code>ncpus > 0</code> , 0 otherwise. Can be requested only inside of a select statement. Cannot use sum from chunks as job-wide limit. Type: integer. Python type: <code>int</code> The number of lines in <code>PBS_NODEFILE</code> is the sum of the values of <code>mpiprocs</code> for all chunks requested by the job. For each chunk with <code>mpiprocs=P</code> , the host name for that chunk is written to the <code>PBS_NODEFILE</code> <i>P</i> times.
<code>mpparch</code>	Deprecated. MPP compute node system type. Can be requested only outside of a select statement. Allowable values: XT or X2. Cray-only resource. Ignored at other systems. Type: string. Python type: <code>str</code>
<code>mppdepth</code>	Deprecated. Depth (number of threads) of each processor. Specifies the number of processors that each processing element will use. Can be requested only outside of a select statement. Cray-only resource. Ignored at other systems. Default: 1. Type: integer. Python type: <code>int</code>

Table 5-1: Built-in Resources

Resource	Description
mpphost	Deprecated. MPP host. Can be requested only outside of a select statement. Cray-only resource. Ignored at other systems. Type: string. Python type: str
mpplabels	<p>Deprecated. List of node labels. Runs the application only on those nodes with the specified labels. Format: comma-separated list of labels and/or a range of labels. Any lists containing commas should be enclosed in quotes escaped by backslashes. For example:</p> <pre>#PBS -l mpplabels=\"red,blue\"</pre> <p>or</p> <pre>qsub -l mpplabels=\"red,blue\"</pre> <p>Can be requested only outside of a select statement. Cray-only resource. Ignored at other systems. Type: string. Python type: str, one of “soft” or “hard”</p>
mppmem	Deprecated. The maximum memory for all applications. The per-processing-element maximum resident set size memory limit. Can be requested only outside of a select statement. Cray-only resource. Ignored at other systems. Type: size. Python type: pbs.size
mppnodes	<p>Deprecated. Manual placement list consisting of a comma-separated list of nodes (node1,node2), a range of nodes (node1-node2), or a combination of both formats. Node values are expressed as decimal numbers. The first number in a range must be less than the second number (i.e., 8-6 is invalid). A complete node list is required. Any lists containing commas should be enclosed in quotes escaped by backslashes. For example:</p> <pre>#PBS -l mppnodes=\"40-48,52-60,84,86,88,90\"</pre> <p>or</p> <pre>qsub -l mppnodes=\"40-48,52-60,84,86,88,90\"</pre> <p>Can be requested only outside of a select statement. Cray-only resource. Ignored at other systems. Type: string. Python type: str</p>
mppnppn	Deprecated. Number of processing elements (PEs) per node. Can be requested only outside of a select statement. Cray-only resource. Ignored at other systems. Type: integer. Python type: int

Table 5-1: Built-in Resources

Resource	Description
mppwidth	Deprecated. Number of processing elements (PEs) for the job. Can be requested only outside of a select statement. Cray-only resource. Ignored at other systems. Type: integer. Python type: int
naccelerators	<p>Indicates number of accelerators on the host. Host-level. On Cray, should not be requested for jobs; PBS does not pass the request to ALPS. On Cray, PBS sets this resource only on vnodes whose hosts have at least one accelerator with state = <i>UP</i>. PBS sets this resource to the number of accelerators with state = <i>UP</i>. For Cray, PBS sets this resource on the 0th NUMA node (the vnode with <code>PBScrayseg=0</code>), and the resource is shared by other vnodes on the compute node.</p> <p>For example, on vnodeA_2_0:</p> <pre>resources_available.naccelerators=1</pre> <p>On vnodeA_2_1:</p> <pre>resources_available.naccelerators=@vnodeA_2_0</pre> <p>Can be requested only inside of a select statement. Consumable. Format: long. Python type: int</p>
nchunk	<p>This is the number of chunks requested between plus symbols in a select statement. For example, if the select statement is <code>-lselect 4:ncpus=2+12:ncpus=8</code>, the value of <code>nchunk</code> for the first part is <i>4</i>, and for the second part it is <i>12</i>. The <code>nchunk</code> resource cannot be named in a select statement; it can only be specified by placing a number before the colon, as in the above example. When the number is omitted, <code>nchunk</code> is 1. Non-consumable. This resource can be used to specify the default number of chunks at the server or queue (replacing <code>mppwidth</code>.)</p> <p>Example: <code>set queue myqueue default_chunk.nchunk=2</code></p> <p>Settable by Manager and Operator; readable by all. This resource cannot be used in server and queue <code>resources_min</code> and <code>resources_max</code>. Format: Integer. Python type: int. Default value: <i>1</i></p>
ncpus	Number of processors requested. Cannot be shared across vnodes. Can be requested only inside of a select statement. Consumable. Type: integer. Python type: int

Table 5-1: Built-in Resources

Resource	Description
netwins	Number of network windows on a high performance switch. Can be requested only inside of a select statement. Read-only. Type: integer. Python type: int
nice	Nice value under which the job is to be run. Host-dependent. Can be requested only outside of a select statement. Type: integer. Python type: int
nodect	Deprecated. Number of chunks in resource request from selection directive, or number of hosts requested from node specification. Otherwise defaults to value of 1. Can be requested only outside of a select statement. Read-only. Type: integer. Python type: int
nodes	Deprecated. Number of hosts requested. Integer.
ompthreads	<p>Number of OpenMP threads for this chunk. Defaults to <code>ncpus</code> if not specified. Can be requested only inside of a select statement. Non-consumable. Cannot use sum from chunks as job-wide limit. Type: integer. Python type: int</p> <p>For the MPI process with rank 0, the environment variables <code>NCPUS</code> and <code>OMP_NUM_THREADS</code> are set to the value of <code>ompthreads</code>. For other MPI processes, behavior is dependent on MPI implementation.</p>
pcput	Amount of CPU time allocated to any single process in the job. Establishes a job resource limit. Non-consumable. Can be requested only outside of a select statement. Type: duration. Python type: <code>pbs.duration</code>
pmem	Amount of physical memory (workingset) for use by any single process of the job. Establishes a job resource limit. Can be requested only outside of a select statement. Non-consumable. Type: size. Python type: <code>pbs.size</code>
pvmem	Amount of virtual memory for use by the job. Establishes a job resource limit. Can be requested only outside of a select statement. Not consumable. Type: size. Python type: <code>pbs.size</code>
software	Site-specific software specification. Can be requested only outside of a select statement. Allowable values and effect on job placement are site-dependent. Type: string. Python type: <code>pbs.software</code>

Table 5-1: Built-in Resources

Resource	Description
start_time	Read-only. The estimated start time for this job. Cannot be requested for a job; used for reporting only. Type: long. Python type: int
vmem	Amount of virtual memory for use by all concurrent processes in the job. Establishes a per-chunk limit. Can be requested only inside of a select statement. Consumable. Type: size. Python type: pbs.size
vnode	Name of virtual node (vnode) on which to execute. For use inside chunks only. Site-dependent. Can be requested only inside of a select statement. Type: string. Python type: str See the <code>pbs_node_attributes(7B)</code> man page.
vntype	This resource represents the type of the vnode. Automatically set by PBS to one of two specific values for cray vnodes. Has no meaning for non-Cray vnodes. Can be requested only inside of a select statement. Non-consumable. Format: String array. Automatically assigned values for Cray vnodes: <i>cray_compute</i> : This vnode represents part of a compute node. <i>cray_login</i> : This vnode represents a login node. Default value: None. Python type: str
walltime	Actual elapsed (wall-clock, except during Daylight Savings transitions) time during which the job can run. Establishes a job resource limit. Can be requested only outside of a select statement. Non-consumable. Default: 5 years. Type: duration. Python type: pbs.duration

5.5 Custom Cray Resources

PBS provides custom resources specifically created for the Cray. They are listed in the following table:

Table 5-2: Custom Cray Resources

Resource Name	Description
PBScrayhost	<p>On CLE 2.2, this is set to “<i>default</i>”.</p> <p>On CLE 3.0 and higher, used to delineate a Cray system, containing ALPS, login nodes running PBS MOMs, and compute nodes, from a separate Cray system with a separate ALPS. Non-consumable. The value of PBScrayhost is set to the value of mpp_host for this system.</p> <p>Format: String.</p> <p>Default: CLE 2.2: “<i>default</i>”; CLE 3.0 and higher: None</p>
PBScraylabel_ <label name>	<p>Tracks labels applied to compute nodes. For each label on a compute node, PBS creates a custom resource whose name is a concatenation of <i>PBScraylabel_</i> and the name of the label. PBS sets the value of the resource to <i>True</i> on all vnodes representing the compute node.</p> <p>Name format: <i>PBScraylabel_<label name></i></p> <p>For example, if the label name is <i>Blue</i>, the name of this resource is <i>PBScraylabel_Blue</i>.</p> <p>Format: Boolean. Default: None</p>
PBScraynid	<p>Used to track the node ID of the associated compute node. All vnodes representing a particular compute node share a value for PBScraynid. Non-consumable.</p> <p>The value of PBScraynid is set to the value of node_id for this compute node.</p> <p>Non-consumable. Format: String. Default: None</p>

Table 5-2: Custom Cray Resources

Resource Name	Description
PBScrayorder	<p>Used to track the order in which compute nodes are listed in the Cray inventory. All vnodes associated with a particular compute node share a value for PBScrayorder. Non-consumable.</p> <p>Vnodes for the first compute node listed are assigned a value of 1 for PBScrayorder. The vnodes for each subsequent compute node listed are assigned a value one greater than the previous value.</p> <p>Do not use this resource in a resource request.</p> <p>Format: Integer. Default: None</p>
PBScrayseg	<p>Tracks the segment ordinal of the associated NUMA node. For the first NUMA node of a compute host, the segment ordinal is 0, and the value of PBScrayseg for the associated vnode is 0. For the second NUMA node, the segment ordinal is 1, PBScrayseg is 1, and so on.</p> <p>Non-consumable. Format: String. Default: None</p>

5.6 Specifying Architectures

The `resources_available.arch` resource is the value reported by MOM unless explicitly set by the Administrator. The values for `arch` are:

Table 5-3: Values for `resources_available.arch`

OS	Resource Label
AIX 5	aix4
CLE	XT
HP-UX 11	hpux11
Linux	linux
Linux with cpusets	linux_cpuset
Solaris	solaris7

Table 5-3: Values for resources_available.arch

OS	Resource Label
Unicos	unicos
Unicos MK2	unicosmk2
Unicos SMP	unicosmp

Chapter 6

Attributes

This chapter lists all of the PBS attributes. Attributes are listed by the PBS object they modify. For example, all attributes of jobs are listed in [section 6.11, “Job Attributes”, on page 375](#). Attributes are case-sensitive.

6.1 When Attribute Changes Take Effect

When you set the value of most attributes, the change takes place immediately. You do not need to restart any daemons in order to make the change.

6.2 How To Set Attributes

Most attributes are set using the `qmgr` command. However, some vnode attributes must be set using the `pbs_mom -s insert` command, to create a Version 2 configuration file. For information about these requirements, see [section 3.5.2, “Choosing Configuration Method” on page 48 in the PBS Professional Administrator’s Guide](#). The following are the instructions for setting all other attributes.

To set the value of an attribute, use the `qmgr` command, either from the command line or within `qmgr`:

```
qmgr -c 'set <object> <attribute> = <value>'  
Qmgr: set <object> <attribute> = <value>
```

To unset the value of an attribute:

```
qmgr -c 'unset <object> <attribute>'  
Qmgr: unset <object> <attribute>
```

where *<object>* is one of *server*, *queue*, *hook*, *node*, or *sched*.

For example, to set `resources_max.walltime` at the server to be 24 hours:

```
Qmgr: set server resources_max.walltime = 24:00:00
```

See “[qmgr](#)” on page 151.

6.3 Viewing Attribute Values

If you want to view attribute values, the following commands are helpful:

`qstat`; see [“qstat” on page 194 of the PBS Professional Reference Guide](#)

`qmgr`; see [“qmgr” on page 151 of the PBS Professional Reference Guide](#)

`pbs_rstat`; see [“pbs_rstat” on page 78 of the PBS Professional Reference Guide](#)

- To see server attributes, use one of the following:

`qstat -B -f`

`Qmgr: list server`

- To see queue attributes, use one of the following:

`qstat -Q -f <queue name>`

`Qmgr: list queue <queue name>`

- To see job attributes:

`qstat -f <job ID>`

- To see hook attributes:

`Qmgr: list hook <hook name>`

- To see scheduler attributes:

`Qmgr: list sched`

- To see vnode attributes:

`Qmgr: list node <node name>`

- To see reservation attributes:

`Qmgr: pbs_rstat -F`

6.4 Attribute Table Format

In the following tables, the columns contain the following information:

Name

The name of the attribute

Description

A description of the attribute's function

Format

The attribute's format

Val/Opt

If the attribute can take only specific values or options, each is listed here

Value/Option Description

If the attribute can take only specific values or options, the behavior of each value or option is described here

Default Value, Def Val

The attribute's default value, if any

Python Type

The attribute's Python attribute value type

User, Oper, Mgr

Indicates the actions allowed for unprivileged users, Operators, and Managers

The following table shows the operations allowed and their symbols:

Table 6-1: User, Operator, Manager Actions

Symbol	Explanation
<i>r</i>	Entity can read attribute
<i>w</i>	Entity can directly set or alter attribute
<i>s</i>	Entity can set but not alter attribute
<i>a</i>	Entity can alter but not set attribute
<i>i</i>	Entity can indirectly set attribute
-	Entity cannot set or alter attribute, whether directly or indirectly

6.5 Caveats

- The Python types listed as Python dictionaries support a restricted set of operations. They can reference values by index. Other features, such as `has_key()`, are not available.
- Do not use `qmgr` to set attributes for reservation queues.

6.6 Server Attributes

Server attributes are divided into these groups:

- Those that can be set by an operator or manager
- Those that are read-only

Server Attributes									
Name	Description	Format	Val / Opt	Value/Option Description	Default Value	Python Type	User	Oper	Mgr
The following server attributes can be set by an operator or manager:									
<code>acl_host_enable</code>	Specifies whether the server obeys the host access control list in the <code>acl_hosts</code> server attribute.	<i>Boolean.</i>		When this attribute is <i>True</i> , the server limits host access according to the access control list.	<i>False</i> ; all hosts allowed access	bool	r	r	r, w
<code>acl_hosts</code>	List of hosts from which services can be requested of this server. Requests from the Server's host always honored whether or not that host is in the list. This list contains the fully qualified domain names of the hosts. List is evaluated left-to-right; first match in list is used.	<i>String.</i> Form: <code>[+/-]host-name.domain[,...]</code>			None; all hosts allowed access	pbs.acl	r	r	r, w
<code>acl_resv_group_enable</code>	Specifies whether the server obeys the group reservation access control list in the <code>acl_resv_groups</code> server attribute.	<i>Boolean</i>		When this attribute is <i>True</i> , the server limits group access according to the access control list.	<i>False</i> ; all groups allowed access	bool	r	r	r, w

Server Attributes									
Name	Description	Format	Val / Opt	Value/Option Description	Default Value	Python Type	User	Operator	Manager
<code>acl_resv_groups</code>	List of groups allowed or denied permission to create reservations in this PBS complex. The groups in the list are groups on the server host, not submission hosts. List is evaluated left-to-right; first match in list is used.	<i>String.</i> Form: <code>[+ -]group_name[,...]</code>				<code>pbs.acl</code>	r	r	r, w
<code>acl_resv_host_enable</code>	Specifies whether the server obeys the host reservation access control list in the <code>acl_resv_hosts</code> server attribute.	<i>Boolean</i>		When this attribute is <i>True</i> , the server limits host access according to the access control list.	<i>False</i> ; access allowed from all hosts	<code>bool</code>	r	r	r, w
<code>acl_resv_hosts</code>	List of hosts from which reservations can be created in this PBS complex. This list is made up of the fully-qualified domain names of the hosts. List is evaluated left-to-right; first match in list is used.	<i>String.</i> Form: <code>[+ -]hostname.domain[,...]</code>			None; access allowed from all hosts	<code>pbs.acl</code>	r	r	r, w
<code>acl_resv_user_enable</code>	Specifies whether the server limits which users are allowed to create reservations, according to the access control list in the <code>acl_resv_users</code> server attribute.	<i>Boolean</i>		When this attribute is <i>True</i> , the server limits user reservation creation according to the access control list.	<i>False</i> ; all users are allowed to create reservations	<code>bool</code>	r	r	r, w

Server Attributes									
Name	Description	Format	Val / Opt	Value/Option Description	Default Value	Python Type	User	Oper	Mgr
acl_resv_users	List of users allowed or denied permission to create reservations in this PBS complex. List is evaluated left-to-right; first match in list is used.	<i>String</i> . Form: <i>[+ -]user</i> <i>[@host][,...]</i>			None	pbs.acl	r	r	r, w
acl_user_enable	Specifies whether the server limits which users are allowed to run commands at the server, according to the control list in the <code>acl_users</code> server attribute.	<i>Boolean</i>		When this attribute is <i>True</i> , the server limits user access according to the access control list.	<i>False</i> ; all users have access	bool	r	r	r, w
acl_users	List of users allowed or denied permission to run commands at this server. List is evaluated left-to-right; first match in list is used.	<i>String</i> . Form: <i>[+ -]user</i> <i>[@host][,...]</i>			None; all users allowed access	pbs.acl	r	r	r, w
backfill_depth	Modifies backfilling behavior. Sets the number of jobs that are to be backfilled around. Recommendation: set this to less than 100.	Integer. Must be ≥ 1 .	≥ 1	PBS backfills around the specified number of jobs.	Unset; backfill depth is 1	int	r	r, w	r, w
			<i>Unset</i>	Backfill depth is set to 1					
comment	Informational text.	<i>String</i> of any form			None	str	r	r, w	r, w

Server Attributes									
Name	Description	Format	Val / Opt	Value/Option Description	Default Value	Python Type	User	Operator	Manager
default_chunk	The list of resources which will be inserted into each chunk of a job's select specification if the corresponding resource is not specified by the user. This provides a means for a site to be sure a given resource is properly accounted for even if not specified by the user.	<i>String.</i> Form: <i>default_chunk.<res>=<val>,default_chunk.<res>=<val>,...</i>			None	Dictionary: default_chunk[<resource name>]=<value> where <resource name> is any built-in or custom resource	r	r, w	r, w
default_node	No longer used.						-	-	-
default_qdel_arguments	Argument to qdel command. Automatically added to all qdel commands. See qdel (1B). Overrides standard defaults. Overridden by arguments given on the command line.	<i>String.</i> Form: - <i>Wsuppress_mail=<N> .</i>			None	pbs.args	r	r, w	r, w
default_qsub_arguments	Arguments that are automatically added to the qsub command. Any valid arguments to qsub command, such as job attributes. Setting a job attribute via default_qsub_arguments sets that attribute for each job which does not explicitly override it. See qsub (1B). Settable by the administrator via the qmgr command. Overrides standard defaults. Overridden by arguments given on the command line and in script directives.	<i>String.</i> Form: <option> <value> <option> <value> , e.g. -r y -N MyJob To add to existing: Qmgr: s s default_qsub_arguments += "<option> <value>"			None	pbs.args	r	r, w	r, w

Server Attributes									
Name	Description	Format	Val / Opt	Value/Option Description	Default Value	Python Type	User	Operator	Manager
default_queue	The name of the default target queue. Used for requests that do not specify a queue name. Must be set to an existing queue.	<i>Queue name</i>			None; must be set to an existing queue	pbs.queue	r	r, w	r, w
eligible_time_enable	Controls starving behavior. Viewable via <code>qstat</code> by job owner, Operator and Manager.	<i>Boolean</i>	<i>True</i>	The value of the job's <code>eligible_time</code> attribute is used for its starving time.	<i>False</i>	bool	r	r	r, w
			<i>False</i>	The value of <code>now() - etime</code> is used for the job's starving time.					
est_start_time_freq	Interval at which PBS calculates estimated start times and vnodes for all jobs. Best value is workload dependent. Recommendation: set this to two hours.	<i>Duration</i> . Expressed as an integer number of seconds or a <i>Duration</i> . See section , "Duration", on page 405	>0	PBS calculates estimated start times and vnodes for all jobs at the specified interval.	Unset	pbs.duration	r	r, w	r, w
			0	PBS calculates estimated start times and vnodes for all jobs at every scheduling cycle.					
			<i>Unset</i>	PBS does not calculate estimated start times or vnodes for all jobs; PBS calculates these only for the top N jobs specified in <code>backfill_depth</code> .					

Server Attributes									
Name	Description	Format	Val / Opt	Value/Option Description	Default Value	Python Type	User	Oper	Mgr
flatuid	<p>Used for authorization allowing users to submit and alter jobs. Specifies whether user names are treated as being the same across the PBS server and all submission hosts in the PBS complex. Can be used to allow users without accounts at the Server host to submit jobs.</p> <p>If UserA has an account at the Server host, PBS requires that UserA@<server host> is the same as UserA@<execution host>.</p>	<i>Boolean</i>	<i>True</i>	<p>PBS assumes that UserA@<submit host> is same user as UserA@<server>.</p> <p>Jobs that run under the name of the job owner do not need authorization.</p> <p>A job submitted under a different username, by using the u option to the qsub command, requires authorization.</p> <p>Entries in .rhosts or hosts.equiv are not checked, so even if UserA@host1 has an entry for UserB@host2, UserB@host2 cannot operate on UserA@host1's jobs.</p> <p>User without account on Server can submit jobs.</p>	<i>False</i> ; authorization is required	bool	r	r	r, w
			<i>False</i>	<p>PBS does not assume that UserA@<submission host> is the same user as UserA@<server host>.</p> <p>Jobs that run under the name of the job owner need authorization.</p> <p>Users must have accounts on the Server host to submit jobs.</p>					
job_history_duration	Specifies the length of time PBS will keep each job's history.	<i>Duration</i>			<i>Two weeks</i>	int	r	r	r, w

Server Attributes									
Name	Description	Format	Val / Opt	Value/Option Description	Default Value	Python Type	User	Oper	Mgr
job_history_enabled	Enables job history management. Setting this attribute to <i>True</i> enables job history management.	<i>Boolean</i>			<i>False</i>	bool	r	r	r, w
job_requeue_timeout	Specifies the length of time that can be taken while requeueing a job. Minimum allowed value: 1 second. Maximum allowed value: 3 hours.	Duration			45 seconds	pbs.duration	r	r, w	r, w

Server Attributes								
Name	Description	Format	Val / Opt	Value/Option Description	Default Value	Python Type	User	Oper Mgr
job_sort_formula	Formula for computing job priorities. Described in the PBS Professional Administrator's Guide . If the attribute <code>job_sort_formula</code> is set, the scheduler will use the formula in it to compute job priorities. If it is unset, the scheduler computes job priorities according to fairshare, if fairshare is enabled. If neither is defined, the scheduler uses <code>job_sort_key</code> . When the scheduler sorts jobs according to the formula, it computes a priority for each job, where that priority is the value produced by the formula. Jobs with a higher value get higher priority. Can be set by Manager or Operator. Viewable by users, Manager or Operator. The formula can be made up of expressions, where expressions contain terms which are added, subtracted, multiplied, or divided, and which can contain parentheses, exponents, and unary plus and minus.	<i>String</i> containing mathematical formula			Unset	<code>pbs.job_sort_formula</code>	r, w	r, w, r, w
log_events	Specifies the types of events which are logged.	<i>Integer</i> representation of bit string			511	int	r	r, w, r, w

Server Attributes									
Name	Description	Format	Val / Opt	Value/Option Description	Default Value	Python Type	User	Oper	Mgr
mail_from	The username from which server-generated mail is sent to users. Mail is sent to this address upon failover. On Windows, requires fully qualified mail address.	<i>String</i>			<i>adm</i>	str	r	r	r, w
managers	List of PBS managers.	<i>String</i> . Form: “ <i>user@host.sub.domain[,user@host.sub.domain..]</i> ”. The host, sub-domain, or domain name may be wildcarded with an asterisk (*).			<i>Root on the server host</i>	pbs.acl	r	r	r, w
max_array_size	The maximum number of sub-jobs allowed in any array job.	<i>Integer</i>			<i>10000</i>	int	r	r, w	r, w
max_concurrent_provision	The maximum number of vnodes allowed to be in the process of being provisioned. Cannot be set to zero.	Integer	>0		5	int	r	r	r, w

Server Attributes									
Name	Description	Format	Val / Opt	Value/Option Description	Default Value	Python Type	User	Operator	Manager
max_group_res	Old limit attribute. Incompatible with new limit attributes. The maximum amount of the specified resource that any single group may consume in this PBS complex.	<i>String</i> . Form: <i>max_group_res.<resource>=<value></i>	Any PBS resource, e.g. “ncpus”, “mem”, “pmem”		None	Dictionary: max_group_res[<resource name>]=<value> where <resource name> is any built-in or custom resource	r	r, w	r, w
max_group_res_soft	Old limit attribute. Incompatible with new limit attributes. The soft limit for the specified resource that any single group may consume in this complex. If a group is consuming more than this amount of the specified resource, their jobs are eligible to be preempted by jobs from groups who are not over their soft limit.	<i>String</i> . Form: <i>“max_group_res_soft.resource_name=value”</i>	Any PBS resource, e.g. “ncpus”, “mem”, “pmem”, etc.		None	Dictionary: max_group_res_soft[<resource name>]=<value> where <resource name> is any built-in or custom resource	r	r, w	r, w
max_group_un	Old limit attribute. Incompatible with new limit attributes. The maximum number of jobs owned by the users in one group allowed to be running within this complex at one time.	<i>Integer</i>			No limit	int	r	r, w	r, w

Server Attributes									
Name	Description	Format	Val / Opt	Value/Option Description	Default Value	Python Type	User	Oper	Mgr
max_group_run_soft	Old limit attribute. Incompatible with new limit attributes. The maximum number of jobs owned by the users in one group allowed to be running in this complex at one time. If a group has more than this number of jobs running, their jobs are eligible to be preempted by jobs from groups who are not over their soft limit.	<i>Integer</i>			None	int	r	r, w	r, w
max_queued	Limit attribute. The maximum number of jobs allowed to be queued or running in the complex. Can be specified for projects, users, groups, or all. Cannot be used with old limit attributes.	Limit specification. See Chapter 7, "Formats", on page 403 .			None	None	r	r, w	r, w
max_queued_res.<resource>	Limit attribute. The maximum amount of the specified resource allowed to be allocated to jobs queued or running in the complex. Can be specified for projects, users, groups, or all. Cannot be used with old limit attributes.	Limit specification. See Chapter 7, "Formats", on page 403 .			None	None	r	r, w	r, w

Server Attributes									
Name	Description	Format	Val / Opt	Value/Option Description	Default Value	Python Type	User	Operator	Manager
max_run	Limit attribute. The maximum number of jobs allowed to be running in the complex. Can be specified for projects, users, groups, or all. Cannot be used with old limit attributes.	Limit specification. See Chapter 7, "Formats", on page 403 .			None	None	r	r, w	r, w
max_run_res. <resource>	Limit attribute. The maximum amount of the specified resource allowed to be allocated to jobs running in the complex. Can be specified for projects, users, groups, or all. Cannot be used with old limit attributes.	Limit specification. See Chapter 7, "Formats", on page 403 .			None	None	r	r, w	r, w
max_run_res. soft.<resource>	Limit attribute. Soft limit on the amount of the specified resource allowed to be allocated to jobs running in the complex. Can be specified for projects, users, groups, or all. Cannot be used with old limit attributes.	Limit specification. See Chapter 7, "Formats", on page 403 .			None	None	r	r, w	r, w
max_run_soft	Limit attribute. Soft limit on the number of jobs allowed to be running in the complex. Can be specified for projects, users, groups, or all. Cannot be used with old limit attributes.	Limit specification. See Chapter 7, "Formats", on page 403 .			None	None	r	r, w	r, w

Server Attributes									
Name	Description	Format	Val / Opt	Value/Option Description	Default Value	Python Type	User	Oper	Mgr
max_running	Old limit attribute. Incompatible with new limit attributes. The maximum number of jobs allowed to be selected for execution at any given time, from all possible jobs.	<i>Integer</i>			None	int	r	r, w	r, w
max_user_res	Old limit attribute. Incompatible with new limit attributes. The maximum amount of the specified resource that any single user may consume within this complex.	<i>String.</i> Form: <i>max_user_res.<resource>=<value></i>	Any PBS resource, e.g. “ncpus”, “mem”, “pmem”, etc.		None	Dictionary: max_user_res[<resource name>]=<value> where <resource name> is any built-in or custom resource	r	r, w	r, w
max_user_res_soft	Old limit attribute. Incompatible with new limit attributes. The soft limit on the amount of the specified resource that any single user may consume within a complex. If a user is consuming more than this amount of the specified resource, their jobs are eligible to be preempted by jobs from users who are not over their soft limit.	<i>String.</i> Form: <i>max_user_res_soft.resource_name=value</i>	Any valid PBS resource, e.g. “ncpus”, “mem”, “pmem”, etc		None	Dictionary: max_user_res_soft[<resource name>]=<value> where <resource name> is any built-in or custom resource	r	r, w	r, w

Server Attributes									
Name	Description	Format	Val / Opt	Value/Option Description	Default Value	Python Type	User	Oper	Mgr
max_user_run	Old limit attribute. Incompatible with new limit attributes. The maximum number of jobs owned by a single user allowed to be running within the complex at one time.	<i>Integer</i>			None	int	r	r, w	r, w
max_user_run_soft	Old limit attribute. Incompatible with new limit attributes. The soft limit on the number of jobs owned by a single user that are allowed to be running within this complex at one time. If a user has more than this number of jobs running, their jobs are eligible to be preempted by jobs from users who are not over their soft limit.	<i>Integer</i>			None	int	r	r, w	r, w
node_fail_requeue	Controls whether running jobs are automatically requeued or are deleted when the primary execution vnode fails. Number of seconds to wait after losing contact with Mother Superior before requeueing or deleting jobs. Reverts to default value when server is restarted.	<i>Integer. Seconds.</i>	Unset, zero, less than zero	Jobs are not requeued; they are left in the <i>Running</i> state until the execution vnode is recovered.	310	int	r	r, w	r, w
			Greater than zero	Jobs are requeued if they are marked as rerunnable, or are deleted when the node has been down for the specified number of seconds.					

Server Attributes									
Name	Description	Format	Val / Opt	Value/Option Description	Default Value	Python Type	User	Operator	Manager
node_group_enable	Specifies whether placement sets (which includes node grouping) are enabled. See node_group_key server attribute.	<i>Boolean</i>		When set to <i>True</i> , placement sets are enabled.	<i>False</i>	bool	r	r, w	r, w
node_group_key	Specifies the resources to use for placement sets (node grouping). Overridden by queue's node_group_key attribute. See node_group_enable server attribute.	<i>string_array</i> When specifying multiple resources, enclose the value in double quotes.			Unset	pbs.node_group_key	r	r, w	r, w
operators	List of PBS operators.	<i>String</i> . Form: <i>user@host.sub.domain[,user@host.sub.domain...]. The host, sub-domain, or domain name may be wildcarded with an asterisk (*)</i> .			None	pbs.acl	r	r	r, w
pbs_license_file_location	Deprecated. Do not use.	-	-	-	-	-	-	-	-

Server Attributes									
Name	Description	Format	Val / Opt	Value/Option Description	Default Value	Python Type	User	Oper	Mgr
pbs_license_info	Location of license information. Can be port and hostname of license server, or local pathname to the actual license file associated with a license server. Delimiter between items is colon (":") for UNIX/Linux and semi-colon (";") for Windows.	<i>String</i> . Port and host-name form: <port1>@<host1>:<port2>@<host2>:...:<portN>@<hostN>:<path to license file> where <host1>,<host2>,...<hostN> can be IP addresses, and the license file can be listed first or last.			None	str	r	r	r, w
pbs_license_linger_time	The number of seconds to keep an unused CPU license, when the number of licenses is above the value given by pbs_license_min.	<i>Integer</i> . Seconds.			31536000 seconds (1 year).	pbs.duration	r	r	r, w

Server Attributes									
Name	Description	Format	Val / Opt	Value/Option Description	Default Value	Python Type	User	Oper	Mgr
pbs_license_max	Maximum number of licenses to be checked out at any time, i.e. maximum number of CPU licenses to keep in the PBS local license pool. Sets a cap on the number of CPUs that can be licensed at one time.	<i>Integer</i>			Maximum value for an integer	int	r	r	r, w
pbs_license_min	Minimum number of CPUs to permanently keep licensed, i.e. the minimum number of CPU licenses to keep in the PBS local license pool. This is the minimum number of licenses to keep checked out.	<i>Integer</i>			Zero	int	r	r	r, w
query_other_jobs	Controls whether unprivileged users are allowed to select or query the status of jobs owned by other users.	<i>Boolean</i>		When this attribute is <i>True</i> , unprivileged users can query or select other users' jobs.	<i>True</i>	bool	r	r	r, w
require_cred	Specifies the Kerberos credential authentication method required. All jobs submitted without the specified credential will be rejected. See <code>require_cred_enable</code> . Depends on optional Kerberos and DCE support. Not supported under Windows.	<i>String</i>	<i>krb5</i>		Unset	str	r	r	r, w
			<i>dce</i>						

Server Attributes									
Name	Description	Format	Val / Opt	Value/Option Description	Default Value	Python Type	User	Oper	Mgr
require_cred_enable	Specifies whether the server is to use the Kerberos credential authentication method given in the <code>require_cred</code> server attribute. Depends on optional Kerberos and DCE support. Not supported under Windows.	<i>Boolean</i>		<i>True</i> means use the Kerberos authentication method specified.	<i>False</i> ; no Kerberos credential authentication used	bool	r	r	r, w
reserve_retry_cutoff	The time period before the reservation start time during which PBS does not attempt to reconfirm a degraded reservation. When this value is changed, all degraded reservations use the new value. Must be greater than zero.	<i>Integer. Seconds.</i>			7200 (2 hours)	int	-	-	r, w
reserve_retry_init	The amount of time after a reservation becomes degraded that PBS waits before attempting to reconfirm the reservation. When this value is changed, only reservations that become degraded after the change use the new value. Must be greater than zero.	<i>Integer. Seconds.</i>			7200 (2 hours)	int	-	-	r, w

Server Attributes									
Name	Description	Format	Val / Opt	Value/Option Description	Default Value	Python Type	User	Operator	Manager
resources_available	The list of available resources and their values defined on the server. Each resource is listed on a separate line.	<i>String.</i> Form: <i>resources_available.<resource>=<value></i>			None	Dictionary: resources_available[<resource name>]=<value> where <resource name> is any built-in or custom resource	r	r, w	r, w
resources_cost	No longer used.						-	-	-
resources_default	The list of default job-wide resource values that are set as limits for jobs in this complex when a) the job does not specify a limit, and b) there is no queue default. The value for a string array, e.g. resources_default.<string array resource>, can contain only one string. For host-level resources, see the default_chunk.<resource> server attribute.	<i>String.</i> Form: <i>resources_default.resource_name=value[,...]</i>			No limit	Dictionary: resources_default[<resource name>]=<value> where <resource name> is any built-in or custom resource	r	r, w	r, w

Server Attributes									
Name	Description	Format	Val / Opt	Value/Option Description	Default Value	Python Type	User	Operator	Manager
resources_max	The maximum amount of each resource that can be requested by any single job in this complex, if there is not a <code>resources_max</code> value defined for the queue at which the job is targeted. This attribute functions as a gating value for jobs entering the PBS complex.	<i>String</i> . Form: <code>resources_max.resource_name=value[,...]</code>			No limit	Dictionary: <code>resources_max[<resource name>]=<value></code> where <code><resource name></code> is any built-in or custom resource	r	r, w	r, w
resv_enable	Specifies whether or not advance and standing reservations can be created in this complex.	<i>Boolean</i>		When set to <i>True</i> , new reservations can be created. When changed from <i>True</i> to <i>False</i> , new reservations cannot be created, but existing reservations are honored.	<i>True</i>	bool	r	r	r, w
resv_post_processing_time	The amount of time allowed for reservations to clean up after running jobs. Reservation duration and end time are extended by this amount of time. Jobs are not allowed to run during the cleanup period.	Duration			Unset; behaves as if zero	int	r	r, w	r, w
rpp_highwater	The maximum number of RPP packets that can be in transit at any time.	<i>Integer</i>	Greater than or equal to one		64	int	r	r	r, w

Server Attributes									
Name	Description	Format	Val / Opt	Value/Option Description	Default Value	Python Type	User	Operator	Mgr
rpp_retry	The maximum number of times the RPP network library will try to send a UDP packet again before giving up. The number of retries is added to the original try, so if <code>rpp_retry</code> is set to 2, the total number of tries will be 3.	<i>Integer</i>	Greater than or equal to zero		10	int	r	r	r, w
scheduler_iteration	The time between scheduling iterations.	<i>Integer</i> . Seconds.			10 minutes (600 seconds)	pbs.duration	r	r, w	r, w
scheduling	Enables scheduling of jobs. Specified by value of -a option to <code>pbs_server</code> command. If -a is not specified, value is taken from previous invocation of <code>pbs_server</code> .	<i>Boolean</i>		When this attribute is set to <i>True</i> , scheduling is enabled.	<i>False</i> if never set via <code>pbs_server</code> command.	bool	r	r, w	r, w
single_signon_password_enabled	Only used on systems requiring passwords, such as Windows or HPCBP. Incompatible with other systems. Specifies whether or not users must give a password for each job. Can be enabled only when no jobs exist, or when all jobs have a bad password hold (“p” hold). Can be disabled only when no jobs exist.	<i>Boolean</i> .	<i>True</i>	Users submitting jobs must specify a password only once; PBS remembers it for future job execution.	UNIX: <i>False</i>	bool	r	r, w	r, w
			<i>False</i>	Users submitting jobs must specify a password for each job.	Windows: <i>True</i>				

Server Attributes									
Name	Description	Format	Val / Opt	Value/Option Description	Default Value	Python Type	User	Oper	Mgr
system_cost	No longer used.						-	-	-
The following server attributes can be set by root only:									
acl_roots	List of users with root privilege who can submit and run jobs in this PBS complex. For any job whose owner is root or Administrator, the job owner must be listed in this access control list, or the job is rejected. List is evaluated left-to-right; first match in list is used. Can be set or altered by root only, and only at the server host.	<i>String.</i> Form: <i>[+/-</i> <i>user</i> <i>[@host]][,...]</i>			None; no root jobs allowed	pbs.acl	r	r	r
The following server attributes are read-only:									
FLicenses	The number of floating licenses currently available for allocation to unlicensed CPUs. One license is required for each virtual CPU.	<i>Integer</i>			None	int	r	r	r

Server Attributes									
Name	Description	Format	Val / Opt	Value/Option Description	Default Value	Python Type	User	Oper	Mgr
license_count	The license_count attribute is made up of these numbers: Avail_Global, Avail_Local, Used, High_Use, Avail_Sockets, Unused_Sockets.	<i>String.</i> Form: <i>Avail_Global: <val></i> <i>Avail_Local: <val></i> <i>Used: <val></i> <i>High_Use: <val></i> <i>Avail_Sockets: <val></i> <i>Unused_Sockets: <val></i>	Avail_Global	The number of PBS CPU licenses still kept by the Altair license server (checked in.)	<i>Avail_Global:0</i> <i>Avail_Local:0</i> <i>Used:0</i> <i>High_Use:0</i> <i>Avail_Sockets:0</i> <i>Unused_Sockets:0</i>	pbs.license_count	r	r	r
			Avail_Local	The number of PBS CPU licenses still kept by PBS (checked out.)					
			Used	The number of PBS CPU licenses currently in use.					
			High_Use	The highest number of PBS CPU licenses ever checked out and used by the current instance of the PBS server.					
			Avail_Sockets	The total number of socket licenses in the license file.					
			Unused_Sockets	The number of unused socket licenses.					
pbs_version	The version of PBS for this server.	<i>String</i>			None	pbs.version	r	r	r
resources_assigned	The total of each type of resource allocated to jobs running in this complex.	<i>String.</i> Form: <i>resources_assigned.<res>=<val>[,resources_assigned.<res>=<val>,...]</i>			None	Dictionary: resources_assigned[<resource name>]=<value> where <resource name> is any built-in or custom resource	r	r	r

Server Attributes									
Name	Description	Format	Val / Opt	Value/Option Description	Default Value	Python Type	User	Oper	Mgr
server_host	The name of the server. The server name is the same as the host name.	Hostname. If the server is listening to a non-standard port, the port number is appended, with a colon, to the host name. Example: host.domain:9999			None	str	r	r	r

Server Attributes									
Name	Description	Format	Val / Opt	Value/Option Description	Default Value	Python Type	User	Oper	Mgr
server_state	The current state of the server:	<i>String</i>	<i>Hot_Start</i>	The server will run first any jobs that were running when it was shut down.	None	Server state constant pbs.SV_STATE_HOT	r	r	r
			<i>Idle</i>	The server is running. Scheduling has been turned off.		Server state constant pbs.SV_STATE_IDLE			
			<i>Active</i>	The server is running. The scheduler is not in a scheduling cycle.		Server state constant pbs.SV_STATE_ACTIVE			
			<i>Scheduling</i>	The server is running. The scheduler is in a scheduling cycle.		Server state constant pbs.SV_STATE_ACTIVE			
			<i>Terminating</i>	The server is terminating. No additional jobs will be run.		Server state constant pbs.SV_STATE_SHUTIMM or pbs.SV_STATE_SHUTSIG			
			<i>Terminating_Delayed</i>	Server is terminating in delayed mode. No new jobs will be run. Server will shut down after all running jobs are finished.		Server state constant pbs.SV_STATE_SHUTDEL			

Server Attributes									
Name	Description	Format	Val / Opt	Value/Option Description	Default Value	Python Type	User	Oper	Mgr
state_count	List of the number of jobs in each state in the complex. Suspended jobs are counted as running.	<i>String</i> . Form: <i>transiting=<X></i> , <i>queued=<Y></i> , ...			None	pbs.state_count	r	r	r
total_jobs	The total number of jobs in the complex. If the <code>job_history_enable</code> attribute is set to <i>True</i> , this includes jobs that are finished, deleted, and moved.	<i>Integer</i>			None	int	r	r	r

6.7 Scheduler Attributes

Scheduler Attributes									
Name	Description	Format	Val / Opt	Value/Option Description	Default Value	Python Type	User	Oper	Mgr
The following attributes are settable.									
do_not_span_psets	Specifies whether or not the scheduler requires the job to fit within one existing placement set.	Boolean	<i>True</i>	The job must fit in one existing placement set. All existing placement sets are checked. If the job fits in an occupied placement set, the job waits for the placement set to be available. If the job can't fit within a single placement set, it won't run.	<i>False</i>		r	r, w	r, w
			<i>False</i>	The scheduler first attempts to place the job in a single placement set, but if it cannot, it allows the job to span placement sets, running on whichever vnodes can satisfy the job's resource request.					
sched_cycle_length	The scheduler's maximum cycle length. Overwritten by the -a alarm option to pbs_sched command.	<i>Duration</i>			<i>20:00</i> (20 minutes)	None	r	r, w	r, w
The following attributes are read-only.									
pbs_version	The version of PBS for this scheduler.	<i>String</i>			None	None	-	r	r
sched_host	The hostname of the machine on which the scheduler runs.	<i>String</i>			<i>Server's host</i>	None	-	r	r

6.8 Reservation Attributes

Reservation attributes are divided into these groups:

- Those that can be set by users, operators, or managers
- Those that are read-only

Reservation Attributes									
Name	Description	Format	Val / Opt	Value/Option Description	Def Val	Python Type	User	Oper	Mgr
The following attributes can be set by user, operator, or manager:									
Account_ Name	No longer used.						-	-	-
Authorize d_Groups	List of groups who can or cannot submit jobs to this reservation. Group names are interpreted relative to the server, not the submission host. List is evaluated left-to-right; first match in list is used. This list is used to set the reservation queue's <code>acl_groups</code> attribute. See the <code>G</code> option to the <code>pbs_rsub</code> command.	<i>String.</i> Form: [+]-]group_ name,... [+]-]group_ name]			Jobs can be submitted by all groups	<code>pbs.acl</code>	r, w	r, w	r, w

Reservation Attributes									
Name	Description	Format	Val / Opt	Value/Option Description	Def Val	Python Type	User	Oper	Mgr
Authorized_Hosts	The list of hosts from which jobs can and cannot be submitted to this reservation. List is evaluated left-to-right; first match in list is used. This list is used to set the reservation queue's <code>acl_hosts</code> attribute. See the <code>H</code> option to the <code>pbs_rsub</code> command.	<i>String</i> . Form: <code>[+ -] \$host-name, ... , [+ -] \$host-name</code>			Jobs can be submitted from all hosts	<code>pbs.acl</code>	r, w	r, w	r, w

Reservation Attributes								
Name	Description	Format	Val / Opt	Value/Option Description	Def Val	Python Type	User	Oper
Authorized_Users	The list of users who can or cannot submit jobs to this reservation. This list is used to set the reservation queue's <code>acl_users</code> attribute. List is evaluated left-to-right; first match in list is used. See the <code>U</code> option to the <code>pbs_rsub</code> command.	<i>String</i> . Form: “ <code>[+ -]user[hostname.domain], ...[+ -]J...</code> ” where, ‘-’ means “deny” and ‘+’ means “allow”. In addition, a single ‘*’ may be used to wildcard various list entries			Reservation owner only	<code>pbs.acl</code>	r, w	r, w
group_list	No longer used.						-	-

Reservation Attributes								
Name	Description	Format	Val / Opt	Value/Option Description	Def Val	Python Type	User	Oper
interactive	Number of seconds that the <code>pbs_rsub</code> command will block while waiting for confirmation or denial of the reservation. See the <code>-l block_time</code> option to the <code>pbs_rsub</code> command.	<i>Integer</i>	Less than zero	The reservation is automatically deleted if it cannot be confirmed in the time specified.	<i>Zero</i>	int	r, w	r, w
			Zero or greater than zero	The reservation is not automatically deleted if it cannot be confirmed in the time specified.				
Mail_Points	Sets the list of events for which mail is sent by the server. Mail is sent to the list of users specified in the <code>Mail_Users</code> attribute. See the <code>m mail_points</code> option to the <code>pbs_rsub</code> command.	<i>String</i> consisting of 1) one or more of the letters “a”, “b”, “c”, “e”, or 2) the string “n”. Cannot use “n” with any other letter	a	Notify when reservation is terminated	“ac”	pbs.mail_points	r, w	r, w
			b	Notify when reservation period begins				
			c	Notify when reservation is confirmed				
			e	Notify when reservation period ends				
			n	Do not send mail. Cannot be used with other letters.				

Reservation Attributes									
Name	Description	Format	Val / Opt	Value/Option Description	Def Val	Python Type	User	Oper	Mgr
Mail_Users	The set of users to whom mail is sent for the reservation events specified in the Mail_Points attribute. See the M mail_list option to the pbs_rsub command.	<i>String</i> . Form: <i>user@host[,user@host,...]</i>			Reservation owner only	pbs.user_list	r, w	r, w	r, w
Priority	No longer used.						-	-	-
reserve_count	The count of occurrences in the standing reservation.	<i>Integer</i>				int	r, w	r, w	r, w
reserve_duration	Reservation duration in seconds. For a standing reservation, this is the duration for one occurrence.	<i>Integer</i>				pbs.duration	r, w	r, w	r, w
reserve_end	The date and time when an advance reservation or the soonest occurrence of a standing reservation ends.	<i>Date</i>				long	r, w	r, w	r, w

Reservation Attributes								
Name	Description	Format	Val / Opt	Value/Option Description	Def Val	Python Type	User	Oper
Reserve_ Name	The name assigned to the reservation during creation, if specified. See the N option to the <code>pbs_rsub</code> command.	<i>String</i> up to 15 characters. First character is alphabetic			None	str	r, w	r, w

Reservation Attributes									
Name	Description	Format	Val / Opt	Value/Option Description	Def Val	Python Type	User	Oper	Mgr
reserve_rule	The rule that describes the recurrence pattern of a standing reservation. See the r option to the pbs_rsub command.	Either of two forms: “ <i>FREQ=freq_spec; COUNT=count_spec; interval_spec</i> ” or “ <i>FREQ=freq_spec; UNTIL=until_spec; interval_spec</i> ”	freq_spec	Frequency with which the standing reservation repeats. Valid values are: <i>WEEKLY DAILY HOURLY</i>	None	str	r, s	r, w	r, w
			count_spec	The exact number of occurrences. Number up to 4 digits in length. Format: <i>integer</i> .	None				
			interval_spec	Specifies interval. Format is one or both of: <i>BYDAY = MO TU WE TH FR SA SU</i> or <i>BYHOUR = 0 1 2 ... 23</i>	None				
			until_spec	Occurrences will start up to but not after date and time specified. Format: <i>YYYYM-MDD[THHMMSS]</i> Year-month-day part and hour-minute-second part separated by a capital <i>T</i> .	None				

Reservation Attributes									
Name	Description	Format	Val / Opt	Value/Option Description	Def Val	Python Type	User	Oper	Mgr
reserve_start	The date and time when the reservation period for the reservation or soonest occurrence begins.	<i>Date</i>			None	long	r, w	r, w	r, w
Resource_List	The list of resources allocated to the reservation. Jobs running in the reservation cannot use in aggregate more than the specified amount of a resource.	<i>String</i> . Form: <i>Resource_List.<resources>=<value></i> , <i>Resource_List.<resources>=<value></i> , ...			None	Dictionary: Resource_List[<resource name>]=<resource value> where <resource> name is any built-in or custom resource	r, w	r, w	r, w
User_List	No longer used.						-	-	-
The following reservation attributes are read-only:									
ctime	The time that the reservation was created.	<i>Date</i> . <i>Seconds since epoch</i> .			None	int	r	r	r
hash-name	No longer used.						-	-	-
mtime	The time that the reservation was last modified.	<i>Date</i>				int	r	r	r

Reservation Attributes								
Name	Description	Format	Val / Opt	Value/Option Description	Def Val	Python Type	User	Oper
Queue	Name of the reservation queue. Jobs that are to use resources belonging to this reservation are submitted to this queue.	Format for an advance reservation: <i>R<unique integer></i> Format for a standing reservation: <i>S<unique integer></i>				pbs.queue	r	r

Reservation Attributes								
Name	Description	Format	Val / Opt	Value/Option Description	Def Val	Python Type	User	Oper
reserve_ID	The reservation identifier.	For an advance reservation: string of the form <i>R<unique integer>.server_name</i> For a standing reservation: string of the form <i>S[unique integer].server_name</i>				str	r	r
reserve_index	The index of the soonest occurrence of a standing reservation.	<i>Integer</i>				int	r	r

Reservation Attributes									
Name	Description	Format	Val / Opt	Value/Option Description	Def Val	Python Type	User	Oper	Mgr
Reserve_Owner	The login name on the submission host of the user who created the reservation.	<i>String. Username@host</i>			Login name of creator	str	r	r	r
reserve_retry	Time at which reservation will be reconfirmed.	<i>Integer. Seconds since epoch.</i>			None	int	r	r	r

Reservation Attributes									
Name	Description	Format	Val / Opt	Value/Option Description	Def Val	Python Type	User	Oper	Mgr
reserve_state	The state of the reservation.		<i>NO RESV_NONE</i>	No reservation yet.	None	reservation state constant: pbs.RESV_STATE_NONE	r	r	r
			<i>UN RESV_UNCONFIRMED</i>	Reservation request is awaiting confirmation.		reservation state constant: pbs.RESV_STATE_UNCONFIRMED			
			<i>CO RESV_CONFIRMED</i>	Resv. confirmed. All occurrences of standing resv. confirmed.		reservation state constant: pbs.RESV_STATE_CONFIRMED			
			<i>WT RESV_WAIT</i>	Unused.		reservation state constant: pbs.RESV_STATE_WAIT			
			<i>TR RESV_TIME_TO_RUN</i>	Start of the reservation period.		reservation state constant: pbs.RESV_STATE_TIME_TO_RUN			
			<i>RN RESV_RUNNING</i>	Resv. period has started; reservation is running.		reservation state constant: pbs.RESV_STATE_RUNNING			
			<i>FN RESV_FINISHED</i>	End of the reservation period.		reservation state constant: pbs.RESV_STATE_FINISHED			
			<i>BD RESV_BEING_DELETE D</i>	Reservation is being deleted.		reservation state constant: pbs.RESV_STATE_BEING_DELETE D			
			<i>DE RESV_DELETED</i>	Reservation has been deleted.		reservation state constant: pbs.RESV_STATE_DELETED			
			<i>DJ RESV_DELETING_JOBS</i>	Jobs belonging to the reservation are being deleted		reservation state constant: pbs.RESV_STATE_DELETING_JOBS			
			<i>DG DEGRADED</i>	Reservation is degraded.		reservation state constant: pbs.RESV_STATE_DEGRADED			

Reservation Attributes									
Name	Description	Format	Val / Opt	Value/Option Description	Def Val	Python Type	User	Oper	Mgr
reserve_substate	The substate of the reservation or occurrence. The substate is used internally by PBS.	<i>Integer</i>			None	int	r	r	r
reserve_type	No longer used.						-	-	-
resv_nodes	The list of each vnode and the resources allocated from it to satisfy the chunks requested for this reservation or occurrence.	<i>String</i> . Form: “(vnode_name:resource=value[:resource=value].. .) [+(vnode_name:resource=value[:resource=value]) +...”			None	pbs.exec_vnode	r	r	r
server	Name of server.	<i>String</i>			None	pbs.server	r	r	r
Variable_List	Not used						-	-	-

6.9 Queue Attributes

Queue attributes are divided into the following groups:

- Those that apply to both execution and routing queues
- Those that apply only to execution queues
- Those that apply only to routing queues

In the following table, Q Type indicates the type of queue to which the attribute applies: R (routing), E (execution):

Queue Attributes									
Name	Description	Format	Q Type	Val / Opt	Value/Option Description	Default Value	Python Type	User	Oper Mgr
The following attributes apply to both routing and execution queues:									
acl_group_enable	Controls whether the queue obeys the access control list defined in the <code>acl_groups</code> queue attribute.	<i>Boolean</i>	R, E		When set to <i>True</i> , the queue limits group access according to the access control list.	<i>False</i> ; all groups allowed access	bool	r	r, w
acl_groups	List of groups which are allowed or denied access to this queue. The groups in the list are groups on the server host, not submitting hosts. List is evaluated left-to-right; first match in list is used.	<i>String</i> . Form: <code>[+/-] group-name[,...]</code>	R, E			None	pbs.acl	r	r, w
acl_host_enable	Controls whether the queue obeys the access control list defined in the <code>acl_hosts</code> queue attribute.	<i>Boolean</i>	R, E		When set to <i>True</i> , the queue limits host access according to the access control list.	<i>False</i> ; all hosts allowed access.	bool	r	r, w

Queue Attributes										
Name	Description	Format	Q Type	Val / Opt	Value/Option Description	Default Value	Python Type	User	Oper	Mgr
acl_hosts	List of hosts from which jobs may be submitted to this queue. List is evaluated left-to-right; first match in list is used.	<i>String</i> . Form: <i>[+ -]hostname[...]</i>	R, E			None	pbs.acl	r	r, w	r, w
acl_user_enabled	Controls whether the queue obeys the access control list defined in the <code>acl_users</code> queue attribute.	<i>Boolean</i>	R, E		When set to <i>True</i> , the queue limits user access according to the access control list.	<i>False</i> ; all users allowed access	bool	r	r, w	r, w
acl_users	List of users allowed or denied access to this queue. List is evaluated left-to-right; first match in list is used.	<i>String</i> . Form: <i>[+ -]user [@host][,...]</i>	R, E			None	pbs.acl	r	r, w	r, w
enabled	Specifies whether this queue accepts new jobs.	<i>Boolean</i>	R, E	<i>True</i>	The queue is <i>enabled</i> . The queue accepts new jobs; new jobs can be enqueued.	<i>False</i>	bool	r	r, w	r, w
				<i>False</i>	The queue does not accept new jobs.					
from_route_only	Specifies whether this queue accepts jobs only from routing queues, or from both execution and routing queues.	<i>Boolean</i>	R, E	<i>True</i>	This queue accepts jobs only from routing queues.	<i>False</i>	bool	r	r	r, w
				<i>False</i>	This queue accepts jobs from both execution and routing queues.					
max_array_size	The maximum number of sub-jobs that are allowed in an array job.	<i>Integer</i>	R, E			No limit	int	r	r, w	r, w

Queue Attributes										
Name	Description	Format	Q Type	Val / Opt	Value/Option Description	Default Value	Python Type	User	Oper	Mgr
max_queueable	Old limit attribute. Incompatible with new limit attributes. The maximum number of jobs allowed to reside in the queue at any given time.	<i>Integer</i>	R, E			No limit	int	r	r, w	r, w
max_queued	Limit attribute. The maximum number of jobs allowed to be queued in or running from this queue. Can be specified for projects, users, groups, or all. Cannot be used with old limit attributes.	Limit specification. See Chapter 7, "Formats", on page 403	R, E			None	None	r	r, w	r, w
max_queued_res.<resource>	Limit attribute. The maximum amount of the specified resource allowed to be allocated to jobs queued in or running from this queue. Can be specified for projects, users, groups, or all. Cannot be used with old limit attributes.	Limit specification. See Chapter 7, "Formats", on page 403	R, E			None	None	r	r, w	r, w
max_running	Old limit attribute. Incompatible with new limit attributes. For an execution queue, this is the largest number of jobs allowed to be running at any given time. For a routing queue, this is the largest number of jobs allowed to be transiting from this queue at any given time.	<i>Integer</i>	R, E			None	int	r	r, w	r, w

Queue Attributes										
Name	Description	Format	Q Type	Val / Opt	Value/Option Description	Default Value	Python Type	User	Oper	Mgr
node_group_key	Specifies the resources to use for placement sets (node grouping). Overrides server's node_group_key attribute. Resources used must be of type string_array.	string_array. Comma-separated list of resource names. When specifying multiple resources, enclose value in double quotes.	R, E			None	pbs.node_group_key	r	r, w	r, w
Priority	The priority of this queue compared to other queues of the same type in this PBS complex. The value of priority has no meaning for routing queues.	Integer. -1024 to 1023	R, E			None	int	r	r, w	r, w
queue_type	The type of this queue. This attribute must be explicitly set at queue creation.	String	R, E	e , execution	Execution queue	None	PBS queue type constants: pbs.QUEUETYPE_EXECUTION or pbs.QUEUETYPE_ROUTE	r	r, w	r, w
				r , route	Routing queue					
require_cred	Specifies the credential type required. All jobs submitted to the named queue without the specified credential will be rejected. Not supported under Windows.	String	R, E	krb5		unset	str	r	r	r, w
				dce						

Queue Attributes									
Name	Description	Format	Q Type	Val / Opt	Value/Option Description	Default Value	Python Type	User	Oper Mgr
<code>require_cred_enable</code>	Specifies whether the credential authentication method specified in the <code>require_cred</code> queue attribute is required for this queue. Not supported under Windows.	<i>Boolean</i>	R, E		When set to <i>True</i> , the credential authentication method is required	<i>False</i>	bool	r	r, w
<code>resources_default</code>	The list of default resource values which are set as limits for a job residing in this queue and for which the job did not specify a limit. If not set, the default limit for a job is determined by the first of the following attributes which is set: server's <code>resources_default</code> , queue's <code>resources_max</code> , server's <code>resources_max</code> . If none of these is set, the job gets unlimited resource usage.	<i>String</i> . Form: <i>resources_default.<resource_name>=<value></i> , <i>resources_default.<resource_name>=<value></i> , ...	R, E			None	Dictionary: <i>resources_default[<resource_name>]=<value></i> where <i><resource_name></i> is any built-in or custom resource	r	r, w
<code>resources_max</code>	The maximum amount of each resource which can be requested by a single job in this queue. The queue value supersedes any server wide maximum limit.	<i>String</i> . Form: <i>resources_max.<resource_name>=<value></i> , <i>resources_max.<resource_name>=<value></i> , ...	R, E			None; infinite usage	Dictionary: <i>resources_max[<resource_name>]=<value></i> where <i><resource_name></i> is any built-in or custom resource	r	r, w

Queue Attributes										
Name	Description	Format	Q Type	Val / Opt	Value/Option Description	Default Value	Python Type	User	Oper	Mgr
resources_min	The minimum amount of each resource that can be requested by a single job in this queue.	<i>String</i> . Form: <i>resources_max.</i> <resource_name>=<value>, <i>resources_max.</i> <resource_name>=<value>, ...	R, E			Zero usage	Dictionary: resources_min[<resource_name>]=<value> where <resource_name> is any built-in or custom resource	r	r, w	r, w
started	Specifies whether jobs in this queue can be scheduled for execution.	<i>Boolean</i>	R, E		Set to <i>True</i> : jobs in this queue can run	<i>False</i>	bool	r	r, w	r, w
state_count	The number of jobs in each state currently residing in this queue.	<i>String</i> . Form: <i>transiting</i> =<val>, <i>exiting</i> =<val>, ...	R, E			None	pbs.state_count	r	r	r
total_jobs	The number of jobs currently residing in this queue.	<i>Integer</i>	R, E			None	int	r	r	r
The following attributes apply only to execution queues:										
checkpoint_min	Specifies the minimum number of minutes of CPU time or wall-time allowed between checkpoints of a job. If a user specifies a time less than this value, this value is used instead. The value given in <i>checkpoint_min</i> is used for both CPU minutes and wall-time minutes.	<i>Integer</i>	E			None	pbs.duration	r	r, w	r, w

Queue Attributes										
Name	Description	Format	Q Type	Val / Opt	Value/Option Description	Default Value	Python Type	User	Oper	Mgr
default_chunk	The list of resources which will be inserted into each chunk of a job's select specification if the corresponding resource is not specified by the user. This provides a means for a site to be sure a given resource is properly accounted for even if not specified by the user.	<i>String.</i> Form: <i>default_chunk.<resource>=<value></i> , <i>default_chunk.<resource>=<value></i> , ...	E			None	Dictionary: <i>default_chunk[<resource name>]=<value></i> where <i><resource name></i> is any built-in or custom resource	r	r, w	r, w
hasnodes	Describes whether this queue has associated vnodes.	<i>Boolean</i>	E		This attribute is set to <i>True</i> if there are vnodes associated with this queue.	<i>False</i> ; no vnodes are associated with this queue	bool	r	r	r, i
kill_delay	The time delay between sending SIGTERM and SIGKILL when a <i>qdel</i> command is issued against a running job.	<i>Integer. Seconds.</i> Must be greater than or equal to <i>zero</i> .	E			<i>10 seconds</i>	<i>pbs.duration</i>	r	r, w	r, w
max_group_res	Old limit attribute. Incompatible with new limit attributes. The maximum amount of the specified resource that any single group may consume in a complex.	<i>String.</i> Form: <i>max_group_res.resource_name=value</i> <i>Example: set server max_group_res.ncpus=6</i>	E	Any PBS resource, e.g. "ncpus", "mem", "pmem", etc.		None	Dictionary: <i>max_group_res[<resource name>]=<value></i> where <i><resource name></i> is any built-in or custom resource	r	r, w	r, w

Queue Attributes									
Name	Description	Format	Q Type	Val / Opt	Value/Option Description	Default Value	Python Type	User	Oper Mgr
max_group_res_soft	Old limit attribute. Incompatible with new limit attributes. The soft limit on the amount of the specified resource that any single group may consume in a complex. If a group is consuming more than this amount of the specified resource, their jobs are eligible to be preempted by jobs from groups who are not over their soft limit.	<i>String</i> . Form: <i>max_group_res_soft.resource_name=value</i> <i>Example: set server max_group_res_soft.ncpus=3</i>	E	Any valid PBS resource, e.g. "ncpus", "mem", "pmem", etc.		None	Dictionary: max_group_res_soft[<resource name>]=<value> where <resource name> is any built-in or custom resource	r	r, w
max_group_run	Old limit attribute. Incompatible with new limit attributes. The maximum number of jobs owned by a group that are allowed to be running from this queue at one time.	<i>Integer</i>	E			None	int	r	r, w
max_group_run_soft	Old limit attribute. Incompatible with new limit attributes. The maximum number of jobs owned by users in a single group that are allowed to be running from this queue at one time. If a group has more than this number of jobs running, their jobs are eligible to be preempted by jobs from groups who are not over their soft limit.	<i>Integer</i>	E			None	int	r	r, w

Queue Attributes									
Name	Description	Format	Q Type	Val / Opt	Value/Option Description	Default Value	Python Type	User	Oper Mgr
max_run	Limit attribute. The maximum number of jobs allowed to be running from this queue. Can be specified for projects, users, groups, or all. Cannot be used with old limit attributes.	Format: Limit specification. See Chapter 7, "Formats", on page 403	E			None	None	r	r, w
max_run_res.<resource>	Limit attribute. The maximum amount of the specified resource allowed to be allocated to jobs running from this queue. Can be specified for projects, users, groups, or all. Cannot be used with old limit attributes.	Format: Limit specification. See Chapter 7, "Formats", on page 403 .	E			None	None	r	r, w
max_run_res_soft.<resource>	Limit attribute. Soft limit on the amount of the specified resource allowed to be allocated to jobs running from this queue. Can be specified for projects, users, groups, or all. Cannot be used with old limit attributes.	Format: Limit specification. See Chapter 7, "Formats", on page 403 .	E			None	None	r	r, w
max_run_soft	Limit attribute. Soft limit on the number of jobs allowed to be running from this queue. Can be specified for projects, users, groups, or all. Cannot be used with old limit attributes.	Format: Limit specification. See Chapter 7, "Formats", on page 403 .	E			None	None	r	r, w

Queue Attributes									
Name	Description	Format	Q Type	Val / Opt	Value/Option Description	Default Value	Python Type	User	Oper Mgr
max_user_res	Old limit attribute. Incompatible with new limit attributes. The maximum amount of the specified resource that any single user may consume.	<i>String</i> . Form: <i>max_user_resource_name=value</i> Example: <i>set server max_user_res.ncpus=6</i>	E	any PBS resource, e.g. "ncpus", "mem", "pmem", etc		None	Dictionary: max_user_res[<resource name>] =<value> where <resource name> is any built-in or custom resource	r	r, w, r, w
max_user_res_soft	Old limit attribute. Incompatible with new limit attributes. The soft limit on the amount of the specified resource that any single user may consume. If a user is consuming more than this amount of the specified resource, their jobs are eligible to be preempted by jobs from users who are not over their soft limit.	<i>String</i> . Form: <i>max_user_res_soft.resource_name=value</i> Example: <i>set server max_user_res_soft.ncpus=3</i>	E	any valid PBS resource, e.g. "ncpus", "mem", "pmem", etc		None	Dictionary: max_user_res_soft[<resource name>] =<value> where <resource name> is any built-in or custom resource	r	r, w, r, w
max_user_run	Old limit attribute. Incompatible with new limit attributes. The maximum number of jobs owned by a single user that are allowed to be running from this queue at one time.	<i>Integer</i>	E			None	int	r	r, w, r, w

Queue Attributes									
Name	Description	Format	Q Type	Val / Opt	Value/Option Description	Default Value	Python Type	User	Oper Mgr
max_user_ run_soft	Old limit attribute. Incompatible with new limit attributes. The soft limit on the number of jobs owned by a single user that are allowed to be running from this queue at one time. If a user has more than this number of jobs running, their jobs are eligible to be preempted by jobs from users who are not over their soft limit.	<i>Integer</i>	E			None	int	r	r, w
resources_ assigned	The total for each kind of resource allocated to jobs running from this queue.	<i>String. Form:</i> <i>resources_assigned.<res>=<val></i> <i><new-line>resources_</i> <i>assigned.<res>=</i> <i><val><new-</i> <i>line>...</i>	E				Dictionary: resources_assigned[<resource name>]=<value> where <resource name> is any built-in or custom resource	r	r r
resources_ available	The list of resources and amounts available to jobs running in this queue. The sum of the resource of each type used by all jobs running from this queue cannot exceed the total amount listed here. See qmgr (1B).	<i>String. Form:</i> <i>resources_available.<resource_name>=<value></i> <i><new-</i> <i>line>resources_</i> <i>available.<resource_</i> <i>name>=<value></i> <i><newline>...</i>	E				Dictionary: resources_available[<resource name>]=<value> where <resource name> is any built-in or custom resource	r	r, w r, w

Queue Attributes									
Name	Description	Format	Q Type	Val / Opt	Value/Option Description	Default Value	Python Type	User	Oper Mgr
The following attributes apply only to routing queues:									
alt_router	No longer used.							-	-
route_destinations	The list of destinations to which jobs may be routed.	<i>String</i> of comma-separated strings. Form: <i>queue_name</i> [<i>@server_host</i> [: <i>port</i>]] Example: Q1, Q2@remote, Q3@remote:15501	R			None. Must be set to at least one valid destination	pbs.route_destinations	r	r, w
route_held_jobs	Specifies whether jobs in the held state can be routed from this queue.	<i>Boolean</i>	R		If <i>True</i> , jobs with a hold can be routed from this queue.	<i>False</i>	bool	r	r, w
route_lifetime	The maximum time a job is allowed to reside in a routing queue. If a job cannot be routed in this amount of time, the job is aborted.	<i>Integer. Seconds.</i>	R	>0	Number of seconds specified	Unset; infinite	pbs.duration	r	r, w
				0	Infinite				
				unset	Infinite				
route_retry_time	Time delay between routing retries. Typically used when the network between servers is down.	<i>Integer. Seconds.</i>	R			30 seconds	pbs.duration	r	r, w
route_waiting_jobs	Specifies whether jobs whose <i>execution_time</i> attribute value is in the future can be routed from this queue.	<i>Boolean</i>	R		If set to <i>True</i> , jobs with a future <i>execution_time</i> attribute can be routed from this queue.	<i>False</i>	bool	r	r, w

6.10 Vnode Attributes

Vnode attributes are divided into the following groups:

- Those that can be set by an operator or manager
- Those that are read-only

Vnode Attributes									
Name	Description	Format	Val / Opt	Value/Option Description	Def Val	Python Type	Usr	Op	Mgr
The following attributes can be set by a manager or operator:									
comment	Information about this vnode. This attribute may be set by the manager to any string to inform users of any information relating to the node. If this attribute is not explicitly set, the PBS Server will use the attribute to pass information about the node status, specifically why the node is down. If the attribute is explicitly set by the manager, it will not be modified by the Server.	String			None	str	r	r	r, w
current_aoe	This attribute identifies the AOE currently instantiated on this vnode. Case-sensitive. Cannot be set on Server's host.	String			Unset	str	r	r	r, w
hpcbp_enable	Enables HPCBP features in Linux MOM.	Boolean		When set to <i>True</i> , HPCBP features are enabled.	False	bool	r	r	r, w

Vnode Attributes									
Name	Description	Format	Val / Opt	Value/Option Description	Def Val	Python Type	Usr	Or	Mgr
hpcbp_stage_protocol	Protocol and port number for staging files to and from HPC Basic Profile Server.	<i>String</i>			<i>scp</i>	<i>str</i>	r	r	r, w
hpcbp_user_name	User account with limited privilege, used for requesting job/node status from HPC Basic Profile Server.	<i>String</i>			None	<i>str</i>	r	r	r, w
hpcbp_webserver_address	URL for HPC Basic Profile Server .	<i>String</i> . The address must start with <i>https://</i> .			None	<i>str</i>	r	r	r, w
lictype	No longer used. (Was deprecated .) Gone from .c file.					None	-	-	-
max_group_run	The maximum number of jobs owned by any users in a single group allowed to run on this vnode at one time.	<i>Integer</i>			None	<i>int</i>	r	r, w	r, w
max_running	The maximum number of jobs allowed to be run on this vnode at any given time.	<i>Integer</i>			None	<i>int</i>	r	r, w	r, w

Vnode Attributes									
Name	Description	Format	Val / Opt	Value/Option Description	Def Val	Python Type	Usg	Or	Mgt
max_user_run	The maximum number of jobs owned by a single user allowed to run on this vnode at one time.	<i>Integer</i>			None	int	r	r, w	r, w
Mom	Hostname of host on which MOM daemon runs. Can be explicitly set by Manager only via <code>qmgr</code> , and only at vnode creation. The server can set this to the FQDN of the host on which MOM runs, if the vnode name is the same as the hostname.	<i>String</i>		.	Value of vnode resource (vnode name.)	str	r	r	r, w
no_multinode_jobs	Controls whether jobs which request more than one chunk are allowed to execute on this vnode. Used for cycle harvesting.	<i>Boolean</i>		When set to <i>True</i> , jobs requesting more than one chunk are not allowed to execute on this vnode	<i>False</i>	bool	r	r	r, w
ntype	The type of the vnode.	<i>String</i>	PBS	Normal vnode, not Globus.	<i>PBS</i>	int	r	r	r, w
			globus	PBS no longer supports Globus. The Globus functionality has been removed from PBS.					
pnames	The list of resources being used for placement sets. Not used for scheduling; advisory only.	<i>String</i> . Comma-separated list of resource names.			None	str	r	r	r, w

Vnode Attributes									
Name	Description	Format	Val / Opt	Value/Option Description	Def Val	Python Type	Usg	On	Mgt
Port	Port number on which MOM daemon listens. Can be explicitly set only via <code>qmgr</code> , and only at vnode creation.	<i>Integer</i>			15002	int	-	r, w	r, w
Priority	The priority of this vnode compared with other vnodes.	<i>Integer</i>	<i>[-1024, +1023]</i> inclusive		None	int	r	r, w	r, w
provision_enable	Controls whether this vnode can be provisioned. If set to <i>True</i> , this vnode may be provisioned. Cannot be set on Server's host.	<i>Boolean</i>			Unset	bool	r	r	r, w
queue	The queue with which this vnode is associated. Each vnode can be associated with at most 1 queue. Queues can be associated with multiple vnodes. Any jobs in a queue that has associated vnodes can run only on those vnodes. If a vnode has an associated queue, only jobs in that queue can run on that vnode.	<i>String</i>	Name of queue	Only jobs in specified queue may run on this vnode.	None	pbs.queue	r	r	r, w
			Unset	Any job in any queue that does not have associated vnodes can run on this vnode.					

Vnode Attributes								
Name	Description	Format	Val / Opt	Value/Option Description	Def Val	Python Type	Usg	Opt Mgt
resources_available	The list of resources and the amounts available on this vnode. If not explicitly set, the amount shown is that reported by the pbs_mom running on the vnode. If a resource value is explicitly set, that value is retained across restarts.	<i>String</i> . Form: <i>resources_available</i> .<resource name>=<value>, <i>resources_available</i> .<resource name> = <value>, ...			None	Dictionary: resources_available['<resource>'] = <val> where <resource> is any custom or built-in resource	r	r, w
resv_enabled	Controls whether the vnode can be used for advance and standing reservations. Reservations are incompatible with cycle harvesting.	<i>Boolean</i>		When set to <i>True</i> , this vnode can be used for reservations. Existing reservations are honored when this attribute is changed from <i>True</i> to <i>False</i> .	<i>True</i>	bool	r	r, w

Vnode Attributes									
Name	Description	Format	Val / Opt	Value/Option Description	Def Val	Python Type	Usr	Opr	Mgr
sharing	Specifies whether more than one job at a time can use the resources of the vnode or the vnode's host. Either (1) the vnode or host is allocated exclusively to one job, or (2) the vnode's or host's unused resources are available to other jobs. Behavior is determined by a combination of vnode sharing attribute and job's placement directive.	String. Example: <i>vnode-name: sharing=force_excl</i>	<i>default_shared</i>	Defaults to <i>shared</i>	<i>default_shared</i>	int	r	r, w	r, w
			<i>default_excl</i>	Defaults to <i>exclusive</i>					
			default_exclhost	Entire host is assigned to the job unless the job's sharing request specifies otherwise					
			<i>ignore_excl</i>	Overrides any job <i>place=excl</i> setting					
			<i>force_excl</i>	Overrides any job <i>place=shared</i> setting					
			force_exclhost	The entire host is assigned to the job, regardless of the job's sharing request					
			Unset	Defaults to <i>shared</i>					
Behavior of vnode is shown in table below:									
Value of sharing	Placement Request (<i>-lplace=</i>)								
	Vnode			Host					
	not specified	<i>place=shared</i>	<i>place=excl</i>	<i>place=exclhost</i>	<i>place!=exclhost</i>				
not set	shared	shared	exclusive	exclusive	depends on place				
<i>default_shared</i>	shared	shared	exclusive	exclusive	depends on place				
<i>default_excl</i>	exclusive	shared	exclusive	exclusive	depends on place				
<i>default_exclhost</i>	exclusive	shared	exclusive	exclusive	depends on place				
<i>ignore_excl</i>	shared	shared	shared	shared	not exclusive				
<i>force_excl</i>	exclusive	exclusive	exclusive	exclusive	not exclusive				

Vnode Attributes									
Name	Description	Format	Val / Opt	Value/Option Description	Def Val	Python Type	Usr	Or	Mgt
state	Shows or sets the state of the vnode.	<i>String</i> . Comma-separated list of one or more states: <i>state</i> , <i>state</i> ...]	<i>free</i>	Node is up and capable of accepting new job(s). Cannot be combined with other states.		int	r	r	r
			<i>offline</i>	Jobs are not to be assigned to this vnode. Can combine: <i>busy</i> , <i>job-busy</i> , <i>job-exclusive</i> , <i>resv_exclusive</i> .			r	r, w	r, w
			<i>down</i>	Node is not responding to queries from the Server. Cannot be combined with <i>free</i> .			r	r	r
			<i>job-busy</i>	All CPUs on the vnode are allocated to jobs. Can combine with: <i>offline</i> , <i>resv_exclusive</i> .			r	r	r
			<i>job-exclusive</i>	Entire vnode is exclusively allocated to one job at the job's request. Can combine with <i>offline</i> and <i>resv_exclusive</i>			r	r	r
			<i>resv-exclusive</i>	Running reservation has requested exclusive use of vnode. Can combine with <i>job-exclusive</i> and <i>offline</i>					
			<i>busy</i>	Vnode is reporting load average greater than allowed max. Can combine with <i>offline</i> .			r	r	r
			<i>provisioning</i>	Vnode is being provisioned. Cannot be combined with any other states.			r	r	r
			<i>wait-provisioning</i>	Vnode needs to be provisioned, but can't: limit reached for concurrent provisioning vnodes. Cannot be combined with other states. See max_concurrent_provision .					
			<i>stale</i>	Vnode was previously reported to server, but is no longer reported to server. Cannot combine with <i>free</i> .			r	r	r
			<i>state-unknown</i>	The Server has never been able to contact the vnode. Either MOM is not running on the vnode, the vnode hardware is down, or there is a network problem.			r	r	r

Vnode Attributes									
Name	Description	Format	Val / Opt	Value/Option Description	Def Val	Python Type	Us	Or	Mnt
The following attributes are read-only:									
in_multi_vnode_host	Specifies whether a vnode is part of a multi-vnoded host. Used internally. Do not set.	<i>Integer</i>	<i>unset</i>	Not part of a multi-vnode host		int			r, w
			<i>1</i>	Part of a multi-vnode host					
jobs	List of jobs running on the vnode.	<i>String</i> . Form: <i>#/jobid,...</i> where <i>#</i> is the number of the processor.				str	r	r	r
license	Indicates whether a vnode is socket-licensed. Set by PBS.	<i>character</i>	<i>ll</i>	Indicates that vnode is socket licensed.	Unset	str	r	r	r
license_info	Indicates number of socket licenses assigned to vnode. Set by PBS.	<i>Integer</i>			Unset	int	r	r	r
pbs_version	The version of PBS for this MOM	<i>String</i>			None	str	r	r	r
pcpus	The number of physical CPUs on the vnode. This is set to the number of CPUs available when MOM starts. For a multiple-vnode MOM, only the natural vnode has pcpus.	<i>Integer</i>			Number of CPUs on startup	int	r	r	r

Vnode Attributes									
Name	Description	Format	Val / Opt	Value/Option Description	Def Val	Python Type	Unit	Opt	Mgr
resv	List of advance and standing reservations pending on the vnode.	<i>String</i> . Comma-separated list of reservation IDs. Form: <i>RNNNN</i> . <server>, <i>RNNNN</i> . <server>, ...			None	str	r	r	r
resources_assigned	The total amount of each resource allocated to jobs running on this vnode.	<i>String</i> . Form: <i>resources_assigned</i> .<resource>=<value>[,<resources_assigned.<resource>=<value>			None	Dictionary: resources_available['<resource>'] = <val> where <resource> is any custom or built-in resource	r	r	r

Vnode Attributes									
Name	Description	Format	Val / Opt	Value/Option Description	Def Val	Python Type	Ustr	Opt	Mgr
topology_info	Contains information intended to be used in hooks. Visible in and usable by hooks only.	<i>XML string</i>			Unset	str	-	-	-

6.11 Job Attributes

Job attributes are divided into the following groups:

- Those that can be set by users, operators, or managers
- Those that are read-only

Job Attributes									
Name	Description	Format	Val / Opt	Value/Option Description	Def Val	Python Type	Ustr	Opt	Mgr
The following job attributes can be set by users, operators, or managers:									
Account_Name	Account to which the job is charged.	<i>String</i>			None	str	r, w	r, w	r, w

Job Attributes										
Name	Description	Format	Val / Opt	Value/Option Description	Def Val	Python Type	U	sr	Op	M
argument_list	JSDL-encoded listing of arguments to job's executable.	Example: if arguments are "A B": <jsd1-hpcpa:Argument>A</jsd1-hpcpa:Argument><jsd1-hpcpa:Argument>B</jsd1-hpcpa:Argument>			None	str	r, w	r, w	r, w	
array	Indicates whether this is a job array.	Boolean		Set to <i>True</i> if this is an array job.	False	bool	r, s	r	r	
array_indices_submitted	Applies only to job arrays. Complete list of indices of subjobs given at submission time.	String. Given as range, e.g. 1–100			None	pbs.range	r, s	r	r	
block	Specifies whether qsub will wait for the job to complete, and return the exit value of the job.	Boolean			False	int	r, s	r	r	

Job Attributes									
Name	Description	Format	Val / Opt	Value/Option Description	Def Val	Python Type	Usr	Opt	Mgt
Checkpoint	Determines when the job will be checkpointed. An \$action script is required to checkpoint the job.	<i>String</i>	c	Checkpoint at intervals, measured in CPU time, set on job's execution queue. If no interval set at queue, job is not checkpointed.	u	pbs.checkpoint	r, w	r, w	r, w
			c = minutes of CPU time	Checkpoint at intervals of specified number of minutes of job CPU time. This value must be > 0. If interval specified is less than that set on job's execution queue, queue's interval is used. Format: <i>integer</i> .					
			w	Checkpoint at intervals, measured in walltime, set on job's execution queue. If no interval set at queue, job is not checkpointed.					
			w = minutes of walltime	Checkpoint at intervals of the specified number of minutes of job walltime. This value must be greater than zero. If the interval specified is less than that set on the execution queue in which the job resides, the queue's interval is used. Format: <i>integer</i> .					
			n	No checkpointing.					
			s	Checkpoint only when the server is shut down.					
			u	Unset. Defaults to behavior when interval argument is set to s.					
comment	Comment about job. Informational only.	<i>String</i>			None	str	r	r, w	r, w

Job Attributes									
Name	Description	Format	Val / Opt	Value/Option Description	Def Val	Python Type	Usr	Opt	Mgt
depend	Specifies inter-job dependencies.	<i>String</i> . Form: <i>type:arg_list [,type:arg_list ...]</i> <i>arg_list</i> is one or more PBS job IDs in the form: <i>jobid[:jobid ...]</i> Must be enclosed in double quotes if it contains commas. Example: “ <i>type:jobid [,jobid...]</i> ”	after: <i>arg_list</i>	This job may run at any point after all jobs in <i>arg_list</i> have started execution.	None; no dependencies	pbs.depend	r, w	r, w	r, w
			afterok: <i>arg_list</i>	This job may run only after all jobs in <i>arg_list</i> have terminated with no errors.					
			afternotok: <i>arg_list</i>	This job may run only after all jobs in <i>arg_list</i> have terminated with errors.					
			afterany: <i>arg_list</i>	This job can run after all jobs in <i>arg_list</i> have finished execution, with or without errors. This job will not run if a job in the <i>arg_list</i> was killed.					
			before: <i>arg_list</i>	Jobs in <i>arg_list</i> may start once this job has started.					
			beforeok: <i>arg_list</i>	Jobs in <i>arg_list</i> may start once this job terminates without errors.					
			beforenotok: <i>arg_list</i>	If this job terminates execution with errors, then jobs in <i>arg_list</i> may begin.					
			beforeany: <i>arg_list</i>	Jobs in <i>arg_list</i> may begin execution once this job terminates execution, with or without errors.					
			on: count	This job may run after <i>count</i> dependencies on other jobs have been satisfied. This type is used with one of the <i>before</i> types listed. Count is an integer greater than 0.					

Job Attributes									
Name	Description	Format	Val / Opt	Value/Option Description	Def Val	Python Type	Usr	Opt	Mbr
eligible_time	The amount of wall clock wait time a job has accrued while the job is blocked waiting for resources. For a job currently accruing <i>eligible_time</i> , if we were to add enough of the right type of resources, the job would start immediately. Viewable via <code>qstat -f</code> .	<i>Duration</i>			Zero	pbs.duration	r	r, w	r, w
Error_Path	The final path name for the file containing the job's standard error stream. If the output path is specified, but does not include a file-name, the default file-name is <code>jobid.ER</code> . If the path name is not specified, the default filename is <code><job name>.e<sequence number></code> . See the <code>qsub</code> and <code>qalter</code> commands.	<i>String</i> . Form: <i>[host-name:]path-name</i>	path_name	Path is relative to the current working directory of command executing on current host.	Path is current working directory where <code>qsub</code> is run. File-name is <code><job name>.e<job number></code> .	str	r, w	r, w	r, w
			path_name	Path is absolute path on current host where command is executing.					
			host-name:path_name	Path is relative to user's home directory on specified host.					
			host-name:path_name	Path is absolute path on named host.					
			No path	Path is current working directory where <code>qsub</code> is executed.					

Job Attributes									
Name	Description	Format	Val / Opt	Value/Option Description	Def Val	Python Type	U sr	O pt	M gr
executable	JSDL-encoded listing of job's executable.	Example: if the executable is ping: <jSDL-hpcpa:Executable>ping</jSDL-hpcpa:Executable>			None	str	r, w	r, w	r, w
Execution_ Time	Time after which the job may execute. Before this time, the job remains queued in the (W)ait state. Can be set when stage-in fails and PBS moves job start time out 30 minutes to allow user to fix problem.	<i>Datetime</i> . See Chapter 7, "Formats", on page 403 .			Unset; no delay	int	r, w	r, w	r, w

Job Attributes									
Name	Description	Format	Val / Opt	Value/Option Description	Def Val	Python Type	Usr	Opt	Mgt
exec_host	If the job is running, this is set to the name of the host or hosts on which the job is executing.	<i>String</i> . Form: <i>host/N[*C][+...]</i> , where “host” is the name of the host, “N” is task slot number starting at 0, on that node, and “C” is the number of CPUs allocated to the job. “*C” does not appear if its value is one.			None	pbs.exec_host	r	r, i	r, i

Job Attributes									
Name	Description	Format	Val / Opt	Value/Option Description	Def Val	Python Type	U _{sr}	O _{pt}	M _{gr}
<code>exec_vnode</code>	<p>List of chunks for the job. Each chunk shows the name of the vnode(s) from which it is taken, along with the host-level, consumable resources allocated from that vnode, and any AOE provisioned on this vnode for this job.</p> <p>If a vnode is allocated to the job but no resources from the vnode are used by the job, the vnode name appears alone.</p> <p>If a chunk is split across vnodes, the name of each vnode and its resources appear inside one pair of parentheses, joined with a plus (“+”) sign.</p>	<p>Each chunk is enclosed in parentheses. Chunks are connected by plus signs. For a job which requested two chunks satisfied by resources from three vnodes, <code>exec_vnode</code> is:</p> <p><code>(vnodeA:ncpus=N:mem=X) + (nodeB:ncpus=P:mem=Y+nodeC:mem=Z)</code>.</p>			None	<code>pbs.exec_vnode</code>	r	r, w	r, w

Job Attributes									
Name	Description	Format	Val / Opt	Value/Option Description	Def Val	Python Type	U sr	O pt	M gr
group_list	A list of group names used to determine the group under which the job runs. When a job runs, the server selects a group name from the list according to the following ordered set of rules: 1. Select the group name for which the associated host name matches the name of the server host. 2. Select the group name which has no associated host name. 3. Use the login group for the user name under which the job will be run.	<i>String</i> . Form: <i>group_name</i> [<i>@host</i>] [<i>,group_name</i> [<i>@host</i>]...] Must be enclosed in double quotes if it contains commas.			None	pbs.group_list	r, w	r, w	r, w
Hold_Types	The set of holds currently applied to the job. If the set is not null, the job will not be scheduled for execution and is said to be in the hold state. The hold state takes precedence over the wait state.	<i>String</i> , made up of the letters 'n', 'o', 'p', 's', 'u'	<i>n</i>	No hold	<i>n</i>	pbs.hold_types	r, w	r, w	r, w
			<i>o</i>	Other hold					
			<i>p</i>	Bad password					
			<i>s</i>	System hold					
			<i>u</i>	User hold					

Job Attributes									
Name	Description	Format	Val / Opt	Value/Option Description	Def Val	Python Type	U sr	O pr	M gr
interactive	Specifies whether the job is interactive. Can be set, but not altered, by unprivileged user.	<i>Boolean</i>		Set to <i>True</i> if this is an interactive job.	<i>False</i>	int	r, w	r	r
Job_Name	The job name. See the <code>qalter</code> and <code>qsub</code> commands.	<i>String</i> up to 15 characters, first character must be alphabetic or numeric			Base name of job script, or STDIN	str	r, w	r, w	r, w
Join_Path	Specifies whether the job's standard error and standard output streams are to be merged and placed in the file specified in the <code>Output_Path</code> job attribute.	<i>Boolean</i>		When set to <i>True</i> , the job's standard error and standard output streams are merged.	<i>False</i>	pbs.join_path	r, w	r, w	r, w

Job Attributes									
Name	Description	Format	Val / Opt	Value/Option Description	Def Val	Python Type	Usr	Opt	Mgt
Keep_Files	Specifies whether the standard output and/or standard error streams are retained on the execution host in the job's staging and execution directory after the job has executed. Otherwise these files are returned to the submission host. Keep_Files overrides the Output_Path and Error_Path attributes.	<i>String</i> . Can be “o”, “e”, “oe”, “eo”, or “n”.	<i>o</i>	The standard output stream is retained. The filename is: <code>job_name.o<sequence number></code>	<i>n</i>	pbs.keep_files	r, w	r, w	r, w
			<i>e</i>	The standard error stream is retained. The filename is: <code>job_name.e<sequence number></code>					
			<i>eo, oe</i>	Both standard output and standard error streams are retained.					
			<i>n</i>	Neither stream is retained. Files returned to submission host.					
Mail_Points	Specifies state changes for which the server sends mail about the job.	<i>String</i> . Can be any of “a”, “b”, “e”, or “n”.	<i>a</i>	Mail is sent on job abort	<i>a</i>	pbs.mail_points	r, w	r, w	r, w
			<i>b</i>	Mail is sent at beginning of job					
			<i>e</i>	Mail is sent at end of job					
			<i>n</i>	No mail is sent. Cannot be used with other options.					
Mail_Users	The set of users to whom mail is sent when the job makes state changes specified in the Mail_Points job attribute.	<i>String</i> . Form: “ <i>user@host[,user@host]</i> ” Must be enclosed in double quotes if it contains commas.			Job owner only	pbs.email_list	r, w	r, w	r, w
no_stdio_sockets	Not used.						-	-	-

Job Attributes									
Name	Description	Format	Val / Opt	Value/Option Description	Def Val	Python Type	U sr	O pr	M gr
Output_Path	The final path name for the file containing the job's standard output stream. If the output path is specified, but does not include a filename, the default filename is <code>jobid.OU</code> . If the path name is not specified, the default filename is <code><job name>.o<job number></code> . See the <code>qsub</code> and <code>qalter</code> commands.	<i>String</i> . Form: <code>[host-name:]path-name</code>	<code>path_name</code> ; relative path	Path is relative to the current working directory of command executing on current host.	Path is current working directory where <code>qsub</code> is run. Filename is <code><job name>.o<job number></code> .	str	r, w	r, w	r, w
			<code>path_name</code> ; absolute path	Path is absolute path on current host where command is executing.					
			<code>host-name:path_name</code> ; relative path	Path is relative to user's home directory on specified host.					
			<code>host-name:path_name</code> ; absolute path	Path is absolute path on named host.					
			No path	Path is current working directory where <code>qsub</code> is executed.					
Priority	The scheduling priority for the job. Higher values indicate greater priority.	<i>Integer</i> . Form: <code>[+ -]nnnn</code>	<code>[-1024, +1023]</code> inclusive		Unset	int	r, w	r, w	r, w

Job Attributes									
Name	Description	Format	Val / Opt	Value/Option Description	Def Val	Python Type	Usr	Opt	Mbr
project	The job's project. A project is a way to tag jobs. Each job can belong to at most one project.	<i>String</i> . Can contain any characters except for the following: Slash ("/"), left bracket ("["), right bracket ("]"), double quote ("\""), semicolon (";"), colon (":"), vertical bar (" "), left angle bracket ("<"), right angle bracket (">"), plus ("+"), comma (","), question mark ("?"), and asterisk ("*").			<i>_pbs_ project _default</i>	str	r, w	r, w	r, w
pset	Shows name of placement set used by the job. On BlueGene, specifies which partition should be used.	<i>String</i>				str	r	r	r, w

Job Attributes									
Name	Description	Format	Val / Opt	Value/Option Description	Def Val	Python Type	U sr	O pt	M gr
Rerunnable	Specifies whether the job can be rerun. Does not affect how a job is treated if the job could not begin execution. See section 3.13.8, "Marking a Job as "Rerunnable" or Not", on page 69 of the PBS Professional User's Guide . Job arrays are required to be rerunnable and are rerunnable by default.	<i>Character.</i> "y" or "n"	<i>y</i> <i>n</i>	The job can be rerun. Once the job starts running, it can never be rerun.	y	bool	r, w	r, w	r, w
Resource_List	The list of resources required by the job. List is a set of name=value strings. The meaning of name and value is dependent upon defined resources. Each value establishes the limit of usage of that resource. If not set, the value for a resource may be determined by a queue or server default established by the administrator. See Chapter 5, "Resources", on page 297 .	<i>String. Form:</i> <i>Resource_List.</i> <i><res>=<value></i> <i>,</i> <i>Resource_List.</i> <i><res>=<value></i> <i>,</i> ...			None	Dictionary: Resource_List[<resource name>]=<value> where <resource name> is any built-in or custom resource	r, w	r, w	r, w

Job Attributes									
Name	Description	Format	Val / Opt	Value/Option Description	Def Val	Python Type	Usr	Opt	Mgr
sandbox	Specifies type of location PBS uses for job staging and execution. User-settable via <code>qsub -wsandbox=<value></code> or via a PBS directive. See the <code>\$jobdir_root</code> MOM configuration option in <code>pbs_mom.8B</code> .	<i>String</i>	<i>PRIVATE</i> <i>HOME</i> or unset	PBS creates job-specific staging and execution directories under the directory specified in the <code>\$jobdir_root</code> MOM configuration option. PBS will use the job owner's home directory for staging and execution.	Unset	str	r, w	r, w	r, w
sched_hint	No longer used.						-	-	-
Shell_Path_List	One or more absolute paths to the program(s) to process the job's script file.	<i>String</i> . Form: <code>"path[@host][, path[@host]...]"</code> Must be enclosed in double quotes if it contains commas.			User's login shell on execution host	<code>pbs.path_list</code>	r, w	r, w	r, w
stagein	The list of files to be staged in prior to job execution.	<i>String</i> . Form: <code>"execution_path@storage_host:storage_path"</code> Must be enclosed in double quotes if it contains commas.			None	<code>pbs.staging_list</code>	r, w	r, w	r, w

Job Attributes									
Name	Description	Format	Val / Opt	Value/Option Description	Def Val	Python Type	Usr	Opt	Mgr
stageout	The list of files to be staged out after job execution.	<i>String</i> . Form: “ <i>execution_path@storage_host:storage_path</i> ” Must be enclosed in double quotes if it contains commas.			None	pbs.staging_list	r, w	r, w	r, w
Submit_arguments	Job submission arguments given on the qsub command line.	<i>String</i>			None	str	r, w	r, w	r, w
umask	The initial umask of the job is set to the value of this attribute when the job is created. This may be changed by umask commands in the shell initialization files such as .profile or .cshrc.	<i>Decimal integer</i>			077	int	r, w	r, w	r, w

Job Attributes									
Name	Description	Format	Val / Opt	Value/Option Description	Def Val	Python Type	Usr	Opt	Mgt
User_List	The list of users which determines the user name under which the job is run on a given host. When a job is to be executed, the server selects a user name from the list according to the following ordered set of rules: 1. Select the user name from the list for which the associated host name matches the name of the server. 2. Select the user name which has no associated host name; the wild card name. 3. Use the Job_Owner as the user name.	<i>String</i> . Form: “ <i>user@host</i> [<i>user@host...</i>]” Must be enclosed in double quotes if it contains commas. May be up to 15 characters in length.			Value of Job_Owner job attribute	pbs.user_list	r, w	r, w	r, w
Variable_List	List of environment variables set in the job’s execution environment. See the qsub (1B) command.	<i>String</i> . Form: “ <i>name=value</i> [<i>name=value...</i>]” Must be enclosed in double quotes if it contains commas.			None	Dictionary: Variable_List[<variable name>]=<value> where <resource name> is any built-in or custom resource	r, w	r, w	r, w

Job Attributes									
Name	Description	Format	Val / Opt	Value/Option Description	Def Val	Python Type	Usr	Opt	Mgt
The following job attributes are read-only:									
accounting_id	Accounting ID for tracking accounting data not produced by PBS.	String			None	str	r	r	r
accrue_type	Indicates what kind of time the job is accruing.	Integer	0 (initial_time)	Job is accruing initial time.	2 (eligible_time)	int	-	-	r
			1 (ineligible_time)	Job is accruing ineligible time.					
			2 (eligible_time)	Job is accruing eligible time.					
			3 (run_time)	Job is accruing run time.					

Job Attributes									
Name	Description	Format	Val / Opt	Value/Option Description	Def Val	Python Type	U	s	M
alt_id	For a few systems, the session id is insufficient to track which processes belong to the job. Where a different identifier is required, it is recorded in this attribute. If set, it will also be recorded in the end-of-job accounting record. For jobs running in CPU sets, the alt_id holds the set name in a form usable by the <code>cpuset (1)</code> command. For HPCBP, holds HPCBP server job ID. On Windows, holds PBS home directory.	<i>String</i> . May contain white spaces.			None	str	r	r	r
array_id	Applies only to subjobs. Array identifier of subjob.	<i>String</i>			None	str	r	r	r
array_index	Applies only to subjobs. Index number of subjob.	<i>String</i>			None	int	r	r	r
array_indices_remaining	Applies only to job arrays. List of indices of subjobs still queued.	<i>String</i> . Range or list of ranges, e.g. 500, 552, 596-1000.			None	str	r	r	r

Job Attributes									
Name	Description	Format	Val / Opt	Value/Option Description	Def Val	Python Type	U	s	M
array_state_count	Applies only to job arrays. Lists number of subjobs in each state.	<i>String</i>			None	pbs.state_count	r	r	r
ctime	The time that the job was created.	<i>Integer</i> . Seconds since epoch.			None	int	r	r	r
egroup	If the job is queued, this attribute is set to the group name under which the job is to be run.	<i>String</i>			None	str	-	-	r
estimated	List of values associated with job's estimated start time. Used to report job's <code>exec_vnode</code> and <code>start_time</code> . Can be set in a hook or via <code>qalter</code> , but PBS will overwrite the values.	<i>String</i> . Form: <i>estimated.<res>=<value></i> , <i>estimated.<res>=<value></i>	<i>exec_vnode</i>	The estimated vnodes used by this job.	Unset	Dictionary: estimated. [<code><resource name>]=<value></code> where <code><resource name></code> is any resource.	r	r, w	r, w
			<i>start_time</i>	The estimated start time for this job.	Unset		r	r, w	r, w
etime	The time that the job became eligible to run when queued in an execution queue.	<i>Integer</i> . Seconds since epoch.			None	int	r	r	r
euser	If the job is queued, this attribute is set to the user name under which the job is to be run.	<i>String</i>			None	str	-	-	r

Job Attributes									
Name	Description	Format	Val / Opt	Value/Option Description	Def Val	Python Type	U	sr	M
Exit_status	Exit status of job. Set to zero for successful execution. If any subjob of an array job has non-zero exit status, the array job has non-zero exit status.	<i>Integer</i>			None	int	r	r	r
forward_x11_cookie	Contains the X authorization cookie.	<i>String</i>			None	str	r	r	r
forward_x11_port	Contains the number of the port being listened to by the port forwarder on the submission host.	<i>Integer</i>			None	int	r	r	r
hashname	No longer used.						-	-	-
jobdir	Path of the job's staging and execution directory on the primary execution host. Either user's home, or private sandbox. Depends on value of <code>sandbox</code> attribute. Viewable via <code>qstat -f</code> .	<i>String</i>				str	r	r	r
Job_Owner	The login name on the submitting host of the user who submitted the batch job.	<i>String. <Username>@<submission host></i>				str	r	r	r

Job Attributes									
Name	Description	Format	Val / Opt	Value/Option Description	Def Val	Python Type	Usr	Opt	Mgr
job_state	The state of the job.	Character	<i>E</i> (Exiting)	The job has finished, with or without errors, and PBS is cleaning up post-execution.	None	PBS job state constant	r, i	r, i	r, i
			<i>H</i> (Held)	The job is held.					
			<i>Q</i> (Queued)	The job resides in an execution or routing queue pending execution or routing. It is not in held or waiting state.					
			<i>R</i> (Running)	The job is in an execution queue and is running.					
			<i>S</i> (Suspend)	The job was executing and has been suspended. The job retains its assigned resources but does not use CPU cycles or walltime.					
			<i>T</i> (Transiting)	The job is being routed or moved to a new destination.					
			<i>U</i> (User suspend)	The job was running on a workstation configured for cycle harvesting and the keyboard/mouse is currently busy. The job is suspended until the workstation has been idle for a configured amount of time.					
			<i>W</i> (Waiting)	The <code>Execution_Time</code> attribute contains a time in the future. Can be set when stage-in fails and PBS moves job start time out 30 minutes to allow user to fix problem.					

Job Attributes									
Name	Description	Format	Val / Opt	Value/Option Description	Def Val	Python Type	Usr	Opt	Mgr
mtime	The time that the job was last modified, changed state, or changed locations.	<i>Integer</i> . Seconds since epoch.			None	int	r	r	r
qtime	The time that the job entered the current queue.	<i>Integer</i> . Seconds since epoch.			None	int	r	r	r
Queue	The name of the queue in which the job currently resides.	<i>String</i>			None	pbs.queue	r	r	r
queue_rank	A number indicating the job's position within its queue. Only used internally by PBS.	<i>Integer</i>			None	int	-	-	r
queue_type	The type of queue in which the job is currently residing.	<i>Character</i>	<i>E</i>	Execution queue	None	PBS queue type constant.	-	-	r
			<i>R</i>	Routing queue					
resources_used	The amount of each resource used by the job.	<i>String</i> . Form: List of name=value pairs; format is <i>resources_used.<res>=<val></i> , <i>resources_used.<res>=<val></i>			None	Dictionary: resources_used [<resource name>]= <value> where <resource name> is any built-in or custom resource	r	r	r

Job Attributes									
Name	Description	Format	Val / Opt	Value/Option Description	Def Val	Python Type	U	s	M
run_count	The number of times the job has been executed.	<i>Integer</i>			Zero	int	-	-	r
schedselect	The union of the select specification of the job, and the queue and server defaults for resources in a chunk.				None	pbs.select	-	-	r
server	The name of the server which is currently managing the job. When the secondary server is running during failover, shows the name of the primary server. After a job is moved to another server, either via qmove or peer scheduling, shows the name of the new server.	<i>String</i>			None	pbs.server	r	r	r
session_id	If the job is running, this is set to the session id of the first executing task.				None	int	r	r	r

Job Attributes									
Name	Description	Format	Val / Opt	Value/Option Description	Def Val	Python Type	U	s	M
Stageout_status	Status of stageout. If stageout succeeded, this is set to <i>1</i> . If stageout failed, this is set to <i>0</i> . Available only for finished jobs. Displayed only if set. If stageout fails for any subjob of an array job, the value of Stageout_status is zero for the array job.	<i>Integer</i>			None	int	r	r	r
stime	The time when the job started execution. Changes when job is restarted.	<i>Integer. Seconds since epoch.</i>			None	int	r	r	r
substate	The substate of the job. The substate is used internally by PBS.	<i>Integer</i>			None	int	r	r	r
sw_index	No longer used. Gone from .c file.						-	-	-

6.12 Hook Attributes

An unset hook attribute takes the default value for that attribute.

Hook attributes can be set by root only.

Hook Attributes									
Name	Description	Format	Val / Opt	Value/Option Description	Default Value	Python Type	U	s	M
The following attributes are settable and readable by root or the Admin at the local server only.									
Type	The type of the hook.	<i>String</i>	<i>site</i>		<i>site</i>				
enabled	Determines whether or not a hook is run when its triggering event occurs.	<i>Boolean</i>	<i>True</i>	Hook runs when triggering event occurs	<i>True</i>				
			<i>False</i>	Hook is not run when triggering event occurs					
freq	Frequency with which <code>exechost_periodic</code> hook runs	<i>Integer</i>		Number of seconds between triggers	<i>120</i>				
user	Specifies who executes the hook.	<i>String</i>	<i>pbsadmin</i>	Hook runs as root	<i>pbsadmin</i>				
			<i>pbsuser</i>	Hook runs as owner of job					

Hook Attributes									
Name	Description	Format	Val / Opt	Value/Option Description	Default Value	Python Type	Use	Opt	Mod
event	List of events that trigger the hook. Can be operated on with the "=", "+=", and "-=" operators. The <i>provision</i> event cannot be combined with any other events.	<i>string_array</i>	<i>queuejob</i>	Triggered when job is queued	"" meaning hook is not triggered				
			<i>modifyjob</i>	Triggered when job is modified					
			<i>movejob</i>	Triggered when job is moved					
			<i>resvsub</i>	Triggered when reservation is created					
			<i>runjob</i>	Triggered when job is run					
			<i>provision</i>	Hook is master provisioning hook					
			<i>execjob_begin</i>	Triggered when MoM receives job					
			<i>execjob_end</i>	Triggered after job finishes or is killed					
			<i>execjob_preterm</i>	Triggered just before job is killed					
			<i>execjob_prologue</i>	Triggered just before first job process					
			<i>execjob_epilogue</i>	Triggered just after job runs successfully					
			<i>exechost_periodic</i>	Triggered at periodic interval on execution hosts					
			""	Hook is not triggered					
order	Indicates relative order of hook execution, for hooks sharing a trigger. Hooks with lower order values execute before those with higher values.	<i>Integer</i>			1				

Hook Attributes									
Name	Description	Format	Val / Opt	Value/Option Description	Default Value	Python Type	Usr	Opt	Mgr
alarm	Specifies the number of seconds to allow a hook to run before the hook times out.	<i>Integer. Must be greater than zero.</i>			30				

Chapter 7

Formats

This chapter describes the formats used in PBS Professional.

7.1 List of Formats

PBS NAME

This is a generic term, used to describe various PBS entities. For example, attribute names are PBS NAMES.

Must start with an alpha-numeric character, and may contain the following:

alpha-numeric , . _ - # : / \ @

Specifically excluded: ` ~ ! \$ % ^ & * () + = < > ? ; ' " |

Accounting Log Entry

logfile-date-time;record-type;id-string;message-text

where

logfile-date-time

Date and time stamp in the format:

mm/dd/yyyy hh:mm:ss

record-type

A single character indicating the type of record

id-string

The job or reservation identifier

message-text

Format: blank-separated *keyword=value* fields.

Message text is ASCII text.

Content depends on the record type.

Attribute Name

PBS NAME

Boolean

Name of resource is a string.

Values:

TRUE, True, true, T, t, Y, y, 1

FALSE, False, false, F, f, N, n, 0

Date

*<Day of week> <Name of month> <Day of month> <HH:MM:SS>
<YYYY>*

Datetime

[[[CC]YY]MM]DD]hhmm[.SS]

where

Table 7-1: Datetime Symbols

Symbol	Meaning
<i>CC</i>	Century
<i>YY</i>	Year
<i>MM</i>	Month
<i>DD</i>	Day of month
<i>hh</i>	Hour
<i>mm</i>	Minute
<i>SS</i>	Second

When setting the value, each portion of the date defaults to the current date, as long as the next-smaller portion is in the future. For example, if today is the 3rd of the month and the specified day *DD* is the 5th, the month *MM* will be set to the current month.

If a specified portion has already passed, the next-larger portion will be set to one after the current date. For example, if the day **DD** is not specified, but the hour **hh** is specified to be 10:00 a.m. and the current time is 11:00 a.m., the day **DD** will be set to tomorrow.

Unspecified portions default to *now*.

Destination Identifier

String used to specify a particular destination. The identifier may be specified in one of three forms:

<queue name>@<server name>

<queue name>

@<server name>

where *<queue name>* is an ASCII character string of up to 15 characters.

Valid characters are alphanumerics, the hyphen and the underscore. The string must begin with a letter.

Duration

A period of time, expressed either as

integer seconds

or

[[hours:]minutes:]seconds[.milliseconds]

in the form:

[[HH:]MM:]SS[.milliseconds]

Note that milliseconds are rounded to nearest second.

Float

Floating point. Allowable values: *[+-] 0-9 [[0-9] ...][.][[0-9] ...]*

Host Name

String of the form

name.domain

where *domain* is a hierarchical, dot-separated list of subdomains. Therefore, a host name cannot contain a dot, "." as a legal character other than as a sub-domain separator.

The name must not contain the commercial at sign, "@", as this is often used to separate a file from the host in a remote file name.

A host name cannot contain a colon, ":".

The maximum length of a host name supported by PBS is defined by **PBS_MAXHOSTNAME**, currently set to **64**.

Job Identifier

sequence_number[.server_name][@server]

Format:

[0-9]+[/\[[0-9]+\]]\.IP_HOSTNAME[.IP_DOMAINNAME]

Job Array Identifier

Job array identifiers are a sequence number followed by square brackets:

sequence_number[][.server_name][@server]

Example:

1234[]

Note that some shells require that you enclose a job array ID in double quotes.

Job Array Range

sequence_number[<first>-<last>][.server_name][@server]

first and *last* are the first and last indices of the subjobs.

Job Name, Job Array Name

A job name or job array name can be at most 15 characters. It must consist of printable, non-whitespace characters. The first character must be alphabetic, numeric, plus sign (“+”), dash or minus (“-”), or underscore (“_”). The name must not contain special characters.

Limit Specification

limit-spec=limit[, limit-spec=limit, ...]

where *limit-spec* is:

Table 7-2: Limit Specification Syntax

<i>limit-spec</i>
<i>o:PBS_ALL</i>
<i>p:PBS_GENERIC</i>
<i>p:<project name></i>
<i>u:PBS_GENERIC</i>

Table 7-2: Limit Specification Syntax

<i>limit-spec</i>
<i>u:<user name></i>
<i>g:PBS_GENERIC</i>
<i>g:<group name></i>

See the PBS Professional Administrator's Guide for an explanation of what each limit-spec means.

logfile-date-time

Date and time stamp in the format:

mm/dd/yyyy hh:mm:ss

long

Long integer. Allowable values: 0-9 [[0-9] ...]

Queue Identifier

To specify a queue at the default server:

<queue name>

To specify all queues at a server:

@<server name>

To specify a queue at a specific server:

<queue name>@<server name>

Queue Name

PBS NAME

Resource Name

PBS NAME

Resource names are case-insensitive.

Resource Value

- PBS NAME, or
- Anything inside double quotes

The format of each data type is defined for that data type. For example, float resources are defined above, in ["Float" on page 405](#).

Size

Number of bytes or words. The size of a word is the word size on the execution host.

Default: bytes

Format:

integer[*suffix*]

where *suffix* can be one of the following:

Table 7-3: Size in Bytes

Suffix	Meaning	Size
b or w	Bytes or words	1
kb or kw	Kilobytes or kilowords	2 to the 10th, or 1024
mb or mw	Megabytes or megawords	2 to the 20th, or 1,048,576
gb or gw	Gigabytes or gigawords.	2 to the 30th, or 1,073,741,824
tb or tw	Terabytes or terawords.	2 to the 40th, or 1024 gigabytes
pb or pw	Petabytes or petawords.	2 to the 50th, or 1,048,576 gigabytes

String (resource value)

Any character, including the space character.

Only one of the two types of quote characters, " or ', may appear in any given value.

Values: [_a-zA-Z0-9][[_a-zA-Z0-9 ! " # \$ % ' () * + , - . / : ; < = > ? @ [\] ^ _ ' { | } ~] ...]

String resource values are case-sensitive.

string_array

Comma-separated list of strings. Strings in string_array may not contain commas.

Subjob Identifier

Subjob identifiers are a sequence number followed by square brackets enclosing the subjob's index:

sequence_number[index][.server_name][@server]

Example:

1234[99]

User Name

String up to 16 characters in length. PBS supports names containing any printable, non-whitespace character except the at sign (“@”). Your platform may place additional limitations on user names.

User Name, Windows

Must conform to the POSIX-1 standard for portability:

- The username must contain only alphanumeric characters, dot (.), underscore (_), and/or hyphen "-".
- The hyphen must not be the first letter of the username.
- If "@" appears in the username, then it will assumed to be in the context of a Windows domain account: username@domainname.
- An exception to the above rule is the space character, which is allowed. If a space character appears in a username string, then it will be displayed quoted and must be specified in a quoted manner.

Vnode Name

- For the natural vnode, the vnode name must conform to legal name for a host
- For other vnodes, the vnode name is a PBS NAME

Chapter 8

States

This chapter lists and describes the states in PBS Professional.

8.1 Job States

Job states are abbreviated to one character.

Table 8-1: Job States

State	Description
<i>B</i>	Job arrays only: job array has started
<i>E</i>	Job is exiting after having run
<i>F</i>	Job is finished. Job has completed execution, job failed during execution, or job was deleted.
<i>H</i>	Job is held. A job is put into a held state by the server or by a user or administrator. A job stays in a held state until it is released by a user or administrator.
<i>M</i>	Job was moved to another server
<i>Q</i>	Job is queued, eligible to run or be routed
<i>R</i>	Job is running
<i>S</i>	Job is suspended by server. A job is put into the suspended state when a higher priority job needs the resources.

Table 8-1: Job States

State	Description
<i>T</i>	Job is in transition (being moved to a new location)
<i>U</i>	Job is suspended due to workstation becoming busy
<i>W</i>	Job is waiting for its requested execution time to be reached or job specified a stagein request which failed for some reason.
<i>X</i>	Subjobs only; subjob is finished (expired.)

8.1.1 Job Substates

Job substates are numeric:

Table 8-2: Job Substates

Substate Number	Substate Description
00	Transit in, prior to waiting for commit
01	Transit in, waiting for commit
02	transiting job outbound, not ready to commit
03	transiting outbound, ready to commit
10	job queued and ready for selection
11	job queued, has files to stage in
14	job staging in files before waiting
15	job staging in files before running
16	job stage in complete
20	job held - user or operator
22	job held - waiting on dependency
30	job waiting until user-specified execution time

Table 8-2: Job Substates

Substate Number	Substate Description
37	job held - file stage in failed
41	job sent to MOM to run
42	Running
43	Suspended by Operator or Manager
44	PBS no longer supports Globus. The Globus functionality has been removed from PBS. Job sent to run under Globus
45	Suspended by Scheduler
50	Server received job obit
51	Staging out stdout/err and other files
52	Deleting stdout/err files and staged-in files
53	Mom releasing resources
54	job is being aborted by server
56	(Set by MOM) Mother Superior telling sisters to kill everything
57	(Set by MOM) job epilogue running
58	(Set by MOM) job obit notice sent
59	Waiting for site "job termination" action script
60	Job to be rerun, MOM sending stdout/stderr back to Server
61	Job to be rerun, staging out files
62	Job to be rerun, deleting files
63	Job to be rerun, freeing resources
70	Array job has begun

Table 8-2: Job Substates

Substate Number	Substate Description
71	Job is waiting for vnode(s) to be provisioned with its requested AOE.
153	(Set by MOM) Mother Superior waiting for delete ACK from sisters

8.2 Job Array States

Job array states map closely to job states except for the ‘*B*’ state. The ‘*B*’ state applies to job arrays and indicates that at least one subjob has left the queued state and is running or has run, but not all subjobs have run. Job arrays will never be in the ‘*R*’, ‘*S*’ or ‘*U*’ states.

Table 8-3: Job Array States

State	Indication
<i>B</i>	The job array has started
<i>E</i>	All subjobs are finished and the server is cleaning up the job array
<i>F</i>	The job array is finished
<i>H</i>	The job array is held
<i>Q</i>	The job array is queued, or has been qrerun
<i>T</i>	The job array is in transit between servers
<i>W</i>	The job array has a wait time in the future

8.3 Subjob States

Subjobs can be in one of six states, listed here.

Table 8-4: Subjob States

State	Indication
<i>E</i>	Ending
<i>F</i>	Finished
<i>Q</i>	Queued
<i>R</i>	Running
<i>S</i>	Suspended
<i>U</i>	Suspended by keyboard activity
<i>X</i>	Expired or deleted; subjob has completed execution or been deleted

8.4 Server States

The state of the server is shown in the `server_state` server attribute. Possible values are:

Table 8-5: Server States

<i>Hot_Start</i>	The Server has been started so that it will run first any jobs that were running when the Server was shut down. Python type: <code>pbs.SV_STATE_HOT</code>
<i>Idle</i>	The Server is running. The Scheduler is between scheduling cycles. Python type: <code>pbs.SV_STATE_IDLE</code>
<i>Scheduling</i>	The Server is running. The Scheduler is in a scheduling cycle. Python type: <code>pbs.SV_STATE_ACTIVE</code>

Table 8-5: Server States

<i>Terminating</i>	The Server is terminating. Python type: pbs.SV_STATE_SHUTIMM or pbs.SV_STATE_SHUTSIG
<i>Terminating_Delayed</i>	The Server is terminating in delayed mode. No new jobs will be run, and the Server will shut down when the last running job finishes. Python type: pbs.SV_STATE_SHUTDEL

8.5 Vnode States

If a vnode's **state** attribute is unset, that is equivalent to the state being *free*. A vnode's state is shown in its **state** attribute, which can take on zero or more of the values listed here. Some vnode state values can be set simultaneously. Values are:

Table 8-6: Vnode States

State	Set By	Description	Can Combine With these States
<i>busy</i>	Server	Node is up and has load average greater than max_load , or is showing keyboard or mouse activity. When the loadave is above max_load , that node is marked <i>busy</i> . The scheduler won't place jobs on a node marked <i>busy</i> . When the loadave drops below ideal_load , or when the mouse and keyboard have not shown any activity for a specified amount of time, the <i>busy</i> mark is removed. Consult your OS documentation to determine values that make sense.	<i>offline</i>

Table 8-6: Vnode States

State	Set By	Description	Can Combine With these States
<i>down</i>	Server	Node is not usable. Existing communication lost between Server and MOM.	Cannot be set with <i>free</i>
<i>free</i>	Server	Node is up and has available CPU(s). Server will mark a vnode “ <i>free</i> ” on first successful ping after vnode was “ <i>down</i> ”.	None
<i>job-busy</i>	Server	Node is up and all CPUs are allocated to jobs.	<i>offline</i> <i>resv-exclusive</i>
<i>job-exclusive</i>	Server	Node is up and has been allocated exclusively to a single job.	<i>offline</i> <i>resv-exclusive</i>
<i>offline</i>	Manager Operator	Node is not usable. Jobs running on this vnode will continue to run. Used by Manager/Operator to mark a vnode not to be used for jobs.	<i>busy</i> <i>job-busy</i> <i>job-exclusive</i> <i>resv-exclusive</i>
<i>provisioning</i>	Server	A vnode is in the provisioning state while it is in the process of being provisioned. No jobs are run on vnodes in the provisioning state.	Cannot be set with any other states
<i>resv-exclusive</i>	Server	Reservation has requested exclusive use of vnode, and reservation is running.	<i>job-exclusive</i> , <i>offline</i>

Table 8-6: Vnode States

State	Set By	Description	Can Combine With these States
<i>stale</i>	Server	<p>MOM managing vnode is not reporting any information about this vnode, but was reporting it previously. Server can still communicate with MOM.</p> <p>A vnode becomes <i>stale</i> when:</p> <ol style="list-style-type: none"> 1. A vnode is defined in the Server 2. MOM starts or restarts and reports a set of vnodes according to her configuration 3. A vnode which existed in the server earlier is not in the set being reported now by MOM 4. That vnode is marked "<i>stale</i>" 	Cannot be set with <i>free</i>
<i>state-unknown, down</i>	Server	Node is not usable. Since Server's latest start, no communication with this vnode. May be network or hardware problem, or no MOM on vnode.	
<i>unresolvable</i>	Server	Server cannot resolve name of vnode	

Table 8-6: Vnode States

State	Set By	Description	Can Combine With these States
<i>wait-provisioning</i>	Server	There is a limit on the maximum number of vnodes that can be in the provisioning state. This limit is specified in the Server's <code>max_concurrent_provision</code> attribute. If a vnode is to be provisioned, but cannot because the number of concurrently provisioning vnodes has reached the specified maximum, the vnode goes into the <i>wait-provisioning state</i> . No jobs are run on vnodes in the <i>wait-provisioning state</i> .	Cannot be set with any other states

8.6 Reservation States

The following table shows the list of possible states for a reservation. The states that you will usually see are *CO*, *UN*, *BD*, and *RN*, although a reservation usually remains unconfirmed for too short a time to see that state.

Table 8-7: Reservation States

Code	Numeric	State	Description
<i>BD</i>	7	<i>RESV_BEING_DELETED</i>	Transitory state; reservation is being deleted
<i>CO</i>	2	<i>RESV_CONFIRMED</i>	Reservation confirmed
<i>DG</i>	10	<i>RESV_DEGRADED</i>	Vnode(s) allocated to reservation unavailable

Table 8-7: Reservation States

Code	Nu meric	State	Description
<i>DE</i>	8	<i>RESV_DELETED</i>	Transitory state; reservation has been deleted
<i>DJ</i>	9	<i>RESV_DELETING_JOBS</i>	Jobs remaining after reservation's end time being deleted
<i>FN</i>	6	<i>RESV_FINISHED</i>	Transitory state; reservation's end time has arrived and reservation will be deleted
<i>NO</i>	0	<i>RESV_NONE</i>	No reservation yet
<i>RN</i>	5	<i>RESV_RUNNING</i>	Time period from reservation's start time to end time is being traversed
<i>TR</i>	4	<i>RESV_TIME_TO_RUN</i>	Transitory state; reservation's start time has arrived
<i>UN</i>	1	<i>RESV_UNCONFIRMED</i>	Reservation not confirmed
<i>WT</i>	3	<i>RESV_WAIT</i>	Unused

8.6.1 Degraded Reservation Substates

The following table shows states and substates for degraded reservations:

Table 8-8: Degraded Reservation States and Substates

		Reservation Time in Future	Reservation Time Is Now
Advance Reser- vation	State	<i>RESV_DEGRADED</i>	<i>RESV_RUNNING</i>
	Sub- state	<i>RESV_DEGRADED</i>	<i>RESV_DEGRADED</i>

Table 8-8: Degraded Reservation States and Substates

		Reservation Time in Future	Reservation Time Is Now
Soonest Occur- rence	State	<i>RESV_DEGRADED</i>	<i>RESV_RUNNING</i>
	Sub- state	<i>RESV_DEGRADED</i>	<i>RESV_DEGRADED</i>
Non-soonest Occurrence Only	State	<i>RESV_CONFIRMED</i>	N/A
	Sub- state	<i>RESV_DEGRADED</i>	N/A

Chapter 9

Accounting Log

This chapter describes the accounting log. The PBS accounting log is a text file with each entry terminated by a newline. There is no limit to the size of an entry.

9.1 Log Entry Format

The format of an entry is:

logfile-date-time;record-type;id-string;message-text

where

logfile-date-time

Date and time stamp in the format:

mm/dd/yyyy hh:mm:ss

record-type

A single character indicating the type of record

id-string

The job or reservation identifier

message-text

Message text format is blank-separated *keyword=value* fields.

Message text is ASCII text.

Content depends on the record type.

There is no dependable ordering of the content of each message.

9.1.1 Resources

Values for requested resources are written in the same units as those in the resource requests. Values for `resources_used` and `resources_assigned` are written in kb. A suffix is always written unless the quantity is measured in bytes.

For `Resource_List` and `resources_used`, there is one entry per resource, corresponding to the resources requested and used, respectively.

If a job ends between MOM poll cycles, `resources_used.RES` numbers will be slightly lower than they are in reality. For long-running jobs, the error percentage will be minor.

9.2 Record Types

The record types are:

A

Job was aborted by the server.

B

Beginning of reservation period. If the log entry is for a reservation, the *message-text* field contains information describing the specified reservation. Possible information includes:

Table 9-1: Reservation Information

Attribute	Explanation
<code>owner=ownername</code>	Name of party who submitted the reservation request.
<code>name=reservation_name</code>	If submitter supplied a name string for the reservation.
<code>queue=queue_name</code>	The name of the reservation queue.
<code>ctime=creation_time</code>	Time at which the reservation was created; seconds since the epoch.
<code>start=period_start</code>	Time at which the reservation period is to start, in seconds since the epoch.
<code>end=period_end</code>	Time at which the reservation period is to end, in seconds since the epoch.

Table 9-1: Reservation Information

Attribute	Explanation
<code>duration = reservation_duration</code>	The duration specified or computed for the reservation, in seconds.
<code>exec_host= vnode_list</code>	List of each vnode with vnode-level, consumable resources allocated from that vnode. <code>exec_host=vnodeA/P*C [+vnodeB/P * C]</code> where P is a unique index and C is the number of CPUs assigned to the reservation, 1 if omitted.
<code>Authorized_Users = user_list</code>	The list of users who are and are not authorized to submit jobs to the reservation.
<code>Authorized_Groups= group_list</code>	The list of groups who are and are not authorized to submit jobs to the reservation.
<code>Authorized_Hosts= host_list</code>	The list of hosts from which jobs may and may not be submitted to the reservation.
<code>Resource_List= resources_list</code>	List of resources requested by the reservation. Resources are listed individually as, for example: <code>Resource_List.ncpus = 16</code> <code>Resource_List.mem = 1048676.</code>

C

Job was checkpointed and held.

D

Job was deleted by request. The *message-text* will contain `requestor=user@host` to identify who deleted the job.

E

Job ended (terminated execution). In this case, the *message-text* field contains information about the job. The end of job accounting record will not be written until all of the resources have been freed. The “end” entry in the

job end record will include the time to stage out files, delete files, and free the resources. This will not change the recorded `walltime` for the job. Possible information includes:

Table 9-2: Job End Record Contents

Attribute	Explanation
<code>Exit_status=</code> <code><value></code>	The exit status of the job. See section 12.10, "Job Exit Codes" on page 829 in the PBS Professional Administrator's Guide .
<code>account=</code> <code><account_string></code>	If job has an "account name" string.
<code>accounting_id=</code> <code><jidvalue></code>	CSA JID, job container ID
<code>alt_id=</code> <code><ID></code>	Optional alternate job identifier. Included only for certain systems: On Altix machines with supported versions of ProPack or Performance Suite, the alternate ID will show the path to the job's cpuset, starting with <code>/PBSPRO/</code> .
<code>ctime=</code> <code><time></code>	Time in seconds when job was created (first submitted), seconds since epoch.
<code>eligible_time=</code> <code><time></code>	Amount of time job has waited while blocked on resources.
<code>end=</code> <code><time></code>	Time in seconds since epoch when this accounting record was written.
<code>etime=</code> <code><time></code>	Time in seconds when job became eligible to run, i.e. was enqueued in an execution queue and was in the "Q" state. Reset when a job moves queues. Not affected by galtering.

Table 9-2: Job End Record Contents

Attribute	Explanation
<code>exec_host= <vnode_list></code>	List of each vnode with vnode-level, consumable resources allocated from that vnode. <code>exec_host=vnodeA/P*C</code> [+vnodeB/P * C] where P is a unique index and C is the number of CPUs assigned to the job, 1 if omitted.
<code>exec_vnode= <vnode_list></code>	List of each vnode with vnode-level, consumable resources from that vnode. (vnodeA+resource_name=P+..)+(vnodeB+resource_name=P+.. P is the amount of that resource allocated from that vnode.
<code>group=<groupname></code>	The group name under which the job executed.
<code>jobname=<job_name></code>	The name of the job.
<code>project=<project name></code>	The job's project name at the end of the job.
<code>qtime=<time></code>	Time in seconds when job was queued into current queue.
<code>queue=<queue_name></code>	The name of the queue in which the job executed.
<code>resources_used.<resource>=<amount></code>	Resources used by the job as reported by MOM. Typically includes <code>ncpus</code> , <code>mem</code> , <code>vmem</code> , <code>cput</code> , <code>walltime</code> , <code>cpupercent</code> . Walltime does not include suspended time.

Table 9-2: Job End Record Contents

Attribute	Explanation
resource_assigned.<resource>= = <value>	<p>Not a job attribute; simply a label for reporting job resource assignment.</p> <p>The value of <code>resources_assigned</code> reported in the Accounting records is the actual amount assigned to the job by PBS. All allocated consumable resources will be included in the "resource_assigned" entries, one resource per entry. Consumable resources include <code>ncpus</code>, <code>mem</code> and <code>vmem</code> by default, and any custom resource defined with the <code>-n</code> or <code>-f</code> flags. A resource will not be listed if the job does not request it directly or inherit it by default from queue or server settings. For example, if a job requests one CPU on an Altix that has four CPUs per blade/vnode and that vnode is allocated exclusively to the job, even though the job requested one CPU, it is assigned all 4 CPUs.</p>
Resource_List.<resource>= <amount>	<p>List of resources requested by the job. Resources are listed individually as, for example:</p> <pre>Resource_List.ncpus =16 Resource_List.mem =1048676.</pre>
run_count=<value>	The number of times the job has been executed.

Table 9-2: Job End Record Contents

Attribute	Explanation
<code>session=<sessionID></code>	Session number of job.
<code>start=<time></code>	Time when job execution started.
<code>user=<username></code>	The user name under which the job executed.

F

Resource reservation period finished.

K

Scheduler or server requested removal of the reservation. The *message-text* field contains: `requestor=Server@host` or `requestor=Sched-
uler@host` to identify who deleted the resource reservation.

k

Resource reservation terminated by ordinary client - e.g. an owner issuing a `pbs_rdel` command. The *message-text* field contains: `requestor=user@host` to identify who deleted the resource reservation.

L

License information. This line in the log will have the following fields:

*Log date; record type; keyword; specification for floating license; hour;
day; month; max*

The following table explains each field:

Table 9-3: Licensing Information in Accounting Log

Field	Explanation
Log date	Date of event
record type	Indicates license info
keyword	license
specification for floating license	Indicates that this is floating license info
hour	Number of licenses used in the last hour

Table 9-3: Licensing Information in Accounting Log

Field	Explanation
day	Number of licenses used in the last day
month	Number of licenses used in the last month
max	Maximum number of licenses ever used. Not dependent on server restarts.

M

Information about moved jobs. Contains date, time, record type, job ID, destination.

Q

Job entered a queue. For this kind of record type, the *message-text* contains `queue=name` identifying the queue into which the job was placed. There will be a new Q record each time the job is routed or moved to a new (or the same) queue.

R

Job was rerun.

In this case, the *message-text* field contains information about the job. Possible information includes:

Table 9-4: Job Rerun Record Contents

Attribute	Explanation
Exit_status=<value>	The exit status of the previous start of the job. See section 12.10, "Job Exit Codes" on page 829 in the PBS Professional Administrator's Guide .
account= <account_string>	If job has an "account name" string.
accounting_id=<jidvalue>	CSA JID, job container ID

Table 9-4: Job Rerun Record Contents

Attribute	Explanation
<code>alt_id=<ID></code>	Optional alternate job identifier. Included only for certain systems: On Altix machines with supported versions of ProPack or Performance Suite, the alternate ID will show the path to the job's cpuset, starting with <code>/PBSPro/</code> .
<code>ctime=<time></code>	Time in seconds when job was created (first submitted), seconds since epoch.
<code>eligible_time=<time></code>	Amount of time job has waited while blocked on resources, starting at creation time.
<code>end=<time></code>	Time in seconds since epoch when this accounting record was written.
<code>etime=<time></code>	Time in seconds when job most recently became eligible to run, i.e. was enqueued in an execution queue and was in the "Q" state. Reset when a job moves queues. Not affected by qaltering.
<code>exec_host= <vnode_list></code>	List of each vnode with vnode-level, consumable resources allocated from that vnode. <code>exec_host=vnodeA/P*C [+vnodeB/P * C]</code> where P is a unique index and C is the number of CPUs assigned to the job, 1 if omitted.

Table 9-4: Job Rerun Record Contents

Attribute	Explanation
<code>exec_vnode= <vnode_list></code>	List of each vnode with vnode-level, consumable resources from that vnode. (vnodeA+resource_name=P+..)+(vnodeB+resource_name=P+..) P is the amount of that resource allocated from that vnode.
<code>group=<groupname></code>	The group name under which the job executed.
<code>jobname=<job_name></code>	The name of the job.
<code>project=<project name></code>	The job's project name.
<code>qtime=<time></code>	Time in seconds when job was queued into current queue.
<code>queue=<queue_name></code>	The name of the queue in which the job is enqueued.
<code>Resource_List.<resource>=<amount></code>	List of resources requested by the job. Resources are listed individually as, for example: <code>Resource_List.ncpus =16</code> <code>Resource_List.mem =1048676.</code>
<code>run_count=<value></code>	The number of times the job has been executed.
<code>session=<sessionID></code>	Session number of job.
<code>start=<time></code>	Time when job execution started most recently.
<code>user=<username></code>	The user name under which the job executed.

S

Job execution started. The *message-text* field contains:

Table 9-5: Job Start Record Contents

Attribute	Explanation
<code>accounting_id=identifier_string</code>	An identifier that is associated with system-generated accounting data. In the case where accounting is CSA, <i>identifier_string</i> is a job container identifier or JID created for the PBS job.
<code>ctime=time</code>	Time in seconds when job was created (first submitted).
<code>etime=time</code>	Time in seconds when job became eligible to run; no holds, etc.
<code>exec_host=vnode_list</code>	List of each vnode with vnode-level, consumable resources allocated from that vnode. <code>exec_host=vnodeA/P*C [+vnodeB/P * C]</code> where P is the job number and C is the number of CPUs assigned to the job, 1 if omitted.
<code>group=groupname</code>	The group name under which the job will execute.
<code>jobname=job_name</code>	The name of the job.
<code>project=<project name></code>	The job's project name at the start of the job.
<code>qtime=time</code>	Time in seconds when job was queued into current queue.
<code>queue=queue_name</code>	The name of the queue in which the job resides.
<code>resource_assigned</code>	Not a job attribute; instead simply a label for reporting resources assigned to a job. Consumable resources that were allocated to that job.

Table 9-5: Job Start Record Contents

Attribute	Explanation
<code>Resource_List.resource=amount</code>	List of resources requested by the job. Resources are listed individually as, for example: <code>Resource_List.ncpus=16 Resource_List.mem=1048676</code> .
<code>session=sessionID</code>	Session number of job.
<code>start=time</code>	Time in seconds when job execution started.
<code>user=username</code>	The user name under which the job will execute.

T

Job was restarted from a checkpoint file.

U

Created unconfirmed reservation on Server. The *message-text* field contains `requestor=user@host` to identify who requested the reservation.

Y

Reservation confirmed by the Scheduler. The *message-text* field contains `requestor=user@host` to identify who requested the reservation.

Chapter 10

Example Configurations

This chapter shows some configuration-specific scenarios which will hopefully clarify any configuration questions. Several configuration models are discussed, followed by several complex examples of specific features.

- Single Vnode System

- Single Vnode System with Separate PBS Server

- Multi-vnode complex

- Complex Multi-level Route Queues (including group ACLs)

- Multiple User ACLs

For each of these possible configuration models, the following information is provided:

- General description for the configuration model

- Type of system for which the model is well suited

- Contents of Server nodes file

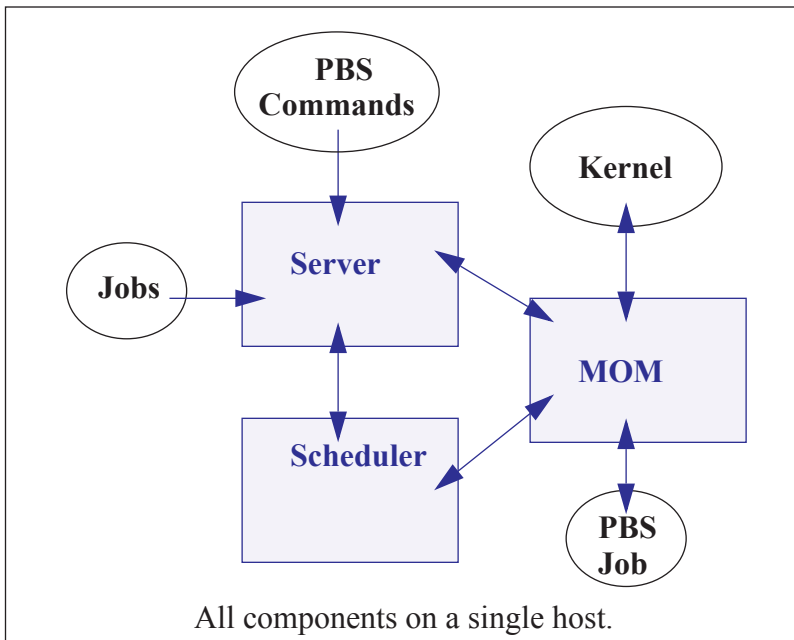
- Any required Server configuration

- Any required MOM configuration

- Any required Scheduler configuration

10.1 Single Vnode System

Running PBS on a single vnode/host as a standalone system is the least complex configuration. This model is most applicable to sites who have a single large Server system, a single SMP system (e.g. an SGI Origin server), or even a vector supercomputer. In this model, all three PBS components run on the same host, which is the same host on which jobs will be executed, as shown in the figure below.



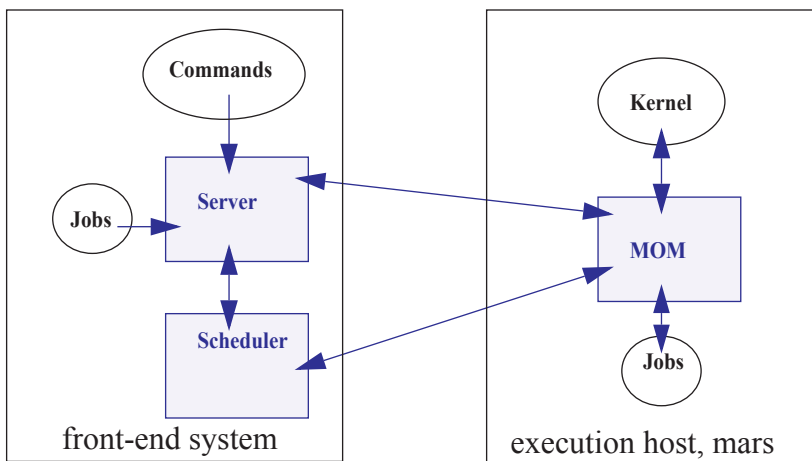
For this example, let's assume we have a 32-CPU server machine named "mars". We want users to log into mars and jobs will be run via PBS on mars.

In this configuration, the server's default `nodes` file (which should contain the name of the host on which the Server was installed) is sufficient. Our example `nodes` file would contain only one entry: `mars`

The default MOM and Scheduler `config` files, as well as the default queue/Server limits are also sufficient in order to run jobs. No changes are required from the default configuration, however, you may wish to customize PBS to your site.

10.2 Separate Server and Execution Host

A variation on the model presented above would be to provide a “front-end” system that ran the PBS Server and Scheduler, and from which users submitted their jobs. Only the MOM would run on our execution server, mars. This model is recommended when the user load would otherwise interfere with the computational load on the Server.



In this case, the PBS `server_priv/nodes` file would contain the name of our execution server mars, but this may not be what was written to the file during installation, depending on which options were selected. It is possible the hostname of the machine on which the Server was installed was added to the file, in which case you would need to use `qmgr (1B)` to manipulate the contents to contain one vnode: mars. If the default scheduling policy, based on available CPUs and memory, meets your requirements, then no changes are required in either the MOM or Scheduler configuration files.

However, if you wish the execution host (mars) to be scheduled based on load average, the following changes are needed. Edit MOM's `mom_priv/config` file so that it contains the target and maximum load averages:

```
$ideal_load 30
$max_load 32
```

In the Scheduler `sched_priv/sched_config` file, the following options would need to be set:

```
load_balancing: True all
```

10.3 Multiple Execution Hosts

The multi-vnode complex model is a very common configuration for PBS. In this model, there is typically a front-end system as we saw in the previous example, with a number of back-end execution hosts. The PBS Server and Scheduler are typically run on the front-end system, and a MOM is run on each of the execution hosts, as shown in the diagram to the right.

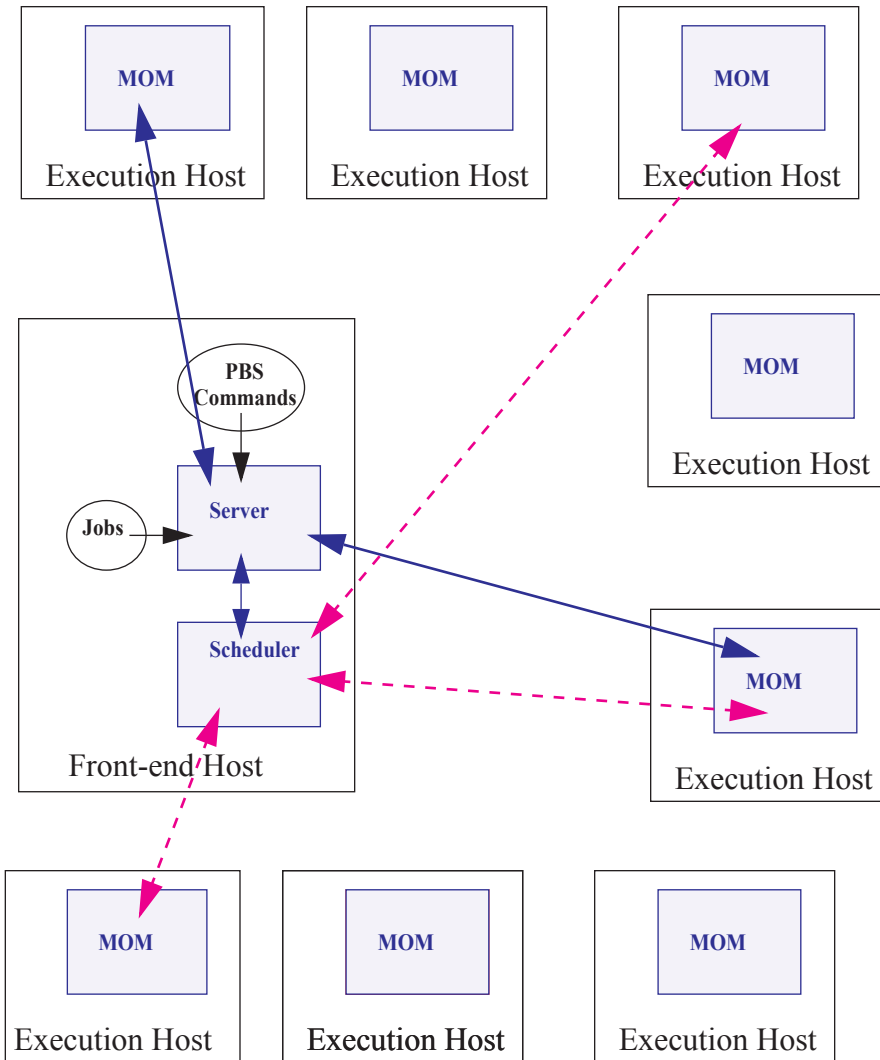
In this model, the server's `nodes` file will need to contain the list of all the vnodes in the complex.

The MOM `config` file on each vnode will need two static resources added, to specify the target load for each vnode. If we assume each of the vnodes in our “planets” cluster is a 32-processor system, then the following example shows what might be desirable ideal and maximum load values to add to the MOM `config` files:

```
$ideal_load 30
$max_load 32
```

Furthermore, suppose we want the Scheduler to load balance the workload across the available vnodes, making sure not to run two jobs in a row on the same vnode (round robin vnode scheduling). We accomplish this by editing the Scheduler configuration file and enabling load balancing:

```
load_balancing: True all
smp_cluster_dist: round_robin
```



This diagram illustrates a multi-vnode complex configuration wherein the Scheduler and Server communicate with the MOMs on the execution hosts. Jobs are submitted to the Server, scheduled for execution by the Scheduler, and then transferred to a MOM when it's time to be run. MOM periodically sends status information back to the Server, and answers resource requests from the Scheduler.

10.4 Complex Multi-level Route Queues

There are times when a site may wish to create a series of route queues in order to filter jobs, based on specific resources, or possibly to different destinations. For this example, consider a site that has two large Server systems, and a Linux cluster. The Administrator wants to configure route queues such that everyone submits jobs to a single queue, but the jobs get routed based on (1) requested architecture and (2) individual group IDs. In other words, users request the architecture they want, and PBS finds the right queue for them. Only groups “math”, “chemistry”, and “physics” are permitted to use either server systems; while anyone can use the cluster. Lastly, the jobs coming into the cluster should be divided into three separate queues for long, short, and normal jobs. But the “long” queue was created for the astronomy department, so only members of that group should be permitted into that queue. Given these requirements, let’s look at how we would set up such a collection of route queues. (Note that this is only one way to accomplish this task. There are various other ways too.)

First we create a queue to which everyone will submit their jobs. Let’s call it “submit”. It will need to be a route queue with three destinations, as shown:

```
Qmgr: create queue submit
Qmgr: set queue submit queue_type = Route
Qmgr: set queue submit route_destinations = server_1
Qmgr: set queue submit route_destinations += server_2
Qmgr: set queue submit route_destinations += cluster
Qmgr: set queue submit enabled = True
Qmgr: set queue submit started = True
```

Now we need to create the destination queues. (Notice in the above example, we have already decided what to call the three destinations: `server_1`, `server_2`, `cluster`.) First we create the `server_1` queue, complete with a group ACL, and a specific architecture limit.

```
Qmgr: create queue server_1
Qmgr: set queue server_1 queue_type = Execution
Qmgr: set queue server_1 from_route_only = True
Qmgr: set queue server_1 resources_max.arch = linux
Qmgr: set queue server_1 resources_min.arch = linux
Qmgr: set queue server_1 acl_group_enable = True
Qmgr: set queue server_1 acl_groups = math
Qmgr: set queue server_1 acl_groups += chemistry
Qmgr: set queue server_1 acl_groups += physics
Qmgr: set queue server_1 enabled = True
Qmgr: set queue server_1 started = True
```

Next we create the queues for `server_2` and `cluster`. Note that the `server_2` queue is very similar to the `server_1` queue, only the architecture differs. Also notice that the `cluster` queue is another route queue, with multiple destinations.

```
Qmgr: create queue server_2  
Qmgr: set queue server_2 queue_type = Execution  
Qmgr: set queue server_2 from_route_only = True  
Qmgr: set queue server_2 resources_max.arch = sv2  
Qmgr: set queue server_2 resources_min.arch = sv2  
Qmgr: set queue server_2 acl_group_enable = True  
Qmgr: set queue server_2 acl_groups = math  
Qmgr: set queue server_2 acl_groups += chemistry  
Qmgr: set queue server_2 acl_groups += physics  
Qmgr: set queue server_2 enabled = True  
Qmgr: set queue server_2 started = True  
Qmgr: create queue cluster  
Qmgr: set queue cluster queue_type = Route  
Qmgr: set queue cluster from_route_only = True  
Qmgr: set queue cluster resources_max.arch = linux  
Qmgr: set queue cluster resources_min.arch = linux  
Qmgr: set queue cluster route_destinations = long  
Qmgr: set queue cluster route_destinations += short  
Qmgr: set queue cluster route_destinations += medium  
Qmgr: set queue cluster enabled = True  
Qmgr: set queue cluster started = True
```

In the cluster queue above, you will notice the particular order of the three destination queues (long, short, medium). PBS will attempt to route a job into the destination queues in the order specified. Thus, we want PBS to first try the long queue (which will have an ACL on

it), then the `short` queue (with its short time limits). Thus any jobs that had not been routed into any other queues (server or cluster) will end up in the `medium` cluster queue. Now to create the remaining queues.

```
Qmgr: create queue long
Qmgr: set queue long queue_type = Execution
Qmgr: set queue long from_route_only = True
Qmgr: set queue long resources_max.cput = 20:00:00
Qmgr: set queue long resources_max.walltime = 20:00:00
Qmgr: set queue long resources_min.cput = 02:00:00
Qmgr: set queue long resources_min.walltime = 03:00:00
Qmgr: set queue long acl_group_enable = True
Qmgr: set queue long acl_groups = astrology
Qmgr: set queue long enabled = True
Qmgr: set queue long started = True

Qmgr: create queue short
Qmgr: set queue short queue_type = Execution
Qmgr: set queue short from_route_only = True
Qmgr: set queue short resources_max.cput = 01:00:00
Qmgr: set queue short resources_max.walltime = 01:00:00
Qmgr: set queue short enabled = True
Qmgr: set queue short started = True
Qmgr: create queue medium
Qmgr: set queue medium queue_type = Execution
Qmgr: set queue medium from_route_only = True
Qmgr: set queue medium enabled = True
Qmgr: set queue medium started = True
Qmgr: set server default_queue = submit
```

Notice that the `long` and `short` queues have time limits specified. This will ensure that jobs of certain sizes will enter (or be prevented from entering) these queues. The last queue, `medium`, has no limits, thus it will be able to accept any job that is not routed into any other queue.

Lastly, note the last line in the example above, which specified that the default queue is the new `submit` queue. This way users will simply submit their jobs with the resource and architecture requests, without specifying a queue, and PBS will route the job into the correct location. For example, if a user submitted a job with the following syntax, the job would be routed into the `server_2` queue:

```
qsub -l select=arch=sv2:ncpus=4 testjob
```

10.5 External Software License Management

PBS Professional can be configured to schedule jobs based on externally-controlled licensed software. A detailed example is provided in [section 5.14.7.3.ii, "Example of Floating, Externally-managed License with Features"](#) on page 343 in the [PBS Professional Administrator's Guide](#).

10.6 Multiple User ACL Example

A site may have a need to restrict individual users to particular queues. In the previous example we set up queues with group-based ACLs, in this example we show user-based ACLs. Say a site has two different groups of users, and wants to limit them to two separate queues (perhaps with different resource limits). The following example illustrates this.

```
Qmgr: create queue structure
Qmgr: set queue structure queue_type = Execution
Qmgr: set queue structure acl_user_enable = True
Qmgr: set queue structure acl_users = curly
Qmgr: set queue structure acl_users += jerry
Qmgr: set queue structure acl_users += larry
Qmgr: set queue structure acl_users += moe
Qmgr: set queue structure acl_users += tom
Qmgr: set queue structure resources_max.nodes = 48
Qmgr: set queue structure enabled = True
Qmgr: set queue structure started = True

Qmgr: create queue engine
Qmgr: set queue engine queue_type = Execution
Qmgr: set queue engine acl_user_enable = True
Qmgr: set queue engine acl_users = bill
Qmgr: set queue engine acl_users += bobby
Qmgr: set queue engine acl_users += chris
Qmgr: set queue engine acl_users += jim
Qmgr: set queue engine acl_users += mike
Qmgr: set queue engine acl_users += rob
Qmgr: set queue engine acl_users += scott
Qmgr: set queue engine resources_max.nodes = 12
Qmgr: set queue engine resources_max.walltime=04:00:00
Qmgr: set queue engine enabled = True
Qmgr: set queue engine started = True
```

Chapter 11

Run Limit Error Messages

This chapter lists the error messages generated when limits are exceeded. See [section 5.15.1, "Managing Resource Usage By Users, Groups, and Projects, at Server & Queues"](#) on page 360 in the PBS Professional Administrator's Guide.

11.1 Run Limit Error Messages

When a job would exceed a limit by running, the job's comment field is set to one of the following messages. The following table shows the limit attribute, where the limit is applied, to whom the limit is applied, and the message.

Table 11-1: Job Run Limit Error Messages

Attribute	Where Applied	To Whom Applied	Message
max_run	queue	o: PBS_ALL	Not Running: Queue <Q> job limit has been reached.
max_run	server	o: PBS_ALL	Not Running: Server job limit has been reached.
max_run	server	p:PBS_GENERIC	Not Running: Project has reached server running limit.
max_run	queue	p:PBS_GENERIC	Not Running: Project has reached queue<queue-name>'s running limit.

Table 11-1: Job Run Limit Error Messages

Attribute	Where Applied	To Whom Applied	Message
max_run	server	p:<project name>	Not Running: Server job limit reached for project <project name>
max_run	queue	p:<project name>	Not Running: Queue <queue-name> job limit reached for project <project name>
max_run	queue	g: PBS_GENERIC	Not Running: Group has reached queue <Q> running limit.
max_run	server	g: PBS_GENERIC	Not Running: Group has reached server running limit.
max_run	queue	u: PBS_GENERIC	Not Running: User has reached queue <Q> running job limit.
max_run	server	u: PBS_GENERIC	Not Running: User has reached server running job limit.
max_run	queue	g:<group name>	Queue <Q> job limit reached for group <G>
max_run	server	g:<group name>	Server job limit reached for group <G>
max_run	queue	u:<user name>	Queue <Q> job limit reached for user <U>
max_run	server	u:<user name>	Server job limit reached for user <U>
max_run_res	queue	o: PBS_ALL	Queue <Q> job limit reached on resource <res>
max_run_res	server	o: PBS_ALL	Server job limit reached on resource <res>
max_run_res	queue	p:PBS_GENERIC	Not Running: Queue <queue-name> per-project limit reached on resource <res>
max_run_res	server	p:PBS_GENERIC	Not Running: Server per-project limit reached on resource <res>

Table 11-1: Job Run Limit Error Messages

Attribute	Where Applied	To Whom Applied	Message
max_run_res	server	p:<project name>	Not Running: would exceed project <project_name>'s limit on resource <res> in complex
max_run_res	queue	p:<project name>	Not Running: would exceed project <project_name>'s limit on resource <res> in queue <queue-name>
max_run_res	queue	g: PBS_GENERIC	Queue <Q> per-group limit reached on resource <res>
max_run_res	server	g: PBS_GENERIC	Server per-group limit reached on resource <res>
max_run_res	queue	u: PBS_GENERIC	Queue <Q> per-user limit reached on resource <res>
max_run_res	server	u: PBS_GENERIC	Server per-user limit reached on resource <res>
max_run_res	queue	g:<group name>	would exceed group <G>'s limit on resource <res> in queue <Q>
max_run_res	server	g:<group name>	would exceed group <G>'s limit on resource <res> in complex
max_run_res	queue	u:<user name>	would exceed user <U>'s limit on resource <res> in queue <Q>
max_run_res	server	u:<user name>	would exceed user <U>'s limit on resource <res> in complex

Error Codes

The following table lists all the PBS error codes, their textual names, and a description of each.

Table 12-1: Error Codes

Error Name	Error Code	Description
PBSE_NONE	0	No error
PBSE_UNKJOBID	15001	Unknown Job Identifier
PBSE_NOATTR	15002	Undefined Attribute
PBSE_ATTRRO	15003	Attempt to set READ ONLY attribute
PBSE_IVALREQ	15004	Invalid request
PBSE_UNKREQ	15005	Unknown batch request
PBSE_TOOMANY	15006	Too many submit retries
PBSE_PERM	15007	No permission
PBSE_BADHOST	15008	Access from host not allowed
PBSE_JOBEXIST	15009	Job already exists

Table 12-1: Error Codes

Error Name	Error Code	Description
PBSE_SYSTEM	15010	System error occurred
PBSE_INTERNAL	15011	Internal Server error occurred
PBSE_REGROUTE	15012	Parent job of dependent in route queue
PBSE_UNKSIG	15013	Unknown signal name
PBSE_BADATVAL	15014	Bad attribute value
PBSE_MODATRRUN	15015	Cannot modify attribute in run state
PBSE_BADSTATE	15016	Request invalid for job state
PBSE_UNKQUE	15018	Unknown queue name
PBSE_BADCRED	15019	Invalid Credential in request
PBSE_EXPIRED	15020	Expired Credential in request
PBSE_QUNOENB	15021	Queue not enabled
PBSE_QACCESS	15022	No access permission for queue
PBSE_BADUSER	15023	Missing userID, username, or GID. Returned under following conditions: 1. User does not have a password entry (getpwnam() returns null). 2. User's UID is zero and root isn't allowed to run jobs (acl_roots). 3. PBS_O_HOST is not set in the job.
PBSE_HOPCOUNT	15024	Max hop count exceeded
PBSE_QUEEXIST	15025	Queue already exists
PBSE_ATTRTYPE	15026	Incompatible queue attribute type

Table 12-1: Error Codes

Error Name	Error Code	Description
PBSE_OBJBUSY	15027	Object Busy
PBSE_QUENBIG	15028	Queue name too long
PBSE_NOSUP	15029	Feature/function not supported
PBSE_QUENOEN	15030	Can't enable queue, lacking definition
PBSE_PROTOCOL	15031	Protocol (ASN.1) error. Message is distorted or truncated.
PBSE_BADATLST	15032	Bad attribute list structure
PBSE_NOCONNECTS	15033	No free connections
PBSE_NOSERVER	15034	No Server to connect to
PBSE_UNKRESC	15035	Unknown resource
PBSE_EXCQRESC	15036	Job exceeds Queue resource limits
PBSE_QUENODFLT	15037	No Default Queue Defined
PBSE_NORERUN	15038	Job Not Rerunnable
PBSE_ROUTEREJ	15039	Route rejected by all destinations
PBSE_ROUTEEXPD	15040	Time in Route Queue Expired
PBSE_MOMREJECT	15041	Request to MOM failed
PBSE_BADSCRIPT	15042	(qsub) Cannot access script file
PBSE_STAGEIN	15043	Stage In of files failed
PBSE_RESCUNAV	15044	Resources temporarily unavailable
PBSE_BADGRP	15045	Bad Group specified
PBSE_MAXQUED	15046	Max number of jobs in queue
PBSE_CKPBSY	15047	Checkpoint Busy, may be retries

Table 12-1: Error Codes

Error Name	Error Code	Description
PBSE_EXLIMIT	15048	Limit exceeds allowable
PBSE_BADACCT	15049	Bad Account attribute value
PBSE_ALRDYEXIT	15050	Job already in exit state
PBSE_NOCOPYFILE	15051	Job files not copied
PBSE_CLEANEOUT	15052	Unknown job id after clean init
PBSE_NOSYNCMSTR	15053	No Master in Sync Set
PBSE_BADDEPEND	15054	Invalid dependency
PBSE_DUPLIST	15055	Duplicate entry in List
PBSE_DISPROTO	15056	Bad DIS based Request Protocol
PBSE_EXECTHERE (Obsolete)	15057	Cannot execute there (Obsolete; no longer used.)
PBSE_SISREJECT	15058	Sister rejected
PBSE_SISCOMM	15059	Sister could not communicate
PBSE_SVRDOWN	15060	Request rejected -server shutting down
PBSE_CKPSHORT	15061	Not all tasks could checkpoint
PBSE_UNKNODE	15062	Named vnode is not in the list
PBSE_UNKNODEATR	15063	Vnode attribute not recognized
PBSE_NONODES	15064	Server has no vnode list
PBSE_NODENBIG	15065	Node name is too big
PBSE_NODEEXIST	15066	Node name already exists
PBSE_BADNDATVAL	15067	Bad vnode attribute value

Table 12-1: Error Codes

Error Name	Error Code	Description
PBSE_MUTUALEX	15068	State values are mutually exclusive
PBSE_GMODERR	15069	Error(s) during global mod of vnodes
PBSE_NORELYMOM	15070	Could not contact MOM
Reserved	15076	Not used.
PBSE_TOOLATE	15077	Reservation submitted with a start time that has already passed
PBSE_genBatchReq	15082	Batch request generation failed
PBSE_mgrBatchReq	15083	qmgr batch request failed
PBSE_UNKRESVID	15084	Unknown reservation ID
PBSE_delProgress	15085	Delete already in progress
PBSE_BADTSPEC	15086	Bad time specification(s)
PBSE_RESVMSG	15087	So reply_text can return a msg
PBSE_BADNODESPEC	15089	Node(s) specification error
PBSE_LICENSECPU	15090	Licensed CPUs exceeded
PBSE_LICENSEINV	15091	License is invalid
PBSE_RESVAUTH_H	15092	Host not authorized to make AR
PBSE_RESVAUTH_G	15093	Group not authorized to make AR
PBSE_RESVAUTH_U	15094	User not authorized to make AR
PBSE_R_UID	15095	Bad effective UID for reservation
PBSE_R_GID	15096	Bad effective GID for reservation
PBSE_IBMSPSWITCH	15097	IBM SP Switch error
PBSE_LICENSEUNAV	15098	Floating License unavailable

Table 12-1: Error Codes

Error Name	Error Code	Description
	15099	UNUSED
PBSE_RESCNOTSTR	15100	Resource is not of type string
PBSE_SSIGNON_UNSET_REJECT	15101	rejected if SVR_ssignon_enable not set
PBSE_SSIGNON_SET_REJECT	15102	rejected if SVR_ssignon_enable set
PBSE_SSIGNON_BAD_TRANSITION1	15103	bad attempt: true to false
PBSE_SSIGNON_NOCONNECT_DEST	15105	couldn't connect to destination host during a user migration request
PBSE_SSIGNON_NO_PASSWORD	15106	no per-user/per-server password
PBSE_MaxArraySize	15107	max array size exceeded
PBSE_INVALSELECTRESC	15108	resource invalid in select spec
PBSE_INVALJOBRESC	15109	invalid job resource
PBSE_INVALNODEPLACE	15110	node invalid w/place select
PBSE_PLACENOSELECT	15111	cannot have place w/o select
PBSE_INDIRECTHOP	15112	too many indirect resource levels
PBSE_INDIRECTBT	15113	target resource undefined
PBSE_NGBLUEGENE	15114	No node_group_enable on Blue-Gene
PBSE_NODESTALE	15115	Cannot change state of stale vnode
PBSE_DUPRESC	15116	cannot dup resource within a chunk
PBSE_CONNFULL	15117	server connection table full
PBSE_LICENSE_MIN_BADVAL	15118	bad value for pbs_license_min
PBSE_LICENSE_MAX_BADVAL	15119	bad value for pbs_license_max

Table 12-1: Error Codes

Error Name	Error Code	Description
PBSE_LICENSE_LINGER_BADVAL	15120	bad value for pbs_license_linger_time
PBSE_LICENSE_SERVER_DOWN	15121	License server is down
PBSE_LICENSE_BAD_ACTION	15122	Not allowed action with licensing
PBSE_BAD_FORMULA	15123	invalid sort formula
PBSE_BAD_FORMULA_KW	15124	invalid keyword in formula
PBSE_BAD_FORMULA_TYPE	15125	invalid resource type in formula
PBSE_BAD_RRULE_YEARLY	15126	reservation duration exceeds 1 year
PBSE_BAD_RRULE_MONTHLY	15127	reservation duration exceeds 1 month
PBSE_BAD_RRULE_WEEKLY	15128	reservation duration exceeds 1 week
PBSE_BAD_RRULE_DAILY	15129	reservation duration exceeds 1 day
PBSE_BAD_RRULE_HOURLY	15130	reservation duration exceeds 1 hour
PBSE_BAD_RRULE_MINUTELY	15131	reservation duration exceeds 1 minute
PBSE_BAD_RRULE_SECONDLY	15132	reservation duration exceeds 1 second
PBSE_BAD_RRULE_SYNTAX	15133	invalid recurrence rule syntax
PBSE_BAD_RRULE_SYNTAX2	15134	invalid recurrence rule syntax
PBSE_BAD_ICAL_TZ	15135	Undefined timezone info directory
PBSE_HOOKERROR	15136	error encountered related to hooks
PBSE_NEEDQUET	15137	need queue type set

Table 12-1: Error Codes

Error Name	Error Code	Description
PBSE_ETEERROR	15138	not allowed to alter attribute when <code>eligible_time_enable</code> is off
PBSE_HISTJOBID	15139	History job ID
PBSE_JOBHISTNOTSET	15140	<code>job_history_enable</code> not SET
PBSE_MIXENTLIMS	15141	mixing old and new limit enforcement
	15145	Server host not allowed to be provisioned
	15146	While provisioning, provisioning attributes can't be modified
	15147	State of provisioning vnode can't be changed
	15148	Vnode can't be deleted while provisioning
	15149	Attempt to set an AOE that is not in <code>resources_available.aoe</code>
	15150	Illegal job/reservation submission/alteration
PBSE_MOM_INCOMPLETE_HOOK	15167	Execution hooks not fully transferred to a particular MoM
PBSE_MOM_REJECT_ROOT_SCRIPTS	15168	A MoM has rejected a request to copy a hook-related file, or a job script to be executed by root
PBSE_HOOK_REJECT	15169	A MoM received a reject result from a mom hook
PBSE_HOOK_REJECT_RERUNJOB	15170	Hook rejection requiring a job to be rerun

Table 12-1: Error Codes

Error Name	Error Code	Description
PBSE_HOOK_REJECT_DELETEJOB	15171	Hook rejection requiring a job to be deleted
Resource monitor specific error codes		
PBSE_RMUNKNOWN	15201	Resource unknown
PBSE_RMBADPARAM	15202	Parameter could not be used
PBSE_RMNOPARAM	15203	A needed parameter did not exist
PBSE_RMEXIST	15204	Something specified didn't exist
PBSE_RMSYSTEM	15205	A system error occurred
PBSE_RMPART	15206	Only part of reservation made
PBSE_SSIGNON_BAD_TRANSITION2	15207	bad attempt: false to true
PBSE_ALPSRELERR	15209	PBS is unable to release the ALPS reservation

Chapter 13

Request Codes

When reading the PBS event logfiles, you may see messages of the form “Type 19 request received from PBS_Server...”. These “type codes” correspond to different PBS batch requests. The following table lists all the PBS type codes and the corresponding request of each.

Table 13-1: Request Codes

0	PBS_BATCH_Connect
1	PBS_BATCH_QueueJob
2	UNUSED
3	PBS_BATCH_jobscript
4	PBS_BATCH_RdytoCommit
5	PBS_BATCH_Commit
6	PBS_BATCH_DeleteJob
7	PBS_BATCH_HoldJob
8	PBS_BATCH_LocateJob
9	PBS_BATCH_Manager
10	PBS_BATCH_MessJob
11	PBS_BATCH_ModifyJob

Table 13-1: Request Codes

12	PBS_BATCH_MoveJob
13	PBS_BATCH_ReleaseJob
14	PBS_BATCH_Rerun
15	PBS_BATCH_RunJob
16	PBS_BATCH_SelectJobs
17	PBS_BATCH_Shutdown
18	PBS_BATCH_SignalJob
19	PBS_BATCH_StatusJob
20	PBS_BATCH_StatusQue
21	PBS_BATCH_StatusSvr
22	PBS_BATCH_TrackJob
23	PBS_BATCH_AsyrunJob
24	PBS_BATCH_Rescq
25	PBS_BATCH_ReserveResc
26	PBS_BATCH_ReleaseResc
27	PBS_BATCH_FailOver
48	PBS_BATCH_StageIn
49	PBS_BATCH_AuthenUser
50	PBS_BATCH_OrderJob
51	PBS_BATCH_SelStat
52	PBS_BATCH_RegistDep
54	PBS_BATCH_CopyFiles
55	PBS_BATCH_DelFiles
56	PBS_BATCH_JobObit

Table 13-1: Request Codes

57	PBS_BATCH_MvJobFile
58	PBS_BATCH_StatusNode
59	PBS_BATCH_Disconnect
60	UNUSED
61	UNUSED
62	PBS_BATCH_JobCred
63	PBS_BATCH_CopyFiles_Cred
64	PBS_BATCH_DelFiles_Cred
65	PBS_BATCH_GSS_Context
66	UNUSED
67	UNUSED
68	UNUSED
69	UNUSED
70	PBS_BATCH_SubmitResv
71	PBS_BATCH_StatusResv
72	PBS_BATCH_DeleteResv
73	PBS_BATCH_UserCred
74	PBS_BATCH_UserMigrate
75	PBS_BATCH_ConfirmResv
80	PBS_BATCH_DefSchReply
81	PBS_BATCH_StatusSched
82	PBS_BATCH_StatusRsc

Chapter 14

PBS Environment Variables

14.1 PBS Environment Variables

The following table lists the PBS environment variables:

Table 14-1: PBS Environment Variables

Variable	Meaning
NCPUS	Number of threads, defaulting to number of CPUs, on the vnode
OMP_NUM_THREADS	Same as NCPUS.
PBS_ARRAY_ID	Identifier for job arrays. Consists of sequence number.
PBS_ARRAY_INDEX	Index number of subjob in job array.
PBS_CONF_FILE	Path to <code>pbs.conf</code>
PBS_CPUSET_DEDICATED	Set by <code>mpiexec</code> to assert exclusive use of resources in assigned cpuset.
PBS_ENVIRONMENT	Indicates job type: PBS_BATCH or PBS_INTERACTIVE
PBS_JOBCOOKIE	Unique identifier for inter-MOM job-based communication.

Table 14-1: PBS Environment Variables

Variable	Meaning
PBS_JOBDIR	Pathname of job-specific staging and execution directory
PBS_JOBID	The job identifier assigned to the job or job array by the batch system.
PBS_JOBNAME	The job name supplied by the user.
PBS_MOMPORT	Port number on which this job's MOMs will communicate.
PBS_NODEFILE	The filename containing a list of vnodes assigned to the job.
PBS_NODENUM	Logical vnode number of this vnode allocated to the job.
PBS_O_HOME	Value of HOME from submission environment.
PBS_O_HOST	The host name on which the qsub command was executed.
PBS_O_LANG	Value of LANG from submission environment
PBS_O_LOGNAME	Value of LOGNAME from submission environment
PBS_O_MAIL	Value of MAIL from submission environment
PBS_O_PATH	Value of PATH from submission environment
PBS_O_QUEUE	The original queue name to which the job was submitted.
PBS_O_SHELL	Value of SHELL from submission environment
PBS_O_SYSTEM	The operating system name where qsub was executed.
PBS_O_TZ	Value of TZ from submission environment
PBS_O_WORKDIR	The absolute path of directory where qsub was executed.

Table 14-1: PBS Environment Variables

Variable	Meaning
PBS_QUEUE	The name of the queue from which the job is executed.
PBS_TASKNUM	The task (process) number for the job on this vnode.
TMPDIR	The job-specific temporary directory for this job.

Chapter 15

File Listing

The following table lists all the PBS files and directories; owner and permissions are specific to UNIX systems.

Table 15-1: File Listing

Directory / File	Owner	Permission	Average Size
PBS_EXEC/	root	drwxr-xr-x	4096
PBS_EXEC/bin	root	drwxr-xr-x	4096
PBS_EXEC/bin/nqs2pbs	root	-rwxr-xr-x	16062
PBS_EXEC/bin/pbsdsh	root	-rwxr-xr-x	111837
PBS_EXEC/bin/pbsnodes	root	-rwxr-xr-x	153004
PBS_EXEC/bin/pbs_dataservice	root	-rwx-----	
PBS_EXEC/bin/pbs_hostn	root	-rwxr-xr-x	35493
PBS_EXEC/bin/pbs_rdel	root	-rwxr-xr-x	151973
PBS_EXEC/bin/pbs_rstat	root	-rwxr-xr-x	156884
PBS_EXEC/bin/pbs_rsub	root	-rwxr-xr-x	167446
PBS_EXEC/bin/pbs_tclsh	root	-rwxr-xr-x	857552

Table 15-1: File Listing

Directory / File	Owner	Permission	Average Size
PBS_EXEC/bin/pbs_wish	root	-rwxr-xr-x	1592236
PBS_EXEC/bin/printjob	root	-rwxr-xr-x	42667
PBS_EXEC/bin/qalter	root	-rwxr-xr-x	210723
PBS_EXEC/bin/qdel	root	-rwxr-xr-x	164949
PBS_EXEC/bin/qdisable	root	-rwxr-xr-x	139559
PBS_EXEC/bin/qenable	root	-rwxr-xr-x	139558
PBS_EXEC/bin/qhold	root	-rwxr-xr-x	165368
PBS_EXEC/bin/qmgr	root	-rwxr-xr-x	202526
PBS_EXEC/bin/qmove	root	-rwxr-xr-x	160932
PBS_EXEC/bin/qmsg	root	-rwxr-xr-x	160408
PBS_EXEC/bin/qorder	root	-rwxr-xr-x	146393
PBS_EXEC/bin/qrerun	root	-rwxr-xr-x	157228
PBS_EXEC/bin/qrls	root	-rwxr-xr-x	165361
PBS_EXEC/bin/grun	root	-rwxr-xr-x	160978
PBS_EXEC/bin/qselect	root	-rwxr-xr-x	163266
PBS_EXEC/bin/qsig	root	-rwxr-xr-x	160083
PBS_EXEC/bin/qstart	root	-rwxr-xr-x	139589
PBS_EXEC/bin/qstat	root	-rwxr-xr-x	207532
PBS_EXEC/bin/qstop	root	-rwxr-xr-x	139584
PBS_EXEC/bin/qsub	root	-rwxr-xr-x	275460
PBS_EXEC/bin/qterm	root	-rwxr-xr-x	132188
PBS_EXEC/bin/tracejob	root	-rwxr-xr-x	64730

Table 15-1: File Listing

Directory / File	Owner	Permission	Average Size
PBS_EXEC/bin/xpbs	root	-rwxr-xr-x	817
PBS_EXEC/bin/xpbsmon	root	-rwxr-xr-x	817
PBS_EXEC/etc	root	drwxr-xr-x	4096
PBS_EXEC/etc/au-nodeupdate.pl	root	-rw-r--r--	
PBS_EXEC/etc/pbs_dedicated	root	-rw-r--r--	557
PBS_EXEC/etc/pbs_habitat	root	-rwx-----	10059
PBS_EXEC/etc/pbs_holidays	root	-rw-r--r--	1173
PBS_EXEC/etc/pbs_init.d	root	-rwx-----	5382
PBS_EXEC/etc/pbs_postinstall	root	-rwx-----	10059
PBS_EXEC/etc/ pbs_resource_group	root	-rw-r--r--	657
PBS_EXEC/etc/pbs_sched_config	root	-r--r--r--	9791
PBS_EXEC/etc/pbs_setlicense	root	-rwx-----	2118
PBS_EXEC/include	root	drwxr-xr-x	4096
PBS_EXEC/include/pbs_error.h	root	-r--r--r--	7543
PBS_EXEC/include/pbs_ifl.h	root	-r--r--r--	17424
PBS_EXEC/include/rm.h	root	-r--r--r--	740
PBS_EXEC/include/tm.h	root	-r--r--r--	2518
PBS_EXEC/include/tm_.h	root	-r--r--r--	2236
PBS_EXEC/lib	root	drwxr-xr-x	4096
PBS_EXEC/lib/libattr.a	root	-rw-r--r--	390274
PBS_EXEC/lib/liblog.a	root	-rw-r--r--	101230

Table 15-1: File Listing

Directory / File	Owner	Permission	Average Size
PBS_EXEC/lib/libnet.a	root	-rw-r--r--	145968
PBS_EXEC/lib/libpbs.a	root	-rw-r--r--	1815486
PBS_EXEC/lib/libsite.a	root	-rw-r--r--	132906
PBS_EXEC/lib/MPI	root	drwxr-xr-x	4096
PBS_EXEC/lib/MPI/ pbsrun.ch_gm.init.in	root	-rw-r--r--	9924
PBS_EXEC/lib/MPI/ pbsrun.ch_mx.init.in	root	-rw-r--r--	9731
PBS_EXEC/lib/MPI/ pbsrun.gm_mpd.init.in	root	-rw-r--r--	10767
PBS_EXEC/lib/MPI/ pbsrun.intelmpi.init.in	root	-rw-r--r--	10634
PBS_EXEC/lib/MPI/ pbsrun.mpich2.init.in	root	-rw-r--r--	10694
PBS_EXEC/lib/MPI/ pbsrun.mx_mpd.init.in	root	-rw-r--r--	10770
PBS_EXEC/lib/MPI/sgimPI.awk	root	-rw-r--r--	6564
PBS_EXEC/lib/pbs_sched.a	root	-rw-r--r--	822026
PBS_EXEC/lib/pm	root	drwxr--r--	4096
PBS_EXEC/lib/pm/PBS.pm	root	-rw-r--r--	3908
PBS_EXEC/lib/xpbs	root	drwxr-xr-x	4096
PBS_EXEC/lib/xpbs/ pbs_acctname.tk	root	-rw-r--r--	3484
PBS_EXEC/lib/xpbs/ pbs_after_depend.tk	root	-rw-r--r--	8637

Table 15-1: File Listing

Directory / File	Owner	Permission	Average Size
PBS_EXEC/lib/xpbs/pbs_auto_upd.tk	root	-rw-r--r--	3384
PBS_EXEC/lib/xpbs/pbs_before_depend.tk	root	-rw-r--r--	8034
PBS_EXEC/lib/xpbs/pbs_bin	root	drwxr-xr-x	4096
PBS_EXEC/lib/xpbs/pbs_bin/xpbs_datadump	root	-rwxr-xr-x	190477
PBS_EXEC/lib/xpbs/pbs_bin/xpbs_scriptload	root	-rwxr-xr-x	173176
PBS_EXEC/lib/xpbs/pbs_bindings.tk	root	-rw-r--r--	26029
PBS_EXEC/lib/xpbs/pbs_bitmaps	root	drwxr-xr-x	4096
PBS_EXEC/lib/xpbs/pbs_bitmaps/curve_down_arrow.bmp	root	-rw-r--r--	320
PBS_EXEC/lib/xpbs/pbs_bitmaps/curve_up_arrow.bmp	root	-rw-r--r--	314
PBS_EXEC/lib/xpbs/pbs_bitmaps/cyclist-only.xbm	root	-rw-r--r--	2485
PBS_EXEC/lib/xpbs/pbs_bitmaps/Downarrow.bmp	root	-rw-r--r--	299
PBS_EXEC/lib/xpbs/pbs_bitmaps/hourglass.bmp	root	-rw-r--r--	557
PBS_EXEC/lib/xpbs/pbs_bitmaps/iconize.bmp	root	-rw-r--r--	287
PBS_EXEC/lib/xpbs/pbs_bitmaps/logo.bmp	root	-rw-r--r--	67243

Table 15-1: File Listing

Directory / File	Owner	Permission	Average Size
PBS_EXEC/lib/xpbs/pbs_bitmaps/ maximize.bmp	root	-rw-r--r--	287
PBS_EXEC/lib/xpbs/pbs_bitmaps/ sm_down_arrow.bmp	root	-rw-r--r--	311
PBS_EXEC/lib/xpbs/pbs_bitmaps/ sm_up_arrow.bmp	root	-rw-r--r--	305
PBS_EXEC/lib/xpbs/pbs_bitmaps/ Uparrow.bmp	root	-rw-r--r--	293
PBS_EXEC/lib/xpbs/pbs_box.tk	root	-rw-r--r--	25912
PBS_EXEC/lib/xpbs/ pbs_button.tk	root	-rw-r--r--	18795
PBS_EXEC/lib/xpbs/ pbs_checkpoint.tk	root	-rw-r--r--	6892
PBS_EXEC/lib/xpbs/ pbs_common.tk	root	-rw-r--r--	25940
PBS_EXEC/lib/xpbs/ pbs_concur.tk	root	-rw-r--r--	8445
PBS_EXEC/lib/xpbs/ pbs_datetime.tk	root	-rw-r--r--	4533
PBS_EXEC/lib/xpbs/ pbs_email_list.tk	root	-rw-r--r--	3094
PBS_EXEC/lib/xpbs/pbs_entry.tk	root	-rw-r--r--	12389
PBS_EXEC/lib/xpbs/ pbs_fileselect.tk	root	-rw-r--r--	7975
PBS_EXEC/lib/xpbs/pbs_help	root	drwxr-xr-x	4096
PBS_EXEC/lib/xpbs/pbs_help/ after_depend.hlp	root	-rw-r--r--	1746

Table 15-1: File Listing

Directory / File	Owner	Permission	Average Size
PBS_EXEC/lib/xpbs/pbs_help/ auto_update.hlp	root	-rw-r--r--	776
PBS_EXEC/lib/xpbs/pbs_help/ before_depend.hlp	root	-rw-r--r--	1413
PBS_EXEC/lib/xpbs/pbs_help/ concur.hlp	root	-rw-r--r--	1383
PBS_EXEC/lib/xpbs/pbs_help/ datetime.hlp	root	-rw-r--r--	698
PBS_EXEC/lib/xpbs/pbs_help/ delete.hlp	root	-rw-r--r--	632
PBS_EXEC/lib/xpbs/pbs_help/ email.hlp	root	-rw-r--r--	986
PBS_EXEC/lib/xpbs/pbs_help/ fileselect.hlp	root	-rw-r--r--	1655
PBS_EXEC/lib/xpbs/pbs_help/ hold.hlp	root	-rw-r--r--	538
PBS_EXEC/lib/xpbs/pbs_help/ main.hlp	root	-rw-r--r--	15220
PBS_EXEC/lib/xpbs/pbs_help/ message.hlp	root	-rw-r--r--	677
PBS_EXEC/lib/xpbs/pbs_help/ misc.hlp	root	-rw-r--r--	4194
PBS_EXEC/lib/xpbs/pbs_help/ modify.hlp	root	-rw-r--r--	6034
PBS_EXEC/lib/xpbs/pbs_help/ move.hlp	root	-rw-r--r--	705

Table 15-1: File Listing

Directory / File	Owner	Permission	Average Size
PBS_EXEC/lib/xpbs/pbs_help/notes.hlp	root	-rw-r--r--	3724
PBS_EXEC/lib/xpbs/pbs_help/preferences.hlp	root	-rw-r--r--	1645
PBS_EXEC/lib/xpbs/pbs_help/release.hlp	root	-rw-r--r--	573
PBS_EXEC/lib/xpbs/pbs_help/select.acctname.hlp	root	-rw-r--r--	609
PBS_EXEC/lib/xpbs/pbs_help/select.checkpoint.hlp	root	-rw-r--r--	1133
PBS_EXEC/lib/xpbs/pbs_help/select.hold.hlp	root	-rw-r--r--	544
PBS_EXEC/lib/xpbs/pbs_help/select.jobname.hlp	root	-rw-r--r--	600
PBS_EXEC/lib/xpbs/pbs_help/select.owners.hlp	root	-rw-r--r--	1197
PBS_EXEC/lib/xpbs/pbs_help/select.priority.hlp	root	-rw-r--r--	748
PBS_EXEC/lib/xpbs/pbs_help/select.qtime.hlp	root	-rw-r--r--	966
PBS_EXEC/lib/xpbs/pbs_help/select.rerun.hlp	root	-rw-r--r--	541
PBS_EXEC/lib/xpbs/pbs_help/select.resources.hlp	root	-rw-r--r--	1490
PBS_EXEC/lib/xpbs/pbs_help/select.states.hlp	root	-rw-r--r--	562

Table 15-1: File Listing

Directory / File	Owner	Permission	Average Size
PBS_EXEC/lib/xpbs/pbs_help/signal.hlp	root	-rw-r--r--	675
PBS_EXEC/lib/xpbs/pbs_help/staging.hlp	root	-rw-r--r--	3702
PBS_EXEC/lib/xpbs/pbs_help/submit.hlp	root	-rw-r--r--	9721
PBS_EXEC/lib/xpbs/pbs_help/terminate.hlp	root	-rw-r--r--	635
PBS_EXEC/lib/xpbs/pbs_help/trackjob.hlp	root	-rw-r--r--	2978
PBS_EXEC/lib/xpbs/pbs_hold.tk	root	-rw-r--r--	3539
PBS_EXEC/lib/xpbs/pbs_jobname.tk	root	-rw-r--r--	3375
PBS_EXEC/lib/xpbs/pbs_listbox.tk	root	-rw-r--r--	10544
PBS_EXEC/lib/xpbs/pbs_main.tk	root	-rw-r--r--	24147
PBS_EXEC/lib/xpbs/pbs_misc.tk	root	-rw-r--r--	14526
PBS_EXEC/lib/xpbs/pbs_owners.tk	root	-rw-r--r--	4509
PBS_EXEC/lib/xpbs/pbs_pbs.tcl	root	-rw-r--r--	52524
PBS_EXEC/lib/xpbs/pbs_pref.tk	root	-rw-r--r--	3445
PBS_EXEC/lib/xpbs/pbs_preferences.tcl	root	-rw-r--r--	4323
PBS_EXEC/lib/xpbs/pbs_prefsave.tk	root	-rw-r--r--	1378

Table 15-1: File Listing

Directory / File	Owner	Permission	Average Size
PBS_EXEC/lib/xpbs/pbs_priority.tk	root	-rw-r--r--	4434
PBS_EXEC/lib/xpbs/pbs_qalter.tk	root	-rw-r--r--	35003
PBS_EXEC/lib/xpbs/pbs_qdel.tk	root	-rw-r--r--	3175
PBS_EXEC/lib/xpbs/pbs_qhold.tk	root	-rw-r--r--	3676
PBS_EXEC/lib/xpbs/pbs_qmove.tk	root	-rw-r--r--	3326
PBS_EXEC/lib/xpbs/pbs_qmsg.tk	root	-rw-r--r--	4032
PBS_EXEC/lib/xpbs/pbs_qrls.tk	root	-rw-r--r--	3674
PBS_EXEC/lib/xpbs/pbs_qsig.tk	root	-rw-r--r--	5171
PBS_EXEC/lib/xpbs/pbs_qsub.tk	root	-rw-r--r--	37466
PBS_EXEC/lib/xpbs/pbs_qterm.tk	root	-rw-r--r--	3204
PBS_EXEC/lib/xpbs/pbs_qtime.tk	root	-rw-r--r--	5790
PBS_EXEC/lib/xpbs/pbs_rerun.tk	root	-rw-r--r--	2802
PBS_EXEC/lib/xpbs/pbs_res.tk	root	-rw-r--r--	4807
PBS_EXEC/lib/xpbs/pbs_spinbox.tk	root	-rw-r--r--	7144
PBS_EXEC/lib/xpbs/pbs_staging.tk	root	-rw-r--r--	12183
PBS_EXEC/lib/xpbs/pbs_state.tk	root	-rw-r--r--	3657
PBS_EXEC/lib/xpbs/pbs_text.tk	root	-rw-r--r--	2738
PBS_EXEC/lib/xpbs/pbs_trackjob.tk	root	-rw-r--r--	13605
PBS_EXEC/lib/xpbs/pbs_wmgr.tk	root	-rw-r--r--	1428

Table 15-1: File Listing

Directory / File	Owner	Permission	Average Size
PBS_EXEC/lib/xpbs/tclIndex	root	-rw-r--r--	19621
PBS_EXEC/lib/xpbs/xpbs.src.tk	root	-rwxr-xr-x	9666
PBS_EXEC/lib/xpbs/xpbsrc	root	-rw-r--r--	2986
PBS_EXEC/lib/xpbsmon	root	drwxr-xr-x	4096
PBS_EXEC/lib/xpbsmon/ pbs_auto_upd.tk	root	-rw-r--r--	3281
PBS_EXEC/lib/xpbsmon/ pbs_bindings.tk	root	-rw-r--r--	9288
PBS_EXEC/lib/xpbsmon/ pbs_bitmaps	root	drwxr-xr-x	4096
PBS_EXEC/lib/xpbsmon/ pbs_bitmaps/cyclist-only.xbm	root	-rw-r--r--	2485
PBS_EXEC/lib/xpbsmon/ pbs_bitmaps/hourglass.bmp	root	-rw-r--r--	557
PBS_EXEC/lib/xpbsmon/ pbs_bitmaps/iconize.bmp	root	-rw-r--r--	287
PBS_EXEC/lib/xpbsmon/ pbs_bitmaps/logo.bmp	root	-rw-r--r--	67243
PBS_EXEC/lib/xpbsmon/ pbs_bitmaps/maximize.bmp	root	-rw-r--r--	287
PBS_EXEC/lib/xpbsmon/ pbs_box.tk	root	-rw-r--r--	15607
PBS_EXEC/lib/xpbsmon/ pbs_button.tk	root	-rw-r--r--	7543
PBS_EXEC/lib/xpbsmon/ pbs_cluster.tk	root	-rw-r--r--	44406

Table 15-1: File Listing

Directory / File	Owner	Permission	Average Size
PBS_EXEC/lib/xpbsmon/pbs_color.tk	root	-rw-r--r--	5634
PBS_EXEC/lib/xpbsmon/pbs_common.tk	root	-rw-r--r--	5716
PBS_EXEC/lib/xpbsmon/pbs_dialog.tk	root	-rw-r--r--	8398
PBS_EXEC/lib/xpbsmon/pbs_entry.tk	root	-rw-r--r--	10697
PBS_EXEC/lib/xpbsmon/pbs_expr.tk	root	-rw-r--r--	6163
PBS_EXEC/lib/xpbsmon/pbs_help	root	drwxr-xr-x	4096
PBS_EXEC/lib/xpbsmon/pbs_help/auto_update.hlp	root	-rw-r--r--	624
PBS_EXEC/lib/xpbsmon/pbs_help/main.hlp	root	-rw-r--r--	15718
PBS_EXEC/lib/xpbsmon/pbs_help/notes.hlp	root	-rw-r--r--	296
PBS_EXEC/lib/xpbsmon/pbs_help/pref.hlp	root	-rw-r--r--	1712
PBS_EXEC/lib/xpbsmon/pbs_help/prefQuery.hlp	root	-rw-r--r--	4621
PBS_EXEC/lib/xpbsmon/pbs_help/prefServer.hlp	root	-rw-r--r--	1409
PBS_EXEC/lib/xpbsmon/pbs_listbox.tk	root	-rw-r--r--	10640
PBS_EXEC/lib/xpbsmon/pbs_main.tk	root	-rw-r--r--	6760

Table 15-1: File Listing

Directory / File	Owner	Permission	Average Size
PBS_EXEC/lib/xpbsmon/pbs_node.tk	root	-rw-r--r--	60640
PBS_EXEC/lib/xpbsmon/pbs_pbs.tk	root	-rw-r--r--	7090
PBS_EXEC/lib/xpbsmon/pbs_pref.tk	root	-rw-r--r--	22117
PBS_EXEC/lib/xpbsmon/pbs_preferences.tcl	root	-rw-r--r--	10212
PBS_EXEC/lib/xpbsmon/pbs_prefsave.tk	root	-rw-r--r--	1482
PBS_EXEC/lib/xpbsmon/pbs_spinbox.tk	root	-rw-r--r--	7162
PBS_EXEC/lib/xpbsmon/pbs_system.tk	root	-rw-r--r--	47760
PBS_EXEC/lib/xpbsmon/pbs_wmgr.tk	root	-rw-r--r--	1140
PBS_EXEC/lib/xpbsmon/tclIndex	root	-rw-r--r--	30510
PBS_EXEC/lib/xpbsmon/xpbsmon.src.tk	root	-rwxr-xr-x	13999
PBS_EXEC/lib/xpbsmon/xpbsmonrc	root	-rw-r--r--	3166
PBS_EXEC/man	root	drwxr-xr-x	4096
PBS_EXEC/man/man1	root	drwxr-xr-x	4096
PBS_EXEC/man/man1/nqs2pbs	root	-rw-r--r--	3276
PBS_EXEC/man/man1/pbs.1B	root	-rw-r--r--	5376
PBS_EXEC/man/man1/pbsdsh.1B	root	-rw-r--r--	2978

Table 15-1: File Listing

Directory / File	Owner	Permission	Average Size
PBS_EXEC/man/man1/pbs_rdel.1B	root	-rw-r--r--	2342
PBS_EXEC/man/man1/pbs_rstat.1B	root	-rw-r--r--	2682
PBS_EXEC/man/man1/pbs_rsub.1B	root	-rw-r--r--	9143
PBS_EXEC/man/man1/qalter.1B	root	-rw-r--r--	21569
PBS_EXEC/man/man1/qdel.1B	root	-rw-r--r--	3363
PBS_EXEC/man/man1/qhold.1B	root	-rw-r--r--	4323
PBS_EXEC/man/man1/qmove.1B	root	-rw-r--r--	3343
PBS_EXEC/man/man1/qmsg.1B	root	-rw-r--r--	3244
PBS_EXEC/man/man1/qorder.1B	root	-rw-r--r--	3028
PBS_EXEC/man/man1/qrerun.1B	root	-rw-r--r--	2965
PBS_EXEC/man/man1/qrls.1B	root	-rw-r--r--	3927
PBS_EXEC/man/man1/qselect.1B	root	-rw-r--r--	12690
PBS_EXEC/man/man1/qsig.1B	root	-rw-r--r--	3817
PBS_EXEC/man/man1/qstat.1B	root	-rw-r--r--	15274
PBS_EXEC/man/man1/qsub.1B	root	-rw-r--r--	36435
PBS_EXEC/man/man1/xpbs.1B	root	-rw-r--r--	26956
PBS_EXEC/man/man1/xpbsmon.1B	root	-rw-r--r--	26365
PBS_EXEC/man/man3	root	drwxr-xr-x	4096
PBS_EXEC/man/man3/ pbs_alterjob.3B	root	-rw-r--r--	5475
PBS_EXEC/man/man3/ pbs_connect.3B	root	-rw-r--r--	3493

Table 15-1: File Listing

Directory / File	Owner	Permission	Average Size
PBS_EXEC/man/man3/ pbs_default.3B	root	-rw-r--r--	2150
PBS_EXEC/man/man3/ pbs_deljob.3B	root	-rw-r--r--	3081
PBS_EXEC/man/man3/ pbs_disconnect.3B	root	-rw-r--r--	1985
PBS_EXEC/man/man3/ pbs_geterrmsg.3B	root	-rw-r--r--	2473
PBS_EXEC/man/man3/ pbs_holdjob.3B	root	-rw-r--r--	3006
PBS_EXEC/man/man3/ pbs_manager.3B	root	-rw-r--r--	4337
PBS_EXEC/man/man3/ pbs_movejob.3B	root	-rw-r--r--	3220
PBS_EXEC/man/man3/ pbs_msgjob.3B	root	-rw-r--r--	2912
PBS_EXEC/man/man3/ pbs_orderjob.3B	root	-rw-r--r--	2526
PBS_EXEC/man/man3/ pbs_rerunjob.3B	root	-rw-r--r--	2531
PBS_EXEC/man/man3/ pbs_resreserve.3B	root	-rw-r--r--	4125
PBS_EXEC/man/man3/ pbs_rlsjob.3B	root	-rw-r--r--	3043
PBS_EXEC/man/man3/ pbs_runjob.3B	root	-rw-r--r--	3484

Table 15-1: File Listing

Directory / File	Owner	Permission	Average Size
PBS_EXEC/man/man3/ pbs_selectjob.3B	root	-rw-r--r--	7717
PBS_EXEC/man/man3/ pbs_sigjob.3B	root	-rw-r--r--	3108
PBS_EXEC/man/man3/ pbs_stagein.3B	root	-rw-r--r--	3198
PBS_EXEC/man/man3/ pbs_statjob.3B	root	-rw-r--r--	4618
PBS_EXEC/man/man3/ pbs_statnode.3B	root	-rw-r--r--	3925
PBS_EXEC/man/man3/ pbs_statque.3B	root	-rw-r--r--	4009
PBS_EXEC/man/man3/ pbs_statserver.3B	root	-rw-r--r--	3674
PBS_EXEC/man/man3/ pbs_submit.3B	root	-rw-r--r--	6320
PBS_EXEC/man/man3/ pbs_submitresv.3B	root	-rw-r--r--	3878
PBS_EXEC/man/man3/ pbs_terminate.3B	root	-rw-r--r--	3322
PBS_EXEC/man/man3/rpp.3B	root	-rw-r--r--	6476
PBS_EXEC/man/man3/tm.3B	root	-rw-r--r--	11062
PBS_EXEC/man/man7	root	drwxr-xr-x	4096
PBS_EXEC/man/man7/ pbs_job_attributes.7B	root	-rw-r--r--	15920
PBS_EXEC/man/man7/ pbs_node_attributes.7B	root	-rw-r--r--	7973

Table 15-1: File Listing

Directory / File	Owner	Permission	Average Size
PBS_EXEC/man/man7/ pbs_queue_attributes.7B	root	-rw-r--r--	11062
PBS_EXEC/man/man7/ pbs_resources.7B	root	-rw-r--r--	22124
PBS_EXEC/man/man7/ pbs_resv_attributes.7B	root	-rw-r--r--	11662
PBS_EXEC/man/man7/ pbs_server_attributes.7B	root	-rw-r--r--	14327
PBS_EXEC/man/man8	root	drwxr-xr-x	4096
PBS_EXEC/man/man8/mpiexec.8B	root	-rw-r--r--	4701
PBS_EXEC/man/man8/pbs- report.8B	root	-rw-r--r--	19221
PBS_EXEC/man/man8/pbsfs.8B	root	-rw-r--r--	3703
PBS_EXEC/man/man8/pbsnodes.8B	root	-rw-r--r--	3441
PBS_EXEC/man/man8/pbsrun.8B	root	-rw-r--r--	20937
PBS_EXEC/man/man8/ pbsrun_unwrap.8B	root	-rw-r--r--	2554
PBS_EXEC/man/man8/ pbsrun_wrap.8B	root	-rw-r--r--	3855
PBS_EXEC/man/man8/ pbs_attach.8B	root	-rw-r--r--	3790
PBS_EXEC/man/man8/pbs_hostn.8B	root	-rw-r--r--	2781
PBS_EXEC/man/man8/pbs_idled.8B	root	-rw-r--r--	2628
PBS_EXEC/man/man8/ pbs_lamboot.8B	root	-rw-r--r--	2739

Table 15-1: File Listing

Directory / File	Owner	Permission	Average Size
PBS_EXEC/man/man8/pbs_migrate_users.8B	root	-rw-r--r--	2519
PBS_EXEC/man/man8/pbs_mom.8B	root	-rw-r--r--	23496
PBS_EXEC/man/man8/pbs_mpihp.8B	root	-rw-r--r--	4120
PBS_EXEC/man/man8/pbs_mpilam.8B	root	-rw-r--r--	2647
PBS_EXEC/man/man8/pbs_mpirun.8B	root	-rw-r--r--	3130
PBS_EXEC/man/man8/pbs_password.8B	root	-rw-r--r--	3382
PBS_EXEC/man/man8/pbs_poe.8B	root	-rw-r--r--	3973
PBS_EXEC/man/man8/pbs_probe.8B	root	-rw-r--r--	3344
PBS_EXEC/man/man8/pbs_sched_cc.8B	root	-rw-r--r--	6731
PBS_EXEC/man/man8/pbs_server.8B	root	-rw-r--r--	7914
PBS_EXEC/man/man8/pbs_tclsh.8B	root	-rw-r--r--	2475
PBS_EXEC/man/man8/pbs_tmrsh.8B	root	-rw-r--r--	3556
PBS_EXEC/man/man8/pbs_wish.8B	root	-rw-r--r--	2123
PBS_EXEC/man/man8/printjob.8B	root	-rw-r--r--	2823
PBS_EXEC/man/man8/qdisable.8B	root	-rw-r--r--	3104
PBS_EXEC/man/man8/qenable.8B	root	-rw-r--r--	2937
PBS_EXEC/man/man8/qmgr.8B	root	-rw-r--r--	7282
PBS_EXEC/man/man8/qrun.8B	root	-rw-r--r--	2850

Table 15-1: File Listing

Directory / File	Owner	Permission	Average Size
PBS_EXEC/man/man8/qstart.8B	root	-rw-r--r--	2966
PBS_EXEC/man/man8/qstop.8B	root	-rw-r--r--	2963
PBS_EXEC/man/man8/qterm.8B	root	-rw-r--r--	4839
PBS_EXEC/man/man8/tracejob.8B	root	-rw-r--r--	4664
PBS_EXEC/pgsql	root	-rwxr-xr-x	
PBS_EXEC/sbin	root	drwxr-xr-x	4096
PBS_EXEC/sbin/pbs-report	root	-rwxr-xr-x	68296
PBS_EXEC/sbin/pbsfs	root	-rwxr-xr-x	663707
PBS_EXEC/sbin/pbs_demux	root	-rwxr-xr-x	38688
PBS_EXEC/sbin/pbs_idled	root	-rwxr-xr-x	99373
PBS_EXEC/sbin/pbs_iff	root	-rwsr-xr-x	133142
PBS_EXEC/sbin/pbs_mom	root	-rwx-----	839326
PBS_EXEC/sbin/pbs_mom.cpuset	root	-rwx-----	0
PBS_EXEC/sbin/pbs_mom.standard	root	-rwx-----	0
PBS_EXEC/sbin/pbs_probe	root	-rwsr-xr-x	83108
PBS_EXEC/sbin/pbs_rcp	root	-rwsr-xr-x	75274
PBS_EXEC/sbin/pbs_sched	root	-rwx-----	705478
PBS_EXEC/sbin/pbs_server	root	-rwx-----	1133650
PBS_EXEC/tcltk	root	drwxr-xr-x	4096
PBS_EXEC/tcltk/bin	root	drwxr-xr-x	4096
PBS_EXEC/tcltk/bin/tclsh8.3	root	-rw-r--r--	552763
PBS_EXEC/tcltk/bin/wish8.3	root	-rw-r--r--	1262257

Table 15-1: File Listing

Directory / File	Owner	Permission	Average Size
PBS_EXEC/tcltk/include	root	drwxr-xr-x	4096
PBS_EXEC/tcltk/include/tcl.h	root	-rw-r--r--	57222
PBS_EXEC/tcltk/include/tclDecls.h	root	-rw-r--r--	123947
PBS_EXEC/tcltk/include/tk.h	root	-rw-r--r--	47420
PBS_EXEC/tcltk/include/tkDecls.h	root	-rw-r--r--	80181
PBS_EXEC/tcltk/lib	root	drwxr-xr-x	4096
PBS_EXEC/tcltk/lib/libtcl8.3.a	root	-rw-r--r--	777558
PBS_EXEC/tcltk/lib/libtclstub8.3.a	root	-rw-r--r--	1832
PBS_EXEC/tcltk/lib/libtk8.3.a	root	-rw-r--r--	1021024
PBS_EXEC/tcltk/lib/libtkstub8.3.a	root	-rw-r--r--	3302
PBS_EXEC/tcltk/lib/tcl8.3	root	drwxr-xr-x	4096
PBS_EXEC/tcltk/lib/tclConfig.sh	root	-rw-r--r--	7076
PBS_EXEC/tcltk/lib/tk8.3	root	drwxr-xr-x	4096
PBS_EXEC/tcltk/lib/tkConfig.sh	root	-rw-r--r--	3822
PBS_EXEC/tcltk/license.terms	root	-rw-r--r--	2233
PBS_HOME	root	drwxr-xr-x	4096
PBS_HOME/aux	root	drwxr-xr-x	4096
PBS_HOME/checkpoint	root	drwx-----	4096

Table 15-1: File Listing

Directory / File	Owner	Permission	Average Size
PBS_HOME/datastore	data service account	-rwx-----	
PBS_HOME/mom_logs	root	drwxr-xr-x	4096
PBS_HOME/mom_priv	root	drwxr-x--x	4096
PBS_HOME/mom_priv/config	root	-rw-r--r--	18
PBS_HOME/mom_priv/jobs	root	drwxr-x--x	4096
PBS_HOME/mom_priv/mom.lock	root	-rw-r--r--	4
PBS_HOME/pbs_environment	root	-rw-r--r--	0
PBS_HOME/sched_logs	root	drwxr-xr-x	4096
PBS_HOME/sched_priv	root	drwxr-x---	4096
PBS_HOME/sched_priv/dedicated_time	root	-rw-r--r--	557
PBS_HOME/sched_priv/holidays	root	-rw-r--r--	1228
PBS_HOME/sched_priv/resource_group	root	-rw-r--r--	0
PBS_HOME/sched_priv/sched.lock	root	-rw-r--r--	4
PBS_HOME/sched_priv/sched_config	root	-rw-r--r--	6370
PBS_HOME/sched_priv/sched_out	root	-rw-r--r--	0
PBS_HOME/server_logs	root	drwxr-xr-x	4096
PBS_HOME/server_priv	root	drwxr-x---	4096
PBS_HOME/server_priv/accounting	root	drwxr-xr-x	4096

Table 15-1: File Listing

Directory / File	Owner	Permission	Average Size
PBS_HOME/server_priv/acl_groups	root	drwxr-x---	4096
PBS_HOME/server_priv/acl_hosts	root	drwxr-x---	4096
PBS_HOME/server_priv/acl_svr	root	drwxr-x---	4096
PBS_HOME/server_priv/acl_svr/managers	root	-rw-----	13
PBS_HOME/server_priv/acl_users	root	drwxr-x---	4096
PBS_HOME/server_priv/jobs	root	drwxr-x---	4096
PBS_HOME/server_priv/license_file	root	-rw-r--r--	34
PBS_HOME/server_priv/queues/newqueue	root	-rw-----	303
PBS_HOME/server_priv/queues/workq	root	-rw-----	303
PBS_HOME/server_priv/resource-def	root		
PBS_HOME/server_priv/server.lock	root	-rw-----	4
PBS_HOME/server_priv/svrlive	root	-rw-----	
PBS_HOME/server_priv/tracking	root	-rw-----	0
PBS_HOME/spool	root	drwxrwxrwt	4096
PBS_HOME/undelivered	root	drwxrwxrwt	4096

Chapter 16

Log Messages

The server, scheduler and MOM all write messages to their log files. Which messages are written depends upon each daemon's event mask. See [section 13.4, "Event Logging" on page 857 in the PBS Professional Administrator's Guide](#).

A few log messages are listed here.

RPP Retries

Logs	Server, scheduler, MOM
Level	0x0002
Form	date; time;event type; reporting daemon; event class; rpp_stats; total (pkts=<packets>, retries=<retries>, fails=<fails>) last <number of seconds> secs (pkts=<packets>,retries=<retries>, fails=<fails>)
Example	03/22/2006 15:20:44;0002; pbs_mom; Svr;rpp_stats; total (pkts=4321, retries=25, fails=3) last 3621 secs (pkts=43, retries=2, fails=0)

Explanation	<p>RPP packet retries, reported both for total number since daemon start (“total”) and since last log message (“last <seconds> secs”). Logged at most once per hour unless this hour’s retry count is 0. The number of seconds since the previous log message is shown in “last <seconds> secs”.</p> <p>pkts: number of RPP packets sent. In “total” group, this is since daemon start (in example, 4321). In “last” group, this is since previous log message (in example, 43).</p> <p>retries: number of RPP data packet retries. In “total” group, this is since daemon start (in example, 25). In “last” group, this is since previous log message (in example, 2).</p> <p>fails: number of failures reported to the caller of the RPP function. In “total” group, this is since daemon start (in example, 3). In “last” group, this is since previous log message (in example, 0).</p> <p>No log message if the number of fails and retries are zero.</p>
-------------	--

cput and mem Logged by Mother Superior

Logs	Mother Superior
Level	0x0100
Form	Date; Time; event class; reporting daemon; Job; Job ID; Hostname; cput; mem
Example	07/02/2007 19:47:14;0100;pbs_mom;Job;40.pepsi;pepsi cput= 0:00:00 mem=4756kb
Explanation	On job exit, Mother superior logs the amount of cput and mem used by this job on each node.

MOM Adds \$clienthost Address

Logs	MOM
Level	Event level 0x0002, event class Server
Form	Adding IP address XXX.XXX.XXX.XXX as authorized
Example	Adding IP address 127.0.0.1 as authorized

Explanation	When MOM starts up, she logs the addresses associated with a host listed in Mom's config file in \$clienthost statements. When MOM receives the list from the Server, addresses associated with other MOMs in the PBS complex will be listed. This occurs as soon as MOM and the Server establish communication and again whenever a node goes down and comes back up, or there is a change to the list of execution hosts (node added to or deleted from the complex). That event and the associated logging may occur at any time.
-------------	--

Scheduler: Job is Invalid

Logs	Scheduler
Level	0x0080
Form	Job is invalid - ignoring for this cycle
Example	Job is invalid - ignoring for this cycle
Explanation	Job failed a validity check such as 1) no egroup, euser, select, place, 2) in peer scheduling, pulling server is not a manager for furnishing server, 3) internal scheduler memory failure

Scheduler: Message Indicating Whether It Is Prime Time

Logs	Scheduler
Level	0x0100 (256)
Form	"It is *P*. It will end in XX seconds at MM/DD/YYYY HH:MM:SS"
Example	"It is prime time. It will end in 29 seconds at 03/10/2007 09:29:31"
Explanation	The scheduler is declaring whether the current time is prime time or non-prime time. The scheduler is stating when this period of prime time or non-prime time will end.

Jobs that Can Never Run

Logs	Scheduler
Level	0x0080

Form	“resource request is impossible to solve: job will never run”
Example	“resource request is impossible to solve: job will never run”
Explanation	The “most deserving” job can never run. Only printed when backfilling is on.

Resource Permission Flag Error

Logs	Server
Level	0x0080, 0x0100
Form	“It is invalid to set both flags 'r' and 'i'. Flag 'r' will be ignored.”
Example	“It is invalid to set both flags 'r' and 'i'. Flag 'r' will be ignored.”
Explanation	The “i” and “r” flags are incompatible. The “i” flag takes precedence.

Error During Evaluation of Tunable Formula

Logs	Scheduler
Level	0x0100
Form	1234.mars;Formula evaluation for job had an error. Zero value will be used
Example	1234.mars;Formula evaluation for job had an error. Zero value will be used
Explanation	Tunable formula produced error when evaluated.

Creation of Job-specific Directory

Logs	MOM
Level	0x0008
Form	“created the job directory <jobdir_root><unique_job_dir_name>”
Example	“created the job directory /Users/user1/pbsjobs/345.myhost”
Explanation	PBS created a job-specific execution and staging directory

Failure to Create Job-specific Directory

Logs	MOM
Level	0x0001
Form	“unable to create the job directory <unique_job_dir_name>”
Example	“unable to create the job directory /Users/user1/pbsjobs/345.myhost”
Explanation	The MOM was unable to create the job-specific staging and execution directory

Failure to Validate MOM’s \$jobdir_root

Logs	MOM
Level	0x0001
Form	"<file>[<linenum>]command “\$jobdir_root <full path>” failed, aborting"
Example	"config[3] command “\$jobdir_root /foodir” failed, aborting"
Explanation	\$jobdir_root exists in the MOM’s configuration file, and the MOM was unable to validate \$jobdir_root; MOM has aborted

Job Eligible Time

Logs	Server
Level	0x0400
Form	MM/DD/YYYY hh:mm:ss;log event;Server@hostname;Job;jobID;job accrued 23 secs of <previous sample type>, new accrue_type=<next_sample_type>, eligible_time=<current amount of eligible_time>
Example	08/07/2007 13:xx:yy;0040;Server@myhost;Job;163.myhost;job accrued 23 secs of eligible_time, new accrue_type=run_time, eligible_time=00:00:23
Explanation	Previous sample was of eligible time; next sample will be of run_time, job has accrued 23 seconds of eligible_time.

Invalid Syntax for Standing Reservation

Logs	Server
Level	0x0080
Form	pbs_rsub error: Undefined iCalendar syntax
Example	pbs_rsub error: Undefined iCalendar syntax
Explanation	Invalid syntax given to pbs_rsub for recurrence rule for standing reservation

Appendix A: License Agreement

CAUTION!

PRIOR TO INSTALLATION OR USE OF THE SOFTWARE YOU MUST CONSENT TO THE FOLLOWING SOFTWARE LICENSE TERMS AND CONDITIONS BY CLICKING THE “I ACCEPT” BUTTON BELOW. YOUR ACCEPTANCE CREATES A BINDING LEGAL AGREEMENT BETWEEN YOU AND ALTAIR. IF YOU DO NOT HAVE THE AUTHORITY TO BIND YOUR ORGANIZATION TO THESE TERMS AND CONDITIONS, YOU MUST CLICK “I DO NOT ACCEPT” AND THEN HAVE AN AUTHORIZED PARTY IN THE ORGANIZATION THAT YOU REPRESENT ACCEPT THESE TERMS.

IF YOU, OR THE ORGANIZATION THAT YOU REPRESENT, HAS A MASTER SOFTWARE LICENSE AGREEMENT (“MASTER SLA”) ON FILE AT THE CORPORATE HEADQUARTERS OF ALTAIR ENGINEERING, INC. (“ALTAIR”), THE MASTER SLA TAKES PRECEDENCE OVER THESE TERMS AND SHALL GOVERN YOUR USE OF THE SOFTWARE.

MODIFICATION(S) OF THESE SOFTWARE LICENSE TERMS IS EXPRESSLY PROHIBITED. ANY ATTEMPTED MODIFICATION(S) WILL BE NONBINDING AND OF NO FORCE OR EFFECT UNLESS EXPRESSLY AGREED TO IN WRITING BY AN AUTHORIZED CORPORATE OFFICER OF ALTAIR. ANY DISPUTE RELATING TO THE VALIDITY OF AN ALLEGED MODIFICATION SHALL BE DETERMINED IN ALTAIR’S SOLE DISCRETION.

Altair Engineering, Inc. - Software License Agreement

THIS SOFTWARE LICENSE AGREEMENT, including any Additional Terms (together with the “Agreement”), shall be effective as of the date of YOUR acceptance of these software license terms and conditions (the “Effective Date”) and is between ALTAIR ENGINEERING, INC., 1820 E. Big Beaver Road, Troy, MI 48083-2031, USA, a Michigan corporation (“Altair”), and YOU, or the organization on whose behalf you have authority to accept these terms (the “Licensee”). Altair and Licensee, intending to be legally bound, hereby agree as follows:

1. DEFINITIONS. In addition to terms defined elsewhere in this Agreement, the following terms shall have the meanings defined below for purposes of this Agreement:

Additional Terms. Additional Terms are those terms and conditions which are determined by an Altair Subsidiary to meet local market conditions.

Documentation. Documentation provided by Altair or its resellers on any media for use with the Products.

Execute. To load Software into a computer's RAM or other primary memory for execution by the computer.

Global Zone: Software is licensed based on three Global Zones: the Americas, Europe and Asia-Pacific. When Licensee has Licensed Workstations located in multiple Global Zones, which are connected to a single License (Network) Server, a premium is applied to the standard Software License pricing for a single Global Zone.

ISV/Independent Software Vendor. A software company providing its products, (“ISV Software”) to Altair's Licensees through the Altair License Management System using Altair License Units.

License Log File. A computer file providing usage information on the Software as gathered by the Software.

License Management System. The license management system (LMS) that accompanies the Software and limits its use in accordance with this Agreement, and which includes a License Log File.

License (Network) Server. A network file server that Licensee owns or leases located on Licensee's premises and identified by machine serial number and/or HostID on the Order Form.

License Units. A parameter used by the LMS to determine usage of the Software permitted under this Agreement at any one time.

Licensed Workstations. Single-user computers located in the same Global Zone(s) that Licensee owns or leases that are connected to the License (Network) Server via local area network or Licensee's private wide-area network.

Maintenance Release. Any release of the Products made generally available by Altair to its Licensees with annual leases, or those with perpetual licenses who have an active maintenance agreement in effect, that corrects programming errors or makes other minor changes to the Software. The fees for maintenance and support services are included in the annual license fee but perpetual licenses require a separate fee.

Order Form. Altair's standard form in either hard copy or electronic format that contains the specific parameters (such as identifying Licensee's contracting office, License Fees, Software, Support, and License (Network) Servers) of the transaction governed by this Agreement.

Products. Products include Altair Software, ISV Software, and/or Suppliers' software; and Documentation related to all of the forgoing.

Proprietary Rights Notices. Patent, copyright, trademark or other proprietary rights notices applied to the Products, packaging or media.

Software. The Altair software identified in the Order Form and any Updates or Maintenance Releases.

Subsidiary. Subsidiary means any partnership, joint venture, corporation or other form of enterprise in which a party possesses, directly or indirectly, an ownership interest of fifty percent (50%) or greater, or managerial or operational control.

Suppliers. Any person, corporation or other legal entity which may provide software or documents which are included in the Software.

Support. The maintenance and support services provided by Altair pursuant to this Agreement.

Templates. Human readable ASCII files containing machine-interpretable commands for use with the Software.

Term. The term of licenses granted under this Agreement. Annual licenses shall have a 12-month term of use unless stated otherwise on the Order Form. Perpetual licenses shall have a term of twenty-five years. Maintenance agreements for perpetual licenses have a 12-month term.

Update. A new version of the Products made generally available by Altair to its Licensees that includes additional features or functionalities but is substantially the same computer code as the existing Products.

2. LICENSE GRANT. Subject to the terms and conditions set forth in this Agreement, Altair hereby grants Licensee, and Licensee hereby accepts, a limited, non-exclusive, non-transferable license to: a) install the Products on the License (Network) Server(s) identified on the Order Form for use only at the sites identified on the Order Form; b) execute the Products on Licensed Workstations in accordance with the LMS for use solely by Licensee's employees, or its onsite Contractors who have agreed to be bound by the terms of this Agreement, for Licensee's internal business use on Licensed Workstations within the Global Zone(s) as iden-

tified on the Order Form and for the term identified on the Order Form; c) make backup copies of the Products, provided that Altair's and its Suppliers' and ISV's Proprietary Rights Notices are reproduced on each such backup copy; d) freely modify and use Templates, and create interfaces to Licensee's proprietary software for internal use only using APIs provided that such modifications shall not be subject to Altair's warranties, indemnities, support or other Altair obligations under this Agreement; and e) copy and distribute Documentation inside Licensee's organization exclusively for use by Licensee's employees and its onsite Contractors who have agreed to be bound by the terms of this Agreement. A copy of the License Log File shall be made available to Altair automatically on no less than a monthly basis. In the event that Licensee uses a third party vendor for information technology (IT) support, the IT company shall be permitted to access the Software only upon its agreement to abide by the terms of this Agreement. Licensee shall indemnify, defend and hold harmless Altair for the actions of its IT vendor(s).

3. RESTRICTIONS ON USE. Notwithstanding the foregoing license grant, Licensee shall not do (or allow others to do) any of the following: a) install, use, copy, modify, merge, or transfer copies of the Products, except as expressly authorized in this Agreement; b) use any back-up copies of the Products for any purpose other than to replace the original copy provided by Altair in the event it is destroyed or damaged; c) disassemble, decompile or “unlock”, reverse translate, reverse engineer, or in any manner decode the Software or ISV Software for any reason; d) sublicense, sell, lend, assign, rent, distribute, publicly display or publicly perform the Products or Licensee's rights under this Agreement; e) allow use outside the Global Zone(s) or User Sites identified on the Order Form; f) allow third parties to access or use the Products such as through a service bureau, wide area network, Internet location or time-sharing arrangement except as expressly provided in Section 2(b); g) remove any Proprietary Rights Notices from the Products; h) disable or circumvent the LMS provided with the Products; or (i) link any software developed, tested or supported by Licensee or third parties to the Products (except for Licensee's own proprietary software solely for Licensee's internal use).

4. OWNERSHIP AND CONFIDENTIALITY. Licensee acknowledges that all applicable rights in patents, copyrights, trademarks, service marks, and trade secrets embodied in the Products are owned by Altair and/or its Suppliers or ISVs. Licensee further acknowledges that the Products, and all copies thereof, are and shall remain the sole and exclusive property of Altair and/or its Suppliers and ISVs. This Agreement is a license and not a sale of the Products. Altair retains all rights in the Products not expressly granted to Licensee herein. Licensee acknowledges that the Products are confidential and constitute valuable assets and trade secrets of Altair and/or its Suppliers and ISVs. Licensee agrees to take the same precautions necessary to protect and maintain the confidentiality of the Products as it does to protect its own information of a confidential nature but in any event, no less than a reasonable degree of care, and shall not disclose or make them available to any person or entity except as expressly provided in this Agreement. Licensee shall promptly notify Altair in the event any unauthorized person obtains access to the Products. If Licensee is required by any governmental

authority or court of law to disclose Altair's or its ISV's or its Suppliers' confidential information, then Licensee shall immediately notify Altair before making such disclosure so that Altair may seek a protective order or other appropriate relief. Licensee's obligations set forth in Section 3 and Section 4 of this Agreement shall survive termination of this Agreement for any reason. Altair's Suppliers and ISVs, as third party beneficiaries, shall be entitled to enforce the terms of this Agreement directly against Licensee as necessary to protect Supplier's intellectual property or other rights.

Altair and its resellers providing support and training to licensed end users of the Products shall keep confidential all Licensee information provided to Altair in order that Altair may provide Support and training to Licensee. Licensee information shall be used only for the purpose of assisting Licensee in its use of the licensed Products. Altair agrees to take the same precautions necessary to protect and maintain the confidentiality of the Licensee information as it does to protect its own information of a confidential nature but in any event, no less than a reasonable degree of care, and shall not disclose or make them available to any person or entity except as expressly provided in this Agreement.

5. MAINTENANCE AND SUPPORT. **Maintenance.** Altair will provide Licensee, at no additional charge for annual licenses and for a maintenance fee for paid-up licenses, with Maintenance Releases and Updates of the Products that are generally released by Altair during the term of the licenses granted under this Agreement, except that this shall not apply to any Term or Renewal Term for which full payment has not been received. Altair does not promise that there will be a certain number of Updates (or any Updates) during a particular year. If there is any question or dispute as to whether a particular release is a Maintenance Release, an Update or a new product, the categorization of the release as determined by Altair shall be final. Licensee agrees to install Maintenance Releases and Updates promptly after receipt from Altair. Maintenance Releases and Updates are subject to this Agreement. Altair shall only be obligated to provide support and maintenance for the most current release of the Software and the most recent prior release. **Support.** Altair will provide support via telephone and email to Licensee at the fees, if any, as listed on the Order Form. If Support has not been procured for any period of time for paid-up licenses, a reinstatement fee shall apply. Support consists of responses to questions from Licensee's personnel related to the use of the then-current and most recent prior release version of the Software. Licensee agrees to provide Altair with sufficient information to resolve technical issues as may be reasonably requested by Altair. Licensee agrees to the best of its abilities to read, comprehend and follow operating instructions and procedures as specified in, but not limited to, Altair's Documentation and other correspondence related to the Software, and to follow procedures and recommendations provided by Altair in an effort to correct problems. Licensee also agrees to notify Altair of a programming error, malfunction and other problems in accordance with Altair's then current problem reporting procedure. If Altair believes that a problem reported by Licensee may not be due to an error in the Software, Altair will so notify Licensee. Questions must be directed to Altair's specially designated telephone support numbers and email addresses. Support will also be available via email at Internet addresses designated by Altair. Support is available

Monday through Friday (excluding holidays) from 8:00 a.m. to 5:00 p.m. local time in the Global Zone where licensed, unless stated otherwise on the Order Form. **Exclusions.** Altair shall have no obligation to maintain or support (a) altered, damaged or Licensee-modified Software, or any portion of the Software incorporated with or into other software not provided by Altair; (b) any version of the Software other than the current version of the Software or the immediately prior release of the Software; (c) problems caused by Licensee's negligence, abuse or misapplication of Software other than as specified in the Documentation, or other causes beyond the reasonable control of Altair; or (d) Software installed on any hardware, operating system version or network environment that is not supported by Altair. Support also **excludes** configuration of hardware, non-Altair Software, and networking services; consulting services; general solution provider related services; and general computer system maintenance.

6. WARRANTY AND DISCLAIMER. Altair warrants for a period of ninety (90) days after Licensee initially receives the Software that the Software will perform under normal use substantially as described in then current Documentation. Supplier software included in the Software and ISV Software provided to Licensee shall be warranted as stated by the Supplier or the ISV. Copies of the Suppliers' and ISV's terms and conditions of warranty are available on the Altair Support website. Support services shall be provided in a workmanlike and professional manner, in accordance with the prevailing standard of care for consulting support engineers at the time and place the services are performed.

ALTAIR DOES NOT REPRESENT OR WARRANT THAT THE PRODUCTS WILL MEET LICENSEE'S REQUIREMENTS OR THAT THEIR OPERATION WILL BE UNINTERRUPTED OR ERROR-FREE, OR THAT IT WILL BE COMPATIBLE WITH ANY PARTICULAR HARDWARE OR SOFTWARE. ALTAIR EXCLUDES AND DISCLAIMS ALL EXPRESS AND IMPLIED WARRANTIES NOT STATED HEREIN, INCLUDING THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NON-INFRINGEMENT. THE ENTIRE RISK FOR THE PERFORMANCE, NON-PERFORMANCE OR RESULTS OBTAINED FROM USE OF THE PRODUCTS RESTS WITH LICENSEE AND NOT ALTAIR. ALTAIR MAKES NO WARRANTIES WITH RESPECT TO THE ACCURACY, COMPLETENESS, FUNCTIONALITY, SAFETY, PERFORMANCE, OR ANY OTHER ASPECT OF ANY DESIGN, PROTOTYPE OR FINAL PRODUCT DEVELOPED BY LICENSEE USING THE PRODUCTS.

7. INDEMNITY. Altair will defend and indemnify, at its expense, any claim made against Licensee based on an allegation that the Software infringes a patent or copyright ("Claim"); provided, however, that this indemnification does not include claims which are based on Supplier software or ISV software, and that Licensee has not materially breached the terms of this Agreement, Licensee notifies Altair in writing within ten (10) days after Licensee first learns of the Claim; and Licensee cooperates fully in the defense of the claim. Altair shall have sole control over such defense; provided, however, that it may not enter into any settlement bind-

ing upon Licensee without Licensee's consent, which shall not be unreasonably withheld. If a Claim is made, Altair may modify the Software to avoid the alleged infringement, provided however, that such modifications do not materially diminish the Software's functionality. If such modifications are not commercially reasonable or technically possible, Altair may terminate this Agreement and refund to Licensee the prorated license fee that Licensee paid for the then current Term. Perpetual licenses shall be pro-rated over a 36-month term. Altair shall have no obligation under this Section 7, however, if the alleged infringement arises from Altair's compliance with specifications or instructions prescribed by Licensee, modification of the Software by Licensee, use of the Software in combination with other software not provided by Altair and which use is not specifically described in the Documentation, and if Licensee is not using the most current version of the Software, if such alleged infringement would not have occurred except for such exclusions listed here. This section 7 states Altair's entire liability to Licensee in the event a Claim is made. No indemnification is made for Supplier and/or ISV Software.

8. LIMITATION OF REMEDIES AND LIABILITY. Licensee's exclusive remedy (and Altair's sole liability) for Software that does not meet the warranty set forth in Section 6 shall be, at Altair's option, either (i) to correct the nonconforming Software within a reasonable time so that it conforms to the warranty; or (ii) to terminate this Agreement and refund to Licensee the license fees that Licensee has paid for the then current Term for the nonconforming Software; provided, however that Licensee notifies Altair of the problem in writing within the applicable Warranty Period when the problem first occurs. Any corrected Software shall be warranted in accordance with Section 6 for ninety (90) days after delivery to Licensee. The warranties hereunder are void if the Software has been improperly installed, misused, or if Licensee has violated the terms of this Agreement.

Altair's entire liability for all claims arising under or related in any way to this Agreement (regardless of legal theory), shall be limited to direct damages, and shall not exceed, in the aggregate for all claims, the license and maintenance fees paid under this Agreement by Licensee in the 12 months prior to the claim on a prorated basis, except for claims under Section 7. **ALTAIR AND ITS SUPPLIERS AND ISVS SHALL NOT BE LIABLE TO LICENSEE OR ANYONE ELSE FOR INDIRECT, SPECIAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES ARISING HEREUNDER (INCLUDING LOSS OF PROFITS OR DATA, DEFECTS IN DESIGN OR PRODUCTS CREATED USING THE SOFTWARE, OR ANY INJURY OR DAMAGE RESULTING FROM SUCH DEFECTS, SUFFERED BY LICENSEE OR ANY THIRD PARTY) EVEN IF ALTAIR OR ITS SUPPLIERS OR ITS ISVS HAVE BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.** Licensee acknowledges that it is solely responsible for the adequacy and accuracy of the input of data, including the output generated from such data, and agrees to defend, indemnify, and hold harmless Altair and its Suppliers and ISVs from any and all claims, including reasonable attorney's fees, resulting from, or in connection with Licensee's use of the Software. No

action, regardless of form, arising out of the transactions under this Agreement may be brought by either party against the other more than two (2) years after the cause of action has accrued, except for actions related to unpaid fees.

9. UNITED STATES GOVERNMENT RESTRICTED RIGHTS. This section applies to all acquisitions of the Products by or for the United States government. By accepting delivery of the Products except as provided below, the government or the party procuring the Products under government funding, hereby agrees that the Products qualify as “commercial” computer software as that term is used in the acquisition regulations applicable to this procurement and that the government's use and disclosure of the Products is controlled by the terms and conditions of this Agreement to the maximum extent possible. This Agreement supersedes any contrary terms or conditions in any statement of work, contract, or other document that are not required by statute or regulation. If any provision of this Agreement is unacceptable to the government, Vendor may be contacted at Altair Engineering, Inc., 1820 E. Big Beaver Road, Troy, MI 48083-2031; telephone (248) 614-2400. If any provision of this Agreement violates applicable federal law or does not meet the government's actual, minimum needs, the government agrees to return the Products for a full refund.

For procurements governed by DFARS Part 227.72 (OCT 1998), the Software, except as described below, is provided with only those rights specified in this Agreement in accordance with the Rights in Commercial Computer Software or Commercial Computer Software Documentation policy at DFARS 227.7202-3(a) (OCT 1998). For procurements other than for the Department of Defense, use, reproduction, or disclosure of the Software is subject to the restrictions set forth in this Agreement and in the Commercial Computer Software - Restricted Rights FAR clause 52.227-19 (June 1987) and any restrictions in successor regulations thereto.

Portions of Altair's PBS Professional Software and Documentation are provided with RESTRICTED RIGHTS. Use, duplication, or disclosure by the Government is subject to restrictions as set forth in subdivision(c)(1)(ii) of the rights in the Technical Data and Computer Software clause in DFARS 252.227-7013, or in subdivision (c)(1) and (2) of the Commercial Computer Software-Restricted Rights clause at 48 CFR52.227-19, as applicable.

10. CHOICE OF LAW AND VENUE. This Agreement shall be governed by and construed under the laws of the state of Michigan, without regard to that state's conflict of laws principles except if the state of Michigan adopts the Uniform Computer Information Transactions Act drafted by the National Conference of Commissioners of Uniform State Laws as revised or amended as of June 30, 2002 (“UCITA”) which is specifically excluded. This Agreement shall not be governed by the United Nations Convention on Contracts for the International Sale of Goods, the application of which is expressly excluded. Each Party waives its right to a jury trial in the event of any dispute arising under or relating to this Agreement. Each party agrees that money damages may not be an adequate remedy for breach of the provisions of

this Agreement, and in the event of such breach, the aggrieved party shall be entitled to seek specific performance and/or injunctive relief (without posting a bond or other security) in order to enforce or prevent any violation of this Agreement.

11. [RESERVED]

12. Notice. All notices given by one party to the other under the Agreement or these Additional Terms shall be sent by certified mail, return receipt requested, or by overnight courier, to the respective addresses set forth in this Agreement or to such other address either party has specified in writing to the other. All notices shall be deemed given upon actual receipt.

Written notice shall be made to:

Altair: Licensee Name & Address:

Altair Engineering, Inc. _____

1820 E. Big Beaver Rd _____

Troy, MI 48083 _____

Attn: Tom M. Perring Attn: _____

13. TERM. For annual licenses, or Support provided for perpetual licenses, renewal shall be automatic for each successive year (“Renewal Term”), upon mutual written execution of a new Order Form. All charges and fees for each Renewal Term shall be set forth in the Order Form executed for each Renewal Term. All Software licenses procured by Licensee may be made coterminous at the written request of Licensee and the consent of Altair.

14. TERMINATION. Either party may terminate this Agreement upon thirty (30) days prior written notice upon the occurrence of a default or material breach by the other party of its obligations under this Agreement (except for a breach by Altair of the warranty set forth in Section 8 for which a remedy is provided under Section 10; or a breach by Licensee of Section 5 or Section 6 for which no cure period is provided and Altair may terminate this Agreement immediately) if such default or breach continues for more than thirty (30) days after receipt of such notice. Upon termination of this Agreement, Licensee must cease using the Software and, at Altair's option, return all copies to Altair, or certify it has destroyed all such copies of the Software and Documentation.

15. GENERAL PROVISIONS. Export Controls. Licensee acknowledges that the Products may be subject to the export control laws and regulations of the United States and other countries, and any amendments thereof. Licensee agrees that Licensee will not directly or indirectly export the Products into any country or use the Products in any manner except in compliance with all applicable U.S. and other countries export laws and regulations. **Notice.** All notices given by one party to the other under this Agreement shall be sent by certified mail, return receipt requested, or by overnight courier, to the respective addresses set forth in this Agreement or to such other address either party has specified in writing to the other. All

notices shall be deemed given upon actual receipt. **Assignment.** Neither party shall assign this Agreement without the prior written consent of other party, which shall not be unreasonably withheld. All terms and conditions of this Agreement shall be binding upon and inure to the benefit of the parties hereto and their respective successors and permitted assigns. **Waiver.** The failure of a party to enforce at any time any of the provisions of this Agreement shall not be construed to be a waiver of the right of the party thereafter to enforce any such provisions. **Severability.** If any provision of this Agreement is found void and unenforceable, such provision shall be interpreted so as to best accomplish the intent of the parties within the limits of applicable law, and all remaining provisions shall continue to be valid and enforceable. **Headings.** The section headings contained in this Agreement are for convenience only and shall not be of any effect in constructing the meanings of the Sections. **Modification.** No change or modification of this Agreement will be valid unless it is in writing and is signed by a duly authorized representative of each party. **Conflict.** In the event of any conflict between the terms of this Agreement and any terms and conditions on a Licensee Purchase Order or comparable document, the terms of this Agreement shall prevail. Moreover, each party agrees any additional terms on any Purchase Order or comparable document other than the transaction items of (a) item(s) ordered; (b) pricing; (c) quantity; (d) delivery instructions and (e) invoicing directions, are not binding on the parties. In the event of a conflict between the terms of this Agreement, and the Additional Terms, the Agreement shall take precedence. **Entire Agreement.** This Agreement, the Additional Terms, and the Order Form(s) attached hereto constitute the entire understanding between the parties related to the subject matter hereto, and supersedes all proposals or prior agreements, whether written or oral, and all other communications between the parties with respect to such subject matter. This Agreement may be executed in one or more counterparts, all of which together shall constitute one and the same instrument. **Execution.** Copies of this Agreement executed via original signatures, facsimile or email shall be deemed binding on the parties.

Index

\$action [268](#)
\$min_check_poll [275](#)
\$prologalarm [275](#)
\$restrict_user_exceptions [276](#)
\$restrict_user_maxsysid [276](#)

A

accelerator [299](#)
accelerator_memory [299](#)
accelerator_model [300](#)
accept an action [1](#)
access
 by group [8](#)
 by user [23](#)
 from host [9](#)
 to a queue [1](#)
 to a reservation [1](#)
 to the Server [1](#)
access control list [1](#)
account [2](#)

Accounting

account [426](#), [430](#)
alt_id [426](#), [431](#)
authorized_groups [425](#)
authorized_hosts [425](#)
authorized_users [425](#)
ctime [424](#), [426](#), [431](#), [433](#)
duration [425](#)
end [424](#), [426](#), [431](#)
etime [426](#), [431](#), [433](#)
Exit_status [426](#), [430](#)
group [427](#), [432](#), [433](#)
jobname [427](#), [432](#), [433](#)
name [424](#)
owner [424](#)
qtime [427](#), [432](#), [433](#)
queue [424](#), [427](#), [432](#), [433](#)
Resource_List [425](#), [428](#), [432](#), [434](#)
session [429](#), [432](#), [434](#)
start [424](#), [429](#), [432](#), [434](#)
user [429](#), [432](#), [434](#)

accounting

account [2](#)

ACL [1](#), [435](#), [440](#), [441](#), [444](#)

action [2](#)

accept [1](#)

active (failover) [2](#)

Index

Active Directory [2](#)
Admin [2](#)
administrator [2](#)
Administrators [2](#)
advance reservation [2](#), [424](#), [425](#), [425](#), [429](#),
[453](#)
aggressive_provision [293](#)
AOE [3](#)
aoe [300](#)
API [3](#)
application checkpoint [3](#)
application operating environment [3](#)
arch [300](#)
array job [3](#), [10](#)
attribute
 definition [3](#)
 rerunnable [18](#)
avoid_provision [293](#)

B

backfill [282](#)
backfill_prime [282](#)
Backfilling [3](#)
batch job [10](#)
batch processing [3](#)
borrowing vnode [3](#)
built-in resource [3](#)
busy [416](#)
by_queue [283](#)

C

checkpoint [269](#), [425](#), [451](#), [472](#), [474](#), [486](#)
 restart [18](#)
 restart file [19](#)
 restart script [19](#)
checkpoint and abort [4](#)
checkpoint and restart [4](#)
checkpoint/restart [4](#)
checkpoint_abort [4](#), [269](#)
chunk [4](#)
cluster [4](#)

commands [4](#)
complex [4](#)
configuration file
 Version 1 [23](#)
 Version 2 [23](#)
consumable resource [4](#)
CPU [5](#)
cpus_per_ssinode [283](#)
cpuset_error_action [270](#)
cput [300](#)
custom resource [5](#)

D

dedicated_prefix [283](#)
degraded reservation [18](#)
delegation [5](#)
destination
 definition [5](#)
 identifier [5](#)
directive [6](#)
Domain Admin Account [6](#)
Domain Admins [6](#)
Domain User Account [6](#)
Domain Users [6](#)
down [417](#)

E

eligible_time [426](#), [431](#)
eligible_time_enable [318](#)
Enterprise Admins [6](#)
entity [6](#)
entity share [6](#)
Environment Variables [463](#)
error codes [449](#)
est_start_time_freq [318](#)
estimated [394](#)
exec_host [425](#)
exec_vnode [300](#)
executable [380](#)
execution host [6](#)
execution queue [7](#)

Index

Execution_Time [380](#)
express_queue [290](#)
externally-provided resources [267](#)

F

failover [7](#)
 idle [9](#)
 primary scheduler [17](#)
 primary server [17](#)
 secondary scheduler [19](#)
 secondary server [20](#)
fair_share [283](#), [286](#)
fair_share_perc [286](#)
Fairshare [7](#)
fairshare [290](#)
fairshare_enforce_no_shares [284](#)
fairshare_entity [283](#)
fairshare_usage_res [284](#)
File

 stage out [21](#)
file [300](#)

 stage in [21](#)
file staging [7](#)

Files

 MOM config [438](#)
 nodes [436](#)
finished jobs [7](#)
float [298](#)
floating license [7](#)
free [417](#)
furnishing queue [7](#)

G

global resource [8](#)
group [8](#)
 access [8](#)
 ID (GID) [8](#)
group limit [8](#)

H

half_life [284](#)

help_starving_jobs [284](#)
history jobs [8](#)
hold [9](#)
Hook [9](#)
hook

 provisioning [17](#)

hooks

 accept [1](#)
 action [2](#)
 reject action [18](#)

host [9](#), [300](#)

 access [9](#)

HPC Basic Profile (HPCBP) [9](#)

HPC Basic Profile Job [9](#)

HPC Basic Profile Server [9](#)

HPCBP Job [9](#)

HPCBP MOM [9](#)

HPS [121](#)

I

IBM HPS [121](#)

idle (failover) [9](#)

index

 subjob [22](#)

indirect resource [9](#)

InfiniBand [120](#), [121](#)

installation account [10](#)

instance [14](#)

Interactive job [10](#)

J

job

 attribute [18](#)

 batch [10](#)

 identifier [11](#)

 kill [12](#)

 owner [15](#)

 rerunnable [18](#)

 route [19](#)

 state [11](#)

Index

- job array [10](#)
 - identifier [11](#)
 - range [11](#)
 - subjob [21](#)
 - subjob index [22](#)
- job ID [11](#)
- Job Submission Description Language [11](#)
- Job Substates [412](#)
- job_priority [286](#)
- job_requeue_timeout [320](#)
- job_sort_key [285](#)
- job-busy [417](#)
- job-exclusive [417](#)
- jobs
 - moved [14](#)
- job-wide resource [11](#)
- Job-wide resource request [12](#)
- JSDL [11](#), [11](#)
- K**
- key [287](#)
- kill job [12](#)
- L**
- license
 - external [443](#)
 - token [22](#)
- license server [12](#)
- license server configuration
 - redundant [18](#)
- License Server List Configuration [12](#)
- Limit
 - Generic project limit [8](#)
 - Individual project limit [10](#)
 - overall [15](#)
- limit [12](#)
 - generic group limit [8](#)
 - generic user limit [8](#)
 - group limit [8](#)
 - individual group limit [10](#)
 - individual user limit [10](#)
 - project [17](#)
 - user limit [23](#)
- load balance [13](#)
- load_balancing [287](#)
- load_balancing_rr [287](#)
- local resource [13](#)
- log_filter [287](#)
- long [298](#)
- M**
- Manager [13](#)
- managing vnode [13](#)
- master provisioning script [13](#)
- master script [13](#)
- max_starve [287](#)
- max_walltime [301](#)
- mem [301](#)
- mem_per_ssinode [287](#)
- memory-only vnode [13](#)
- min_walltime [301](#)
- MOM [13](#)
 - subordinate [22](#)
- mom_resources [287](#)
- monitoring [13](#)
- Mother Superior [14](#)
- moved jobs [14](#)
- mpiprocs [301](#)
- mpparch [301](#)
- mppdepth [301](#)
- mpphost [302](#)
- mpplabels [302](#)
- mppmem [302](#)
- mppnodes [302](#)
- mppnppn [302](#)
- mppwidth [303](#)
- multi-node cluster [438](#)

Index

multinodebusy [269](#)

N

naccelerators [303](#)

nchunk [303](#)

NCPUS [463](#)

ncpus [303](#)

netwins [304](#)

nice [304](#)

node

 definition [14](#)

node_sort_key [288](#)

nodect [304](#)

nodes [304](#)

non-consumable resource [14](#)

nonprimetime_prefix [288](#)

normal_jobs [290](#)

nqs2pbs [31](#)

O

object [14](#)

occurrence of a standing reservation [14](#)

offline [417](#)

OMP_NUM_THREADS [463](#)

ompthreads [304](#)

Operator [15](#)

overall limit [15](#)

owner [15](#)

P

Parallel Operating Environment [121](#)

parameter [15](#)

PBS [464](#)

pbs [52](#)

PBS administrator [2](#), [2](#)

PBS entity [6](#), [15](#)

pbs Module [15](#)

PBS object [14](#), [15](#)

PBS Professional [16](#)

PBS_ARRAY_ID [463](#)

PBS_ARRAY_INDEX [463](#)

pbs_attach [45](#)

PBS_CONF_FILE [463](#)

PBS_CPUSSET_DEDICATED [463](#)

pbs_dataservice [46](#)

pbs_ds_password [47](#)

PBS_ENVIRONMENT [463](#)

PBS_EXEC [15](#)

PBS_HOME [15](#)

pbs_hostn [49](#)

pbs_idled [50](#)

pbs_interactive [53](#)

PBS_JOBCOOKIE [463](#)

PBS_JOBID [464](#)

PBS_JOBNAME [464](#)

pbs_lamboot [54](#)

pbs_migrate_users [55](#)

pbs_mkdirs [56](#)

pbs_mom [58](#)

pbs_mom_globus [64](#)

PBS_MOMPORT [464](#)

pbs_mpihp [65](#)

pbs_mpilam [66](#)

pbs_mpirun [67](#)

PBS_NODENUM [464](#)

PBS_O_HOME [464](#)

PBS_O_HOST [464](#)

PBS_O_LANG [464](#)

PBS_O_LOGNAME [464](#)

PBS_O_MAIL [464](#)

PBS_O_PATH [464](#)

PBS_O_QUEUE [464](#)

PBS_O_SHELL [464](#)

PBS_O_SYSTEM [464](#)

PBS_O_TZ [464](#)

PBS_O_WORKDIR [464](#)

pbs_password [69](#)

pbs_probe [71](#)

pbs_python [73](#)

PBS_QUEUE [465](#)

pbs_rdel [76](#)

pbs_renew [77](#)

pbs_rstat [78](#)

Index

pbs_rsub [79](#)
pbs_sched [88](#)
pbs_server [90](#)
PBS_TASKNUM [465](#)
pbs_tclsh [95](#)
pbs_tmrsh [96](#)
pbs_wish [98](#), [99](#)
pbsadmin [15](#)
PBScrayhost [306](#)
PBScraylabel_ [306](#)
PBScraynid [306](#)
PBScrayorder [307](#)
PBScrayseg [307](#)
pbsdsh [100](#)
pbsfs [102](#)
pbsnodes [105](#)
pbs-report [33](#)
pbsrun [107](#)
pbsrun_unwrap [124](#)
pbsrun_wrap [125](#)
pccput [304](#)
Peer scheduling [16](#)
placement
 set [16](#)
Placement pool [16](#)
Placement set series [16](#)
pmem [304](#)
POE [121](#)
poe [121](#)
policy [16](#)
 scheduling [19](#)
POSIX [16](#)
preempt [16](#), [16](#)
preempt_checkpoint [289](#)
preempt_fairshare [289](#)
preempt_order [289](#)
preempt_prio [290](#)
preempt_priority [286](#)
preempt_queue_prio [291](#)
preempt_requeue [291](#)
preempt_sort [291](#)
preempt_starving [291](#)

preempt_suspend [292](#)
preemption
 level [16](#)
 method [17](#)
preemptive_sched [289](#)
primary scheduler [17](#)
primary server [17](#)
prime_spill [292](#), [292](#)
primetime_prefix [292](#)
printjob [127](#)
Project [17](#)
project [427](#), [432](#), [433](#)
Project limit [17](#)
provision [17](#)
provision_policy [292](#)
provisioned vnode [17](#)
provisioning [417](#)
 hook [17](#)
provisioning tool [17](#)
pulling queue [17](#)
pvmmem [304](#)

Q

qalter [128](#)
qdel [143](#)
qdisable [146](#)
qenable [147](#)
qhold [149](#)
qmgr [151](#), [437](#)
qmove [171](#)
qmsg [173](#)
qorder [175](#)
qrerun [176](#)
qrls [178](#)
qrun [180](#)
qselect [183](#)
qsig [191](#)
qstart [193](#)
qstat [194](#)
qstop [208](#)
qsub [210](#)
qterm [229](#)

Index

queue

- access to a [1](#)
- definition [17](#)
- execution [7](#)
- furnishing [7](#)
- pulling [17](#)
- routing [19](#)

queue_softlimits [290](#)

queuing [18](#)

Quick Start Guide [vii](#)

R

redundant license server configuration [18](#)

reject an action [18](#)

reject_root_scripts [275](#)

request

- resource
 - job-wide [12](#)

requeue [18](#)

reservation [424](#)

- access to a [1](#)
- advance [2](#)
- degradation [18](#)
- degraded [5](#)
- instance [14](#)
- occurrence [14](#)
- soonest occurrence [20](#)
- standing [21](#)
 - instance [14](#)
 - soonest occurrence [20](#)

reservation attributes [424](#)

reservation degradation [18](#)

Resource [18](#)

resource

- built-in [3](#)
- consumable [4](#)
- custom [5](#)
- indirect [9](#)
- job-wide [11](#)
- non-consumable [14](#)
- shared [20](#)

resource request

- job-wide [12](#)

resource_assigned [428](#), [433](#)

Resource_List [425](#), [428](#), [432](#), [434](#)

resource_unset_infinite [293](#)

resources [293](#)

restart [18](#), [269](#)

restart file [19](#)

restart script [19](#)

RESV_BEING_DELETED [419](#)

RESV_CONFIRMED [419](#)

RESV_DEGRADED [419](#)

RESV_DELETED [420](#)

RESV_DELETING_JOBS [420](#)

RESV_FINISHED [420](#)

RESV_NONE [420](#)

RESV_RUNNING [420](#)

RESV_TIME_TO_RUN [420](#)

RESV_UNCONFIRMED [420](#)

RESV_WAIT [420](#)

resv-exclusive [417](#)

round_robin [294](#)

route [19](#)

route queue [435](#), [440](#)

routing queue [19](#)

S

Scheduler [19](#)

- policies [19](#)

scheduling

- policy [16](#), [19](#)

Schema Admins [19](#)

secondary scheduler [19](#)

secondary server [20](#)

sequence number [20](#)

Server [20](#)

- access to the [1](#)

server_dyn_res [294](#)

server_softlimits [290](#)

SGI

- Origin [436](#)

shared resource [20](#)

Index

shrink-to-fit job [20](#)
sister [20](#)
sisterhood [20](#)
size [298](#)
SMP [436](#)
smp_cluster_dist [294](#)
Snapshot checkpoint [20](#)
software [304](#)
soonest occurrence [20](#)
sort_by [295](#)
sort_priority [286](#)
sort_queues [295](#)
stage
 in [21](#)
 out [21](#)
Staging and execution directory [21](#)
stale [418](#)
standing reservation [21](#)
start_time [305](#)
starving_jobs [284](#), [290](#)
state [21](#)
 job [11](#)
state-unknown, down [418](#)
Strict ordering [21](#)
strict_fifo [295](#)
strict_ordering [295](#)
string [298](#)
string_array [298](#)
subject [21](#)
subjob [21](#)
subjob index [22](#)
subordinate MOM [22](#)
sync_time [296](#)

T

task [22](#)
task placement [22](#)
terminate [269](#)
Three-server Configuration [22](#)
time-sharing [436](#), [437](#), [438](#)
TMPDIR [465](#)
token [22](#)

tracejob [232](#)

U

UID [23](#)
unknown_shares [296](#)
US [121](#)
user
 access [23](#)
 definition [23](#)
 ID [23](#)
User Guide [vii](#)
user limit [23](#)
User Space (IBM HPS) [121](#)

V

vchunk [23](#)
Version 1 configuration file [23](#)
Version 2 configuration file [23](#)
vmem [305](#)
vnode [23](#), [305](#)
 borrowing [3](#)
 managing [13](#)
 memory-only [13](#)
vntype [305](#)

W

wait-provisioning [419](#)
walltime [305](#)

X

xpbs [235](#)
xpbsmon [250](#)

PBS Professional® 12.0

Programmer's Guide



PBS Works™

PBS Works is a division of  Altair

Altair PBS Professional 12 Programmer's Guide, updated 1/25/13

Copyright © 2003-2012 Altair Engineering, Inc. All rights reserved.

PBS™, PBS Works™, PBS GridWorks®, PBS Professional®, PBS Analytics™, PBS Catalyst™, e-Compute™, and e-Render™ are trademarks of Altair Engineering, Inc. and are protected under U.S. and international laws and treaties. All other marks are the property of their respective owners.

ALTAIR ENGINEERING INC. Proprietary and Confidential. Contains Trade Secret Information. Not for use or disclosure outside ALTAIR and its licensed clients. Information contained herein shall not be decompiled, disassembled, duplicated or disclosed in whole or in part for any purpose. Usage of the software is only as explicitly permitted in the end user software license agreement.

Copyright notice does not imply publication.

For documentation and the PBS Works forums, go to:

Web: www.pbsworks.com

For more information, contact Altair at:

Email: pbssales@altair.com

Technical Support

Location	Telephone	e-mail
North America	+1 248 614 2425	pbssupport@altair.com
China	+86 (0)21 6117 1666	es@altair.com.cn
France	+33 (0)1 4133 0992	francesupport@altair.com
Germany	+49 (0)7031 6208 22	hwsupport@altair.de
India	+91 80 66 29 4500	pbs-support@india.altair.com
Italy	+39 0832 315573 +39 800 905595	support@altairengineering.it
Japan	+81 3 5396 2881	pbs@altairjp.co.jp
Korea	+82 31 728 8600	support@altair.co.kr
Scandinavia	+46 (0)46 286 2050	support@altair.se
UK	+44 (0)1926 468 600	pbssupport@uk.altair.com

This document is proprietary information of Altair Engineering, Inc.

Table of Contents

Acknowledgements	vii
About PBS Documentation	ix
1 Introduction	1
1.1 Location of API Libraries	1
1.2 Location of Header Files	1
1.3 Example Compilation Line	1
1.4 Deprecations	1
2 Concepts and Components	3
2.1 PBS Components	4
3 Server Functions	7
3.1 General Identifiers	7
3.2 Batch Server Functions	10
3.3 Server Management	10
3.4 Queue Management	13
3.5 Job Management	13
3.6 Server to Server Requests	19
3.7 Deferred Services	22
3.8 Resource Management	26
3.9 Network Protocol	27
4 Batch Interface Library (IFL)	29
4.1 Interface Library Overview	29
4.2 Interface Library Routines	30

Table of Contents

5	RPP Library	79
5.1	RPP Library Routines	79
6	TM Library	85
6.1	TM Library Routines	85
7	RM Library	93
7.1	RM Library Routines	93
8	TCL/tk Interface	97
8.1	TCL/tk API Functions	97
9	Hooks	105
9.1	Introduction	105
9.2	How Hooks Work	106
9.3	Interface to Hooks	107
10	HPC Basic Profile	129
10.1	Introduction	129
10.2	How PBS Works With HPC Basic Profile	129
10.3	Examples	134
10.4	Caveats	142
10.5	See Also	142
	Appendix A: License Agreement	145
	Index	155

List of Manual Pages

<u>openrm, closerm, downrm, configrm, addreq, allreq, getreq, flushreq, activereq, fullresp</u>	<u>94</u>
<u>pbs_alterjob</u>	<u>31</u>
<u>pbs_connect</u>	<u>33</u>
<u>pbs_default</u>	<u>35</u>
<u>pbs_deljob</u>	<u>36</u>
<u>pbs_delresv</u>	<u>37</u>
<u>pbs_disconnect</u>	<u>38</u>
<u>pbs_geterrmsg</u>	<u>39</u>
<u>pbs_holdjob</u>	<u>40</u>
<u>pbs_locjob</u>	<u>41</u>
<u>pbs_manager</u>	<u>42</u>
<u>pbs_module</u>	<u>108</u>
<u>pbs_movejob</u>	<u>45</u>
<u>pbs_msgjob</u>	<u>46</u>
<u>pbs_orderjob</u>	<u>47</u>
<u>pbs_rerunjob</u>	<u>48</u>
<u>pbs_rescreserve, pbs_rescrelease</u>	<u>49</u>
<u>pbs_rlsjob</u>	<u>51</u>
<u>pbs_runjob, pbs_asyrunjob</u>	<u>52</u>
<u>pbs_selectjob</u>	<u>54</u>
<u>pbs_selstat</u>	<u>56</u>
<u>pbs_sigjob</u>	<u>59</u>
<u>pbs_stagein</u>	<u>60</u>
<u>pbs_statfree</u>	<u>61</u>
<u>pbs_stathook(3B)</u>	<u>126</u>
<u>pbs_statjob</u>	<u>62</u>
<u>pbs_statnode, pbs_statvnode, pbs_stathost</u>	<u>64</u>
<u>pbs_statque</u>	<u>66</u>
<u>pbs_statresv</u>	<u>68</u>
<u>pbs_statsched</u>	<u>70</u>
<u>pbs_statserver</u>	<u>72</u>

<u>pbs_submit</u>	<u>74</u>
<u>pbs_submit_resv</u>	<u>76</u>
<u>pbs_tclapi</u>	<u>98</u>
<u>pbs_terminate</u>	<u>78</u>
<u>rpp_open, rpp_bind, rpp_poll, rpp_io, rpp_read, rpp_write, rpp_close, rpp_getaddr,</u>	
<u>rpp_flush, rpp_terminate, rpp_shutdown, rpp_rcommit, rpp_wcommit, rpp_eom, rpp_getc,</u>	
<u>rpp_putc</u>	<u>80</u>
<u>tm_init, tm_nodeinfo, tm_poll, tm_notify, tm_spawn, tm_kill, tm_obit, tm_taskinfo,</u>	
<u>tm_atnode, tm_rescinfo, tm_publish, tm_subscribe, tm_finalize, tm_attach</u>	<u>86</u>

Acknowledgements

PBS Professional is the enhanced commercial version of the PBS software originally developed for NASA. The NASA version had a number of corporate and individual contributors over the years, for which the PBS developers and PBS community are most grateful. Below we provide formal legal acknowledgements to corporate and government entities, then special thanks to individuals.

The NASA version of PBS contained software developed by NASA Ames Research Center, Lawrence Livermore National Laboratory, and MRJ Technology Solutions. In addition, it included software developed by the NetBSD Foundation, Inc., and its contributors, as well as software developed by the University of California, Berkeley and its contributors.

Other contributors to the NASA version of PBS include Bruce Kelly and Clark Streeter of NERSC; Kent Crispin and Terry Heidelberg of LLNL; John Kochmar and Rob Pennington of *Pittsburgh Supercomputing Center*; and Dirk Grunwald of *University of Colorado, Boulder*. The ports of PBS to the Cray T3e and the IBM SP SMP were funded by *DoD USAERDC*, Major Shared Research Center; the port of PBS to the Cray SV1 was funded by DoD MSIC.

No list of acknowledgements for PBS would possibly be complete without special recognition of the first two beta test sites. Thomas Milliman of the *Space Sciences Center* of the *University of New Hampshire* was the first beta tester. Wendy Lin of *Purdue University* was the second beta tester and holds the honor of submitting more problem reports than anyone else outside of NASA.

About PBS Documentation

Where to Keep the Documentation

To make cross-references work, put all of the PBS guides in the same directory.

What is PBS Professional?

PBS is a workload management system that provides a unified batch queuing and job management interface to a set of computing resources.

The PBS Professional Documentation

The documentation for PBS Professional includes the following:

PBS Professional Administrator's Guide:

Provides the PBS administrator with the information required to configure and manage PBS Professional (PBS).

PBS Professional Quick Start Guide:

Provides a quick overview of PBS Professional installation and license file generation.

PBS Professional Installation & Upgrade Guide:

Contains information on installing and upgrading PBS Professional.

PBS Professional User's Guide:

Covers user commands and how to submit, monitor, track, delete, and manipulate jobs.

PBS Professional Programmer's Guide:

Discusses the PBS application programming interface (API).

PBS Professional Reference Guide:

Contains PBS reference material.

PBS Manual Pages:

Describe PBS commands, resources, attributes, APIs

Ordering Software and Publications

To order additional copies of this manual and other PBS publications, or to purchase additional software licenses, contact your Altair sales representative. Contact information is included on the copyright page of this book.

Document Conventions

PBS documentation uses the following typographic conventions:

abbreviation

The shortest acceptable abbreviation of a command or subcommand is underlined.

`command`

Commands such as `qmgr` and `scp`

input

Command-line instructions

`manpage (x)`

File and path names. Manual page references include the section number in parentheses appended to the manual page name.

formats

Formats

Attributes

Attributes, parameters, objects, variable names, resources, types

Values

Keywords, instances, states, values, labels

Definitions

Terms being defined

Output

Output or example code

File contents

Chapter 1

Introduction

This book, the **Programmer's Guide** for PBS Professional, is provided to document the external application programming interfaces to the PBS Professional software.

1.1 Location of API Libraries

All of the libraries containing the PBS API are installed by default in `$PBS_EXEC/lib/`.

1.2 Location of Header Files

Header files used by customer-written code are found in `$PBS_EXEC/include`.

1.3 Example Compilation Line

An example of a compile command might look like the following:

```
cc mycode.c -I/usr/pbs/include -L/usr/pbs/lib -lpbs
```

1.4 Deprecations

The following are deprecated:

`pbs_tclapi`

`pbs_resquery`

Chapter 2

Concepts and Components

PBS is a distributed workload management system. As such, PBS handles the management and monitoring of the computational workload on a set of one or more computers. Modern workload/resource management solutions like PBS include the features of traditional batch queueing but offer greater flexibility and control than first generation batch systems (such as the original batch system NQS).

Workload management systems have three primary roles:

Queuing

The collecting together of work or tasks to be run on a computer. Users submit tasks or “jobs” to the resource management system where they are held until the system is ready to run them.

Scheduling

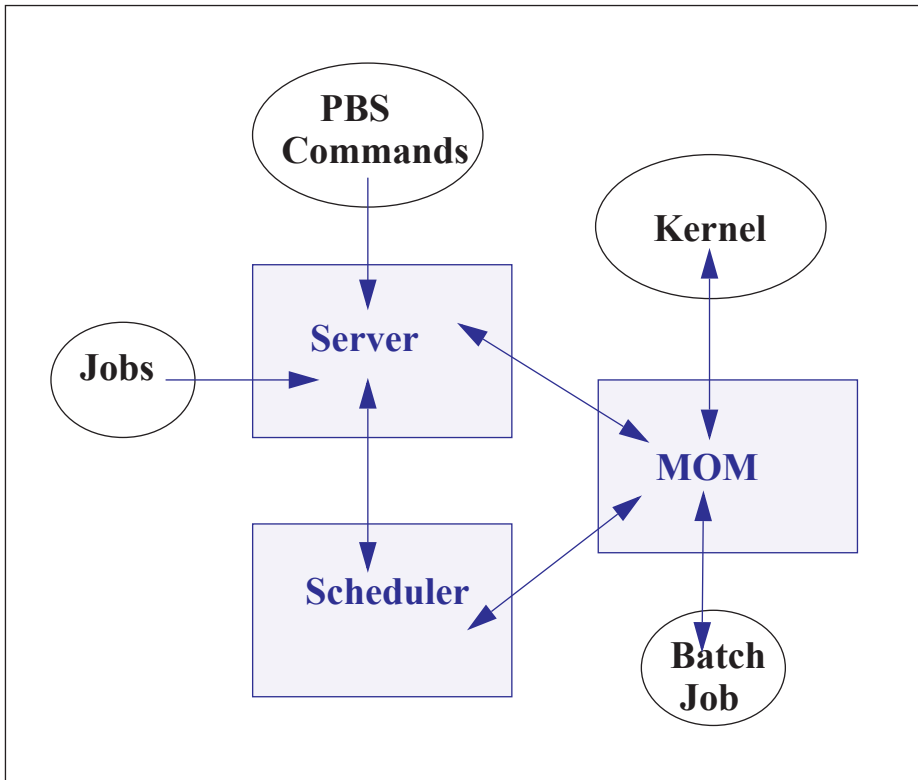
The process of selecting which jobs to run when and where, according to a predetermined policy. Sites balance competing needs and goals on the system(s) to maximize efficient use of resources (both computer time and people time).

Monitoring

The act of tracking and reserving system resources and enforcing usage policy. This covers both user-level and system-level monitoring as well as monitoring of the scheduling algorithms to see how well they are meeting the stated goals

2.1 PBS Components

PBS consist of two major component types: system daemons and user-level commands. A brief description of each is given here to help you make decisions during the installation process.



Job Server

The *Job Server* daemon process is the central focus for PBS. Within this document, it is generally referred to as *the Server* or by the execution name *pbs_server*. All commands and daemons communicate with the Server via an *Internet Protocol* (IP) network. The Server's main function is to provide the basic batch services such as receiving/creating a batch job, modifying the job, protecting the job against system crashes, and running the job. Typically there is one Server managing a given set of resources.

Job Executor (MOM)

The *Job Executor* is the daemon that actually places the job into execution. This daemon, *pbs_mom*, is informally called *MOM* as it is the mother of all executing jobs. (MOM is a reverse-engineered acronym that stands for Machine Oriented Miniserver.) MOM places a job into execution when it receives a copy of the job from a Server. MOM creates a new session that is as identical to a user login session as is possible. For example, if the user's login shell is `csh`, then MOM creates a session in which `.login` is run as well as `.cshrc`. MOM also has the responsibility for returning the job's output to the user when directed to do so by the Server. One MOM daemon runs on each computer which will execute PBS jobs.

Job Scheduler

The *Job Scheduler* daemon, *pbs_sched*, implements the site's policy controlling when each job is run and on which resources. The Scheduler communicates with the various MOMs to query the state of system resources and with the Server to learn about the availability of jobs to execute. The interface to the Server is through the same API as used by the client commands. Note that the Scheduler communicates with the Server with the same privilege as the PBS Manager.

Commands

PBS supplies both command line programs that are POSIX 1003.2d conforming and a graphical interface. These are used to submit, monitor, modify, and delete jobs. These *client commands* can be installed on any system type supported by PBS and do not require the local presence of any of the other components of PBS.

There are three classifications of commands: user commands (which any authorized user can use), operator commands, and manager (or administrator) commands. Operator and Manager commands require specific access privileges, as discussed in the **PBS Professional Administrator's Guide**.

Chapter 3

Server Functions

This chapter presents formal definitions for identifiers and names to be used throughout the remainder of this document, followed by detailed discussion of the various functions of the PBS Professional Server process.

3.1 General Identifiers

The following identifiers or names are referenced throughout this document. Unless otherwise noted, their usage will conform to the definition and syntax described in the following subsections and to the general rules described in the next paragraph. If allowed as part of the identifier, when entering the identifier string on the command line or in a PBS job script directive, embedded single or double quote marks must be escaped by enclosing the string in the other type of quote mark. Therefore, the string may not contain both types of quote marks. If white space is allowed in the identifier string, the string must be quoted when it is entered on the command line or in a PBS job directive.

3.1.1 Account String

An Account String is a string of characters that some Server implementations may use to provide addition accounting or charge information. The syntax is unspecified except that it must be a single string. When provided on the command line to a PBS utility or in a directive in a PBS job script, any embedded white space must be escaped by enclosing the string in quotes.

3.1.2 Attribute Name

An Attribute Name identifies an attribute or data item that is part of the information that makes up a job, queue, or Server. The name must consist of alphanumeric characters plus the underscore, `'_'`, character. It should start with an alphanumeric character. The length is not limited. The names recognized by PBS are listed in sections 2.2, 2.3, and 2.4.

3.1.3 Destination Identifiers

A destination identifier is a string used to specify a particular destination. The identifier may be specified in one of three forms:

queue@server_name

queue

@server_name

where *queue* is an ASCII character string of up to 15 characters. Valid characters are alphanumerics, the hyphen and the underscore. The string must begin with a letter. Queue is the name of a queue at the batch Server specified by *server_name*. That Server will interpret the *queue* string. If *queue* is omitted, a null string is assumed. *server_name* is a string identifying a Server; see *server_name*, below. If *server_name* is omitted, the default Server is assumed.

3.1.4 Default Server

When a Server is not specified to a client, the client will send batch requests to the Server identified as the default Server. A client identifies the default Server by (a) the setting of the environment variable `PBS_DEFAULT` which contains a Server name, or (b) by editing the `PBS_SERVER` variable in the `/etc/pbs.conf` file on the local host. Note that if both are present, `PBS_DEFAULT` overrides the `PBS_SERVER` specification.

3.1.5 Host Name

A Host Name is a string that identifies a host or system on the network. The syntax of the string must follow the rules established by the network. For IP, a host name is of the form *name.domain*, where *domain* is a hierarchical, dot-separated List of subdomains. Therefore, a host name cannot contain a dot, `"."` as a legal character other than as a subdomain separator. The name must not contain the commercial at sign, `"@"`, as this is often used to separate a file from the host in a remote file name. Also, to prevent confusion with port numbers (see section 2.7.9) a host name cannot contain a colon, `":"`. The maximum length of a host name supported by PBS is defined by `PBS_MAXHOSTNAME`, currently set to 64.

3.1.6 Job Identifiers

When the term job identifier is used, the identifier is specified as:

`sequence_number[.server_name][@server]` The `sequence_number` is the number supplied by the Server when the job was submitted. The `server_name` component is the name of the Server which created the job. If it is missing, the name of the default Server will be assumed. `@server` specifies the current location of the job. When the term fully qualified job identifier is used, the identifier is specified as:

`sequence_number.server[@server]`

The `@server` suffix is not required if the job is still resides at the original Server which created the job. The `qsub` command will return a fully qualified job identifier.

3.1.7 Job Name

A Job Name is a string assigned by the user to provide a meaningful label to identify the job. The job name is up to and including 15 characters in length and may contain any printable characters other than white space. It must start with an alphanumeric character. If the user does not assign a name, PBS will assign a default name as described under the `-N` option of the `qsub(1)` command.

3.1.8 Resource Name

A Resource Name identifies a job resource requirement and may also identify a resource usage limit. The name must consist of alphanumeric characters plus the underscore, “_”, character. It should start with an alphanumeric character. The length is not limited. Certain resource names are identified and reserved by POSIX 1003.2d and by PBS. They are listed below in section “Types of Resources”.

3.1.9 Server Name.

Server Name is an ASCII character string of the form: `basic_server_name[:port]` The string identifies a batch Server. Basic Server names are identical to host names. The network routine `gethostbyname` will be used to translate to a network address. The network routine `getservbyname` will be used to determine the port number. An alternate port number may be specified by appending a colon, “:”, and the port number to the host name. This provides the means of specifying an alternate (test) Server on a host

3.1.10 User Name

A User Name is a string which identifies a user on the system under PBS. It is also known as the login name. PBS will accept names up to and including 16 characters. The name may contain any printable, non white space character excluding the commercial at sign, “@”. The various systems on which PBS is executing may place additional limitations on the user name.

3.2 Batch Server Functions

A batch Server provides services in one of two ways, (1) the Server provides a service at the request of a client; or (2) the Server provides a deferred service as a result of a change in conditions monitored by the Server. The Server also performs a number of internal bookkeeping functions that are described in this major section.

3.2.1 Client Service Requests

By definition, clients are processes that make requests of a batch Server. The requests may ask for an action to be performed on one or more jobs, one or more queues, or the Server itself. Those requests that cannot be successfully completed, are rejected. The reason for the rejection is returned in the reply to the client.

3.2.2 Deferred Services

The Server may, depending on conditions being monitored, defer a client service request until a later time. (Deferred services include file staging, job scheduling, etc.) Detailed discussion of the deferred services provided by the Server is given in [section 3.7, “Deferred Services”, on page 22](#) below.

3.3 Server Management

The following sections describe the services provided by a batch Server in response to a request from a client. The requests are grouped in the following subsections by the type of object affected by the request: Server, queue, job, or resource. The batch requests described in

this section control the functioning of the batch Server. The control is either direct as in the Shut Down request, or indirect as when Server attributes are modified. The following table provides the numeric value of each of the batch request codes.

Table 3-1: Batch Request Codes

0	PBS_BATCH_Connect	24	PBS_BATCH_Reseq
1	PBS_BATCH_QueueJob	25	PBS_BATCH_ReserveResc
2	UNUSED	26	PBS_BATCH_ReleaseResc
3	PBS_BATCH_jobscript	27	PBS_BATCH_FailOver
4	PBS_BATCH_RdytoCommit	48	PBS_BATCH_StageIn
5	PBS_BATCH_Commit	49	PBS_BATCH_AuthenUser
6	PBS_BATCH_DeleteJob	50	PBS_BATCH_OrderJob
7	PBS_BATCH_HoldJob	51	PBS_BATCH_SelStat
8	PBS_BATCH_LocateJob	52	PBS_BATCH_RegistDep
9	PBS_BATCH_Manager	54	PBS_BATCH_CopyFiles
10	PBS_BATCH_MessJob	55	PBS_BATCH_DelFiles
11	PBS_BATCH_ModifyJob	56	PBS_BATCH_JobObit
12	PBS_BATCH_MoveJob	57	PBS_BATCH_MvJobFile
13	PBS_BATCH_ReleaseJob	58	PBS_BATCH_StatusNode
14	PBS_BATCH_Rerun	59	PBS_BATCH_Disconnect
15	PBS_BATCH_RunJob	60-61	UNUSED
16	PBS_BATCH_SelectJobs	62	PBS_BATCH_JobCred
17	PBS_BATCH_Shutdown	63	PBS_BATCH_CopyFiles_Cred
18	PBS_BATCH_SignalJob	64	PBS_BATCH_DelFiles_Cred
19	PBS_BATCH_StatusJob	65	PBS_BATCH_GSS_Context
20	PBS_BATCH_StatusQue	66-69	UNUSED
21	PBS_BATCH_StatusSvr	70	PBS_BATCH_SubmitResv

Table 3-1: Batch Request Codes

22	PBS_BATCH_TrackJob	71	PBS_BATCH_StatusResv
23	PBS_BATCH_AsyrunJob	72	PBS_BATCH_DeleteResv

3.3.1 Manage Request

The Manage request supports the `qmgr (8)` command and several of the operator commands. The command directs the Server to create, alter, or delete an object managed by the Server or one of its attributes. For more information, see the `qmgr` command.

3.3.2 Server Status Request

The status of the Server may be requested with a Server Status request. The batch Server will reject the request if the user of the client is not authorized to query the status of the Server. If the request is accepted, the Server will return a Server Status Reply. See the `qstat` command and the Data Exchange Format description for details of which Server attributes are returned to the client.

3.3.3 Start Up

A batch request to start a Server cannot be sent to a Server since the Server is not running. Therefore a batch Server must be started by a process local to the host on which the Server is to run. The Server is started by a `pbs_server` command. The Server recovers the state of managed objects, such as queues and jobs, from the information last recorded by the Server. The treatment of jobs which were in the running state when the Server previously shut down is dictated by the start up mode, see the description of the `pbs_server (8)` command.

3.3.4 Shut Down

The batch Server is "shut down" when it no longer responds to requests from clients and does not perform deferred services. The batch Server is requested to shut down by sending it a Server Shutdown request. The Server will reject the request from a client not authorized to shut down the Server. When the Server accepts a shut down request, it will terminate in the manner described under the `qterm` command. When shutting down, the Server must record the state of all managed objects (jobs, queues, etc.) in non-volatile memory. Jobs which were running will be marked in the secondary state field for possible special treatment when the Server is restarted. If checkpoint is supported, any job running at the time of the shut down request whose Checkpoint attribute is not `n`, will be checkpointed. This includes jobs whose

Checkpoint attribute value is “unspecified”, a value of u. If the Server receives either a SIG-TERM or a SIGSHUTDN signal, the Server will act as if it had received a shut down immediate request.

3.4 Queue Management

The following client requests effect one or more queues managed by the Server. These requests require a privilege level generally assigned to operators and administrators.

3.4.1 Queue Status Request

The status of a queue at the Server may be requested with a Queue Status request. The batch Server will reject the request if any of the following conditions are true:

- The user of the client is not authorized to query the status of the designated queue.
- The designated queue does not exist on the Server.

If the request does not specify a queue, status of all the queues at the Server will be returned. When the request is accepted, the Server will return a Queue Status Reply. See the `qstat` command and the Data Exchange Format description for details of which queue attributes are returned to the client.

3.5 Job Management

The following client requests effect one or more jobs managed by the Server. These requests do not require any special privilege except when the job for which the request is issued is not owned by the user making the request.

3.5.1 Queue Job Request

A Queue Job request is a complex request consisting of several subrequests: Initiate Job Transfer, Job Data, Job Script, and Commit. The end result of a successful Queue Job request is an additional job being managed by the Server. The job may have been created by the request or it may have been moved from another Server. The job resides in a queue managed by the Server. When a queue is not specified in the request, the job is placed in a queue selected by the Server. This queue is known as the default queue. The default queue is an

attribute of the Server that is settable by the administrator. The queue, whether specified or defaulted, is called the target queue. The batch Server will reject a Queue Job Request if any of the following conditions are true:

- The client is not authorized to create a job in the target queue.
- The target queue does not exist at the Server.
- The target queue is not enabled.
- The target queue is an execution queue and a resource requirement of the job exceeds the limits set upon the queue.
- The target queue is an execution queue and an unrecognized resource is requested by the job.
- The job requires access to a user identifier that the client is not authorized to access.

When a job is placed in a execution queue, it is placed in the queued state unless one of the following conditions applies:

- The job has an `execution_time` attribute that specifies a time in the future and the `Hold_Types` attribute has value of `{NONE}`; in which case the job is placed in the waiting state.
- The job has a `Hold_Types` attribute with a value other than `{NONE}`, wherein the job is placed in the held state.

When a job is placed in a routing queue, its state may change based on the conditions described in [section 3.7.4, “Job Routing”, on page 24](#).

A Server that accepts a Queue Job Request for a new job will: (1) add the `PBS_O_QUEUE` variable to the `Variable_List` attribute of the job and set the value to the name of the target queue; (2) add the `PBS_JOBID` variable to the `Variable_List` attribute of the job and set the value to the job identifier assigned to the job; (3) add the `PBS_JOBNAME` variable to the `Variable_List` attribute of the job and set the value to the value of the `Job_Name` attribute of the job. When the Server accepts a Queue Job request for an existing job, the Server will send a Track Job request to the Server which created the job.

3.5.2 Job Credential Request

The Job Credential sub-request is part of the Queue Job complex request. This sub-request transfers a copy of the credential provided by the authentication facility explained below.

3.5.3 Job Script Request

The Job Script sub-request is part of the Queue Job complex request. This sub-request passes a block of the job script file to the receiving Server. The script is broken into 8 kilobyte blocks to prevent having to hold the entire script in memory. One or more Job Script sub-requests may be required to transfer the script file.

3.5.4 Commit Request

The Commit sub-request is part of the Queue Job request. The Commit notifies the receiving Server that all parts of the job have been transferred and the receiving Server should now assume ownership of the job. Prior to sending the Commit, the sending client, command or another Server, is the owner.

3.5.5 Message Job Request

A batch Server can be requested to write a string of characters to one or both output streams of an executing job. This request is primarily used by an operator to record a message for the user. The batch Server will reject a Message Job request if any of the following conditions are true:

- The designated job is not in the running state.
- The user of the client is not authorized to post a message to the designated job.
- The designated job is not owned by the Server.

When the Server accepts the Message Job request, it will forward the request to the primary MOM daemon for the job. (Upon receipt of the Message Job request from the Server, the MOM will append the message string, followed by a new line character, to the file or files indicated. If no file is indicated, the message will be written to the standard error of the job.)

3.5.6 Locate Job Request

A client may ask a Server to respond with the location of a job that was created or is owned by the Server. When the Server accepts the Locate Job request, it returns a Locate Reply. The request will be rejected if any of the following conditions are true:

- The Server does not own (manage) the job, and
- The Server did not create the job.
- The Server is not maintaining a record of the current location of the job.

3.5.7 Delete Job Request

A Delete Job request asks a Server to remove a job from the queue in which it exists and not place it elsewhere. The batch Server will reject a Delete Job Request if any of the following conditions are true:

- The user of the client is not authorized to delete the designated job.
- The designated job is not owned by the Server.
- The designated job is not in an eligible state. Eligible states are queued, held, waiting, running, and transiting.

If the job is in the running state, the Server will forward the Delete Job request to the primary MOM daemon responsible for the job. (Upon receipt, the MOM daemon will first send a SIGTERM signal to the job process group. After a delay specified by the delete request or if not specified, the `kill_delay` queue attribute, the MOM will send a SIGKILL signal to the job process group. The job is then placed into the exiting state.) Option arguments exist to specify the “delay” time (seconds) between the SIGTERM and SIGKILL signals, as well as to “force” the deletion of the job even if the node on it is running is not responding.

3.5.8 Modify Job Request

A batch client makes a Modify Job request to the Server to alter the attributes of a job. The batch Server will reject a Modify Job Request if any of the following conditions are true:

- The user of the client is not authorized to make the requested modification to the job.
- The designated job is not owned by the Server.
- The requested modification is inconsistent with the state of the job.
- A requested resource change would exceed the limits of the queue or Server.
- An unrecognized resource is requested for a job in an execution queue.

When the batch Server accepts a Modify Job Request, it will modify all the specified attributes of the job. When the batch Server rejects a Modify Job Request, it will modify none of the attributes of the job.

3.5.9 Run Job

The "Run Job" request directs the Server to place the specified job into immediate execution. The request is issued by a `qrun` operator command and by the PBS Job Scheduler.

3.5.9.1 Rerun Job Request

To rerun a job is to kill the members of the session (process) group of the job and leave the job in the execution queue. If the `Hold_Types` attribute is not `{NONE}`, the job is eligible to be re-scheduled for execution. The Server will reject the Rerun Job request if any of the following conditions are true:

- The user of the client is not authorized to rerun the designated job.
- The `Rerunnable` attribute of the job has the value `{FALSE}`.
- The job is not in the running state.
- The Server does not own the job.

When the Server accepts the Rerun Job request, the request will be forwarded to the primary MOM responsible for the job, who will then perform the following actions:

- Send a `SIGKILL` signal to the session (process) group of the job.
- Send an `OBIT` notice to the Server with resource usage information
- The Server will then requeue the job in the execution queue in which it was executing.

If the `Hold_Types` attribute is not `{NONE}`, the job will be placed in the held state. If the `execution_time` attribute is a future time, the job will be placed in the waiting state. Otherwise, the job is placed in the queued state.

3.5.10 Hold Job Request

A client can request that one or more holds be applied to a job. The batch Server will reject a Hold Job request if any of the following conditions are true:

- The user of the client is not authorized to add any of the specified holds.
- The batch Server does not manage the specified job.

When the Server accepts the Hold Job Request, it will add each type of hold listed which is not already present to the value of the `Hold_Types` attribute of the job. If the job is in the queued or waiting state, it is placed in the held state. If the job is in running state, then the following additional actions are taken: If check-point / restart is supported by the host system, placing a hold on a running job will cause the job (1) to be checkpointed, (2) the resources assigned to the job will be released, and (3) the job is placed in the held state in the execution queue. If checkpoint / restart is not supported, the Server will only set the requested hold attribute. This will have no effect unless the job is rerun or restarted.

3.5.11 Release Job Request

A client can request that one or more holds be removed from a job. A batch Server rejects a Release Job request if any of the following conditions are true:

- The user of the client is not authorized to add (remove) any of the specified holds.
- The batch Server does not manage the specified job.

When the Server accepts the Release Job Request, it will remove each type of hold listed from the value of the `Hold_Types` attribute of the job. Normally, the job will then be placed in the queued state, unless another hold type is remaining on the job. However, if the job is in the held state and all holds have been removed, the job is placed in the waiting state if the `Execution_Time` attribute specifies a time in the future.

3.5.12 Move Job Request

A client can request a Server to move a job to a new destination. The batch Server will reject a Move Job Request if any of the following conditions are true:

- The user of the client is not authorized to remove the designated job from the queue in which the job resides.
- The user of the client is not authorized to submit a job to the new destination.
- The designated job is not owned by the Server.
- The designated job is not in the queued, held, or waiting state.
- The new destination is disabled.
- The new destination is inaccessible. When the Server accepts a Move Job request, it will
 - Queue the designated job at the new destination.
 - Remove the job from the current queue.

If the destination exists at a different Server, the current Server will transfer the job to the new Server by sending a Queue Job request sequence to the target Server. The Server will insure that a job is neither lost nor duplicated.

3.5.13 Select Jobs Request

A client is able to request from the Server a list of jobs owned by that Server that match a list of selection criteria. The request is a Select Jobs request. All the jobs owned by the Server and which the user is authorized to query are initially eligible for selection. Job attributes and resources relationships listed in the request restrict the selection of jobs. Only jobs which have attributes and resources that meet the specified relations will be selected. The Server will

reject the request if the queue portion of a specified destination does not exist on the Server. When the request is accepted, the Server will return a Select Reply containing a list of zero or more jobs that met the selection criteria.

3.5.14 Signal Job Request

A batch client is able to request that the Server signal the session (process) group of a job. Such a request is called a Signal Job request. The batch Server will reject a Signal Job Request if any of the following conditions are true:

- The user of the client is not authorized to signal the job.
- The job is not in the running state, except for the special signal “resume” when the job must be in the Suspended state.
- The Server does not own the designated job.
- The requested signal is not supported by the host operating system. (The kill system call returns [EINVAL].)

When the Server accepts a request to signal a job, it will forward the request to the primary MOM daemon responsible for the job, who will then send the signal requested by the client to the all processes in the job’s session.

3.5.15 Status Job Request

The status of a job or set of jobs at a destination may be requested with a Status Job request. The batch Server will reject a Status Job Request if any of the following conditions are true:

- The user of the client is not authorized to query the status of the designated job.
- The designated job is not owned by the Server.

When the Server accepts the request, it will return a Job Status Message to the client. See the qstat command and the Data Exchange Format description for details of which job attributes are returned to the client. If the request specifies a job identifier, status will be returned only for that job. If the request specifies a destination identifier, status will be returned for all jobs residing within the specified queue that the user is authorized to query.

3.6 Server to Server Requests

Server to Server requests are a special category of client requests. They are only issued to a Server by another Server.

3.6.1 Track Job Request

A client that wishes to request an action be performed on a job must send a batch request to the Server that currently manages the job. As jobs are routed or moved through the batch network, finding the location of the job can be difficult without a tracking service. The Track Job request forms the basis for this service. A Server that queues a job sends a track job request to the Server which created the job. Additional backup location Servers may be defined. A Server that receives a track job request records the information contained therein. This information is made available in response to a Locate Job request.

3.6.2 Synchronize Job Starts

PBS provides for synchronizing the initiation of separate jobs. This is done to support distributing processing. Job start synchronization is requested through a special dependency attribute. The first job in the set, the “master”, specifies the dependency attribute as:

```
-W synccount=count
```

where `count` is an integer which is the number of other jobs to be synchronized with this job. This job is the master only in the sense that it defines the rendezvous point for the semaphore messages and that it must be submitted first so the identifier is known for the other jobs in the set. The other jobs in the sync set specify the dependency attribute as:

```
-W syncwith=job_identifier
```

where `job_identifier` is the job identifier assigned to the job which contained the sync-count resource, the master job. When the Server queues a job in an execution queue and the job is a member of a sync set, including the “master”, the Server places a system hold on the job. The secondary state is set to indicate the system hold is for sync. The Server managing the non master jobs will register the job with the Server managing the master by sending a Register Dependent request with a "Register" operation. When all jobs have registered, as determined by the count on the master, the Server managing the master job will send a Register Dependent request, with a "Release" operation, request to each job in turn in the set to remove the system hold. The released jobs may now vie for resources. The jobs are released in order of the “cheapest” resources first; the concept of “Resource Costs” will be explained shortly. When the resources required by a released job are available, as determined by the Scheduler, A run Job Request will be issued for that job. The Server which manages the job will send a Register Dependent request with a “Ready” operation to the Server that owns the master job. This request indicates that the dependent job is ready and the job with the next cheapest resources can be released.

If the master of a sync set is aborted before all jobs in the set begin execution, an Abort Job request is sent to all jobs in the set. This is done because the synchronous feature is intended for a set jobs which need communication amongst themselves during execution. If the master is gone, (1) the rendezvous point for Server messages is lost, and (2) the job set is unlikely to be able to establish the inter job communications required.

3.6.3 Job Dependency

PBS provides support for job dependency. A job, the “child”, can be declared to be dependent on one or more jobs, the “parents”. A parent may have any number of children. The dependency is specified as an attribute on the `qsub` command with the `-W` option. The general specification is of the form:

```
-W type=argument[,type=argument,...]
```

See the `qalter(1B)` or `qsub(1B)` man pages for the complete specification of the dependency list, and the **PBS Professional User’s Guide** for detailed discussion of use.

When a Server queues a job with a dependency type of `syncwith`, `after`, `afterok`, `after notok`, or `after-any` in an execution queue, the Server will send a Register Dependent Job request to the Server managing the job specified by the associated `job_identifier`. The request will specify that the Server is to register the dependency. This actually creates a corresponding `before` type dependency attribute entry on the parent (e.g. run job X *before* job Y). If the request is rejected because the parent job does not exist, the child job is aborted. If the request is accepted, a system hold is placed on the child job. When a parent job, with any of the `before...` types of dependency, reaches the required state, started or terminated, the Server executing the parent job sends a Register Dependent Job request to the Server managing the child job directing it to release the child job. If there are no other dependencies on other jobs, the system hold on the child job is removed. When a child job is submitted with an `on` dependency and the parent is submitted with any of the `before...` types of dependencies, the parent will register with the child. This causes the `on` dependency count to be reduced and a corresponding `after...` dependency to be created for the child job. The result is a pairing between corresponding `before...` and `after...` dependency types. If the parent job terminates in a manner that the child is not released, it is up to the user to correct the situation by either deleting the child job or by correcting the problem with the parent job and resubmitting it. If the parent job is resubmitted, it must have a dependency type of `before`, `beforeok`, `beforenotok`, or `beforeany` specified to connect it to the waiting child job.

3.7 Deferred Services

This section describes the deferred services performed by batch Servers: file staging, job selection, job initiation, job routing, job exit, job abort, and the rerunning of jobs after a restart of the Server. The following rules apply to deferred services on behalf of jobs:

- If the Server cannot complete a deferred service for a reason which is permanent, then the job is aborted.
- If the service cannot be completed at the current time but may be later, the service is retried a finite number of times.

3.7.1 Job Scheduling

If the Server attribute `scheduling` is set true, the Server will immediately request a scheduling cycle of the PBS Job Scheduler. While it remains true, the Scheduler will be cycled when any of four events occur:

- Enqueuing of a job in an execution queue or the change of state of a job in an execution queue to Queued from Waiting or Held.
- Termination of a running job. The termination may be normal execution completion, or because the job was deleted by request.
- Elapse of a specified cycle time as established by the administrator.
- The completion of a scheduling cycle in which one and only one job was scheduled for execution. This provides for the implementation of scheduling scripts that must see the impact of the new job on system resources before picking a second job.

While a request for a scheduling cycle is outstanding, the connection to the Scheduler is open, the Server will not make another request of the Scheduler. If the Server attribute `scheduling` is set false, the Server will not contact the scheduler. This condition is indicated by the `server_state` attribute as Idle.

3.7.2 File Staging

Two types of file staging services exist, in-staging before execution and out-staging after execution. These services are requested by an attribute (via the `-w` option) which specifies the files to be staged:

```
-Wstagein=local_file@host:remote_path [,local_file@host:remote_path,...]  
-Wstageout=local_file@host:remote_path [,local_file@host:remote_path,...]
```

A request to stage in a file directs the Server to direct MOM to copy a file from a remote host to the local host. The user must have authority to access the file under the same user name under which the job will be run. The remote file is not modified or destroyed. The file will be available before the job is initiated. If a file cannot be staged in for any reason, any files which were staged-in are deleted and the job is placed into wait state and mail is sent to the job owner.

A request to stage out a file directs the Server to direct MOM to move a file from the local host to a remote host. This service is performed after the job has completed execution and regardless of its exit status. If a file cannot be moved, mail is sent to the job owner. If a file is successfully staged out, the local file is deleted. A version of the BSD 4.4-Lite system utility, `rcp(1)`, will be used to move files over the network. This version of `rcp` has been modified to always return a non-zero exit status on any failure.

3.7.3 Job Initiation

Job initiation is to place a job into execution. The Server may receive a Run Job request from the `qrun` command, or the PBS Job Scheduler. If the request is authenticated, then the Server forwards the Run Job request to the appropriate MOM (as either specified in the Run Job request, or as selected by the Server itself if unspecified).

The receiving MOM daemon will then create a session leader that runs the shell program indicated by the `Shell_Path_List` attribute of the job. The pathname of the script and any script arguments are passed as parameters to the shell. If the path name of the shell is a relative name, the MOM will search its execution path, `$PATH`, for the shell. If the path name of the shell is omitted or is the null string, the MOM uses the login shell for the user under whose name the job is to be run. The MOM will determine the user name under which the job is to be run by the following rules:

1. Select the user identifier from the `User_List` job attribute which has a host name that matches the execution host.
2. Select the user identifier from the `User_List` job attribute which has no associated host name.
3. Use the user name from the `job_owner` attribute of the job.

The MOM will create, in the environment of the session leader of the job, the environment variables named: `PBS_ENVIRONMENT`, the value of which is the string “`PBS_BATCH`”. `PBS_QUEUE` has the value of the name of the execution queue. The MOM will also place in the environment of the session leader of the job, all of the variables and their corresponding values found in the variables attribute of the job. The MOM will place the required limits on the resources for which the host system supports resource limits. If the job had been run before and is now being rerun, the MOM will insure that the standard output and standard error streams of the job are appended to the prior streams, if any. If the MOM and host system

support accounting, the MOM will use the value of the `Account_Name` job attribute as required by the host system. If the MOM and host system support checkpoint, the MOM will set up checkpointing of the job according to the value of the `Checkpoint` job attribute. If checkpoint is supported and the `Checkpoint` attribute requests checkpointing at the minimum interval or a interval less than the minimum interval for the queue, then checkpoint will be set for an interval given by the queue attribute `minimum_interval`. The MOM will set up the standard output stream and the standard error stream of the job according to the following rules:

- The stream will be located in a temporary file in the MOM's `spool` directory.
- If the job attribute `Join_Path` has the value `eo` or the value `oe`, the MOM connects the standard error stream of the job to the same file as the standard output stream.

3.7.4 Job Routing

Job routing is moving a job from a routing queue to one of the destinations associated with the queue. If the `started` queue attribute is `{TRUE}`, the Server will route all eligible jobs which reside in the queue. All jobs in the queued state are eligible. If the queue attribute `route_held_jobs` is `{TRUE}`, jobs in the held state are eligible for routing. If the queue attribute `route_waiting_jobs` is `{TRUE}`, jobs in the waiting state are eligible. The Server will execute the function specified by the queue attribute `route_function` to select a destination for the job. Possible destinations are listed in the queue attribute `route_destinations`. If the destination to which the job is to be routed is at another Server, the current Server will use a Queue Job request sequence to move the job to the new destination. If the Server is unable to route a job to a chosen destination, the Server will select another destination from the list and retry the route. If the Server is unable to route a job to any destination because of a temporary condition, such as being unable to connect with the Server at the destination, the Server will retry the route after a delay specified by the queue attribute `route_retry_time`. The Server will proceed to route other jobs in the queue. The Server will retry the route up to the (queue attribute) `number_retries` times. If the Server is unable to route a job to any destination and all failures are permanent (non-temporary), the Server will abort the job.

3.7.5 Job Exit

When the session leader of a batch job exits, the MOM will perform the following actions in the order listed.

- Place the job in the exiting state.
- “Free” the resources allocated to the job. The actual releasing of resources assigned to the processes of the job is performed by the kernel. PBS will free the resources which it

“reserved” for the job by decrementing the `resources_used` generic data item for the queue and Server.

- Return the standard output and standard error streams of the job to the user. If the `Keep_Files` attribute of the job contains `{KEEP_OUTPUT}`, the Server copies the spooled file holding the standard output stream of the job to the home directory of the user under whose name the job executed. The file name for the output is `job_name.oseq_number`. See the `qsub(1B)` command description. If the `Keep_Files` attribute of the job contains `{KEEP_ERROR}` and the `Join_Path` attribute does not contain `'e'`, the Server copies the spooled file holding the standard error stream of the job to the home directory of the user under whose name the job executed. The file name for the error file is `job_name.eseq_number`.

If the files are not to be kept on the execution host as described above, the temporary file holding the standard output is copied or renamed to the host and path name specified by the job attribute `Output_Path`. If the path name is relative, the file will be located relative to home directory of the user on the receiving host.

- If the `Join_Path` attribute does not contain the value `e`, the standard error of the job is delivered according to the same rules as the standard output described above. If either output file cannot be copied to its specified destination, the Server will send mail to the job owner specifying the current location of the output.
- If the `Mail_Points` job attribute contains the value `{EXIT}`, the Server will send mail to the users listed in the job attribute `Mail_List`.
- If out staging of files is supported, the files listed in the outfile resource will be copied to the specified destination.
- The job will be removed from the execution queue.

3.7.6 Job Aborts

If the Server aborts a job and the `Mail_Points` job attribute contains the value `{ABORT}`, the Server will send mail to the users listed in the job attribute `Mail_List`. The mail message will contain the reason the job was aborted. In addition, the `stdout` and `stderr` files specified for the job, if they exist, will be copied back to the specified location.

3.7.7 Timed Events

The Server performs certain events at a specified time or after a specified time delay. A job may have an `execution_time` attribute set to a time in the future. When that time is reached, the job state is updated. If the Server is unable to make connection with another Server, it is to retry after a time specified by the routing queue attribute `route_retry_time`.

3.7.8 Event Logging

The PBS Server maintains an event logfile, the format and contents of which are documented in the **PBS Professional Administrator's Guide**.

3.7.9 Accounting.

The PBS Server maintains an accounting file, the format and contents of which are documented in the **PBS Professional Administrator's Guide**.

3.8 Resource Management

PBS performs resource allocation at job initiation in two ways depending on the support provided by the host system. Resources are either reservable or non reservable.

3.8.1 Resource Limits

When submitting a job, a user may specify the hard limit of usage for resources known to the system on which the job will run. If the executing job usage of resources exceed the specified limit, the job is aborted. If the user does not specify a limit for a resource type, the limit may be set to a default established by the PBS administrator. The default limit is taken from the first of the following attributes which is set:

1. The current queue's attribute `resources_default`.
2. The Server's attribute `resources_default`.
3. The current queue's attribute `resources_max`.
4. The Server's attribute `resources_max`.

If the user does not specify a limit for a resource and a default is not established via one of the above attributes, the usage of the resource is unlimited.

3.8.2 Resource Names

For additional information, see the **PBS Professional User's Guide** where all resource names are documented.

3.9 Network Protocol

The PBS system fits into a client - Server model, with a batch client making a request of a batch Server and the Server replying. This client - Server communication necessitates an interprocess communication method and a data exchange (data encoding) format. Since the client and Server may reside on different systems, the interprocess communication must be supportable over a network.

While the basic PBS system fits nicely into the client - Server model, it also has aspects of a transaction system. When jobs are being moved between Servers, it is critical that the jobs are not lost or replicated. Updates to a batch job must be applied once and only once. Thus the operation must be atomic. Most of the client to Server requests consist of a single message. Treating these requests as an atomic operation is simple. One request, "Queue Job", is more complex and involves several messages, or subrequests, between the client and the Server. Any of these subrequests might be rejected by the Server. It is important that either side of the connection be able to abort the request (transaction) without losing or replicating the job. The network connection also might be lost during the request. Recovery from a partially transmitted request sequence is critical. The sequence of recovery from lost connections is discussed in the Queue Job Request description.

The batch system data exchange protocol must be built on top of a reliable stream connection protocol. PBS uses TCP/IP and the socket interface to the network. Either the Simple Network Interface, SNI, or the Detailed Network Interface, DNI, as specified by POSIX.12, Protocol Independent Interfaces, could be used as a replacement.

3.9.1 General DIS Data Encoding

The purpose of the "Data is Strings" encoding is to provide a simple, fast, small, machine independent form for encoding data to a character string and back again. Because data can be decoded directly into the final internal data structures, the number of data copy operations are reduced. Data items are represented as people think of them, but preceded with a count of the length of each data item. For small positive integers, it is impossible to tell from the encoded data whether they came from `signed chars`, `shorts`, `ints`, or `longs`. Similarly, for small negative numbers, the only thing that can be determined from the encoded data is that the source datum was not unsigned. It is impossible to tell the word size of the encoding machine, or whether it uses 2's complement, one's complement or sign - magnitude representation, or even if it uses binary arithmetic. All of the basic C data types are handled. Signed and unsigned chars, shorts, ints, longs produce integers. NULL terminated and counted strings produce counted strings (with the terminating NULL removed). Floats, doubles, and long doubles produce real numbers. Complex data must be built up from the basic types. Note that there is no type tagging, so the type and sequence of data to be decoded must be known in advance.

Chapter 4

Batch Interface Library (IFL)

The primary external application programming interface to PBS is the Batch Interface Library, or IFL. This library provides a means of building new batch clients. Any batch service request can be invoked through calls to the batch interface library. Users may wish to build a job which could status itself or spawn off new jobs. Or they may wish to customize the job status display rather than use `qstat`. Administrators may use the interface library to build new control commands.

4.1 Interface Library Overview

The IFL provides a user-callable function corresponding to each batch client command. There is (approximately) a one to one correlation between commands and batch service requests. Additional routines are provided for network connection management. The user callable routines are declared in the header file `PBS_ifl.h`. Users open a connection with a batch Server via a call to `pbs_connect()`. Multiple connections are supported. Before a connection is established, `pbs_connect()` will fork and exec an `pbs_iff` process, as shown in figure 4-1 below. The purpose of `pbs_iff` is to provide the user a credential which validates the user's identity. This credential is included in each batch request. The provided credential prevents a user from spoofing another user's identity.

The credential that is sent to the server consists of: a) user's name from the password file based on running `pbs_iff`'s "real uid" value, and b) unprivileged, client-side port value associated with the original `pbs_connect` request message to the server. The server looks at the entries in its connection table to try and find the entry having these two pieces of information, and which is not yet marked authenticated. To be believed, this information must be gotten from a connection having a privileged, remote-end, port value.

After all requests have been made to a Server, its connection is closed via a call to `pbs_disconnect()`.

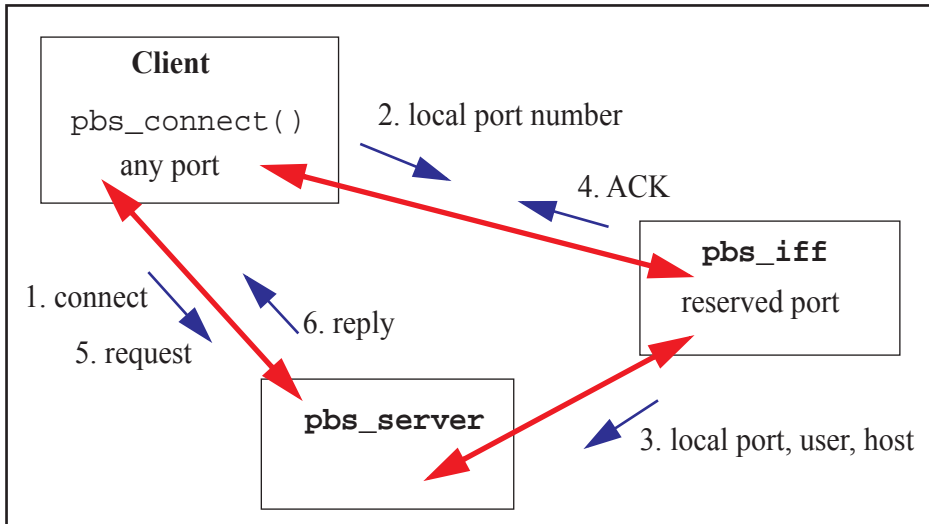


Figure 4-1: Interface Between Client, IFF, and Server

Users request service of a batch Server by calling the appropriate library routine and passing it the required parameters. The parameters correspond to the options and operands on the commands. It is the user's responsibility to ensure the parameters have correct syntax. Each function will return zero upon success and a non-zero error code on failure. These error codes are available in the header file `PBS_error.h`. The library routine will accept the parameters and build the corresponding batch request, then pass it to the Server.

To use `pbs_connect` with Windows, initialize the network library and link with `winsock2`. Call `winsock_init()` before calling `pbs_connect()`, and link against the `ws2_32.lib` library.

Any user-written programs using the IFL API must link with the `pthread` library.

4.2 Interface Library Routines

The following manual pages describe the user-callable functions in the IFL.

pbs_alterjob

alter pbs batch job

SYNOPSIS

```
#include <pbs_error.h>
#include <pbs_ifl.h>
```

```
int pbs_alterjob(int connect, char *job_id, struct attrl *attrib,
char *extend)
```

DESCRIPTION

Issue a batch request to alter a batch job.

A Modify Job batch request is generated and sent to the server over the connection specified by connect which is the return value of pbs_connect().

The argument, job_id , identifies which job is to be altered. It is specified in the form:

sequence_number.server

The parameter, attrib , is a pointer to an attrl structure which is defined in pbs_ifl.h as:

```
struct attrl {
    char *name;
    char *resource;
    char *value
    struct attrl *next;
};
```

The attrib list is terminated by the first entry where next is a null pointer.

The name member points to a string which is the name of the attribute. The value member points to a string which is the value of the attribute. The attribute names are defined in pbs_ifl.h.

If attrib itself is a null pointer, then no attributes are altered.

Associated with an attribute of type `ATTR_1` (the letter `ell`) is a resource name indicated by `resource` in the `attr1` structure. All other attribute types should have a pointer to a null string (“”) for `resource`.

If the resource of the specified resource name is already present in the job’s `Resource_List` attribute, it will be altered to the specified value. If the resource is not present in the attribute, it is added.

Certain attributes of a job may or may not be alterable depending on the state of the job; see `qalter(1B)`.

The parameter, `extend`, is reserved for implementation defined extensions.

SEE ALSO

`qalter(1B)`, `qhold(1B)`, `qrls(1B)`, `qsub(1B)`, `pbs_connect(3B)`, `pbs_holdjob(3B)`, and `pbs_rlsjob(3B)`

DIAGNOSTICS

When the batch request generated by `pbs_alterjob()` function has been completed successfully by a batch server, the routine will return 0 (zero). Otherwise, a non zero error is returned. The error number is also set in `pbs_errno`.

pbs_connect

connect to a PBS batch server

SYNOPSIS

```
#include <pbs_error.h>
#include <pbs_ifl.h>

int pbs_connect(char *server)
extern char *pbs_server;
```

DESCRIPTION

A virtual stream (TCP/IP) connection is established with the server specified by server.

This function must be called before any of the other pbs_ functions. They will transmit their batch requests over the connection established by this function. Multiple requests may be issued over the connection before it is closed.

The connection should be closed by a call to pbs_disconnect() when all requests have been sent to the server.

The parameter called server is of the form

host_name[:port].

If port is not specified, the standard PBS port number will be used.

If the parameter, server, is either the null string or a null pointer, a connection will be opened to the default server. The default server is defined by (a) the setting of the environment variable PBS_DEFAULT which contains a destination, or (b) by adding the parameter PBS_SERVER to the global configuration file /etc/pbs.conf.

The variable pbs_server, declared in pbs_ifl.h, is set on return to point to the server name to which pbs_connect() connected or attempted to connect.

pbs_connect() determines whether or not the complex has a failover server configured. It also determines which server is the primary and which is the secondary. pbs_connect() is called by client commands, and directs traffic to the correct server.

In order to use `pbs_connect` with Windows, initialize the network library and link with `winsock2`. To do this, call `winsock_init()` before calling `pbs_connect()`, and link against the `ws2_32.lib` library.

SEE ALSO

`qsub(1B)`, `pbs_alterjob(3B)`, `pbs_deljob(3B)`, `pbs_disconnect(3B)`,
`pbs_geterrmsg(3B)`, `pbs_holdjob(3B)`, `pbs_locate(3B)`,
`pbs_manager(3B)`, `pbs_movejob(3B)`, `pbs_msgjob(3B)`,
`pbs_rerunjob(3B)`, `pbs_rlsjob(3B)`, `pbs_runjob(3B)`,
`pbs_selectjob(3B)`, `pbs_selstat(3B)`, `pbs_sigjob(3B)`,
`pbs_statjob(3B)`, `pbs_statque(3B)`, `pbs_statserver(3B)`,
`pbs_submit(3B)`, `pbs_terminate(3B)`, `pbs_server(8B)`, and the PBS
Professional Programmer's Guide

DIAGNOSTICS

When the connection to batch server has been successfully created, the routine will return a connection identifier which is positive. Otherwise, a negative value is returned. The error number is set in `pbs_errno`.

pbs_default

return the name of the default PBS server

SYNOPSIS

```
#include <pbs_error.h>
#include <pbs_ifl.h>

char *pbs_default()
```

DESCRIPTION

A character string is returned containing the name of the default PBS server. The default server is defined by (a) the setting of the environment variable `PBS_DEFAULT` which contains a destination, or (b) by adding the parameter `PBS_SERVER` to the global configuration file `/etc/pbs.conf`.

DIAGNOSTICS

If the default server cannot be determined, a NULL value is returned.

SEE ALSO

`qsub(1B)`, `pbs_connect(3B)`, `pbs_disconnect(3B)`, and the PBS Professional Programmer's Guide

pbs_deljob

delete a PBS batch job

SYNOPSIS

```
#include <pbs_error.h>
#include <pbs_ifl.h>
```

```
int pbs_deljob(int connect, char *job_id, char *extend)
```

DESCRIPTION

Issue a batch request to delete a batch job. If the batch job is running, the execution server will send the SIGTERM signal followed by SIGKILL.

A Delete Job batch request is generated and sent to the server over the connection specified by connect which is the return value of pbs_connect().

The argument, job_id, identifies which job is to be deleted. It is specified in the form:

“sequence_number.server”

The argument, extend, is overloaded to serve more than one purpose. If extend points to a string other than the above, it is taken as text to be appended to the message mailed to the job owner. This mailing occurs if the job is deleted by a user other than the job owner.

SEE ALSO

qdel(1B) and pbs_connect(3B)

DIAGNOSTICS

When the batch request generated by the pbs_deljob() function has been completed successfully by a batch server, the routine will return 0 (zero). Otherwise, a non zero error is returned. The error number is also set in pbs_errno.

pbs_delresv

delete a reservation

SYNOPSIS

```
#include <pbs_error.h>
#include <pbs_ifl.h>
```

```
int pbs_delresv(int connect, char *resv_id, char *extend)
```

DESCRIPTION

Issue a batch request to delete a reservation. If the reservation is in state RESV_RUNNING, and there are jobs remaining in the reservation queue, the jobs will be deleted before the reservation is deleted.

A Delete Reservation batch request is generated and sent to the server over the connection specified by connect which is the return value of pbs_connect().

The argument, resv_id, identifies which reservation is to be deleted, it is specified in the form:

```
“R<sequence_number>.<server>”
```

The argument, extend is currently unused.

SEE ALSO

pbs_rdel(1B) and pbs_connect(3B)

DIAGNOSTICS

When the batch request generated by the pbs_delresv() function has been completed successfully by a batch server, the routine will return 0 (zero). Otherwise, a non zero error is returned. The error number is also set in pbs_errno.

pbs_disconnect

disconnect from a pbs batch server

SYNOPSIS

```
#include <pbs_error.h>
#include <pbs_ifl.h>

int pbs_disconnect(int connect)
```

DESCRIPTION

The virtual stream connection specified by connect , which was established with a server by a call to pbs_connect(), is closed.

SEE ALSO

pbs_connect(3B)

DIAGNOSTICS

When the connection to batch server has been successfully closed, the routine will return zero. Otherwise, a non zero error is returned. The error number is also set in pbs_errno.

pbs_geterrmsg

get error message for last pbs batch operation

SYNOPSIS

```
#include <pbs_error.h>
#include <pbs_ifl.h>
```

```
char *pbs_geterrmsg(int connect)
```

DESCRIPTION

Return the error message text associated with a batch server request.

If the preceding batch interface library call over the connection specified by connect resulted in an error return from the server, there may be an associated text message. If it exists, this function will return a pointer to the null terminated text string.

SEE ALSO

pbs_connect(3B)

DIAGNOSTICS

If an error text message was returned by a server in reply to the previous call to a batch interface library function, pbs_geterrmsg() will return a pointer to it. Otherwise, pbs_geterrmsg() returns the null pointer.

pbs_holdjob

place a hold on a pbs batch job

SYNOPSIS

```
#include <pbs_error.h>
#include <pbs_ifl.h>
```

```
int pbs_holdjob(int connect, char *job_id, char *hold_type,
char *extend)
```

DESCRIPTION

Issue a batch request to place a hold upon a job.

A Hold Job batch request is generated and sent to the server over the connection specified by connect which is the return value of pbs_connect().

The argument, job_id , identifies which job is to be held, it is specified in the form:

“sequence_number.server”

The parameter, hold_type , contains the type of hold to be applied. The possible values are defined in pbs_ifl.h.

If hold_type is either a null pointer or points to a null string, USER_HOLD will be applied.

The parameter, extend , is reserved for implementation defined extensions.

SEE ALSO

qhold(1B), pbs_connect(3B), pbs_alterjob(3B), and pbs_rlsjob(3B)

DIAGNOSTICS

When the batch request generated by pbs_holdjob () function has been completed successfully by a batch server, the routine will return 0 (zero). Otherwise, a non zero error is returned. The error number is also set in pbs_errno.

pbs_locjob

locate current location of a pbs batch job

SYNOPSIS

```
#include <pbs_error.h>
#include <pbs_ifl.h>
```

```
char *pbs_locjob(int connect, char *job_id, char *extend)
```

DESCRIPTION

Issue a batch request to locate a batch job. If the server currently manages the batch job, or knows which server does currently manage the job, it will reply with the location of the job.

A Locate Job batch request is generated and sent to the server over the connection specified by connect which is the return value of pbs_connect().

The argument, job_id , identifies which job is to be located, it is specified in the form:

“sequence_number.server”

The argument, extend , is reserved for implementation defined extensions. It is not currently used by this function.

The return value is a pointer to a character sting which contains the current location if known. The syntax of the location string is:

“server_name” .

If the location of the job is not known, the return value is the NULL pointer.

SEE ALSO

qsub(1B) and pbs_connect(3B)

DIAGNOSTICS

When the batch request generated by the pbs_locjob() function has been completed successfully by a batch server, the routine will return a non null pointer to the destination. Otherwise, a null pointer is returned. The error number is set in pbs_erno.

pbs_manager

modifies a PBS batch object

SYNOPSIS

```
#include <pbs_error.h>
#include <pbs_ifl.h>
```

```
int pbs_manager(int connect, int command, int obj_type, char *obj_name,
struct attrpl *attrib, char *extend)
```

DESCRIPTION

Issue a batch request to perform administration functions at a server. With this request, server objects such as queues can be created and deleted, and have their attributes set and unset.

A Manage batch request is generated and sent to the server over the connection specified by connect which is the return value of pbs_connect(). This request requires full batch administrator privilege.

The parameter, command, specifies the operation to be performed. See pbs_ifl.h:

```
MGR_CMD_CREATE  creates the object
MGR_CMD_DELETE  deletes the object
MGR_CMD_SET      sets the value
MGR_CMD_UNSET    unsets the value
MGR_CMD_IMPORT   imports the hook
MGR_CMD_EXPORT   exports the hook
```

The parameter, obj_type, declares the type of object upon which the command operates. See pbs_ifl.h:

```
MGR_OBJ_SERVER  Server object
MGR_OBJ_QUEUE    Queue object
MGR_OBJ_NODE     Node object
MGR_OBJ_HOOK     Hook object
```

The parameter, obj_name, is the name of the specific object.

The parameter, attrib, is a pointer to an attrpl structure which is defined in pbs_ifl.h as:

```
struct attropl {  
    char *name;  
    char *resource;  
    char *value;  
    enum batch_op op;  
    struct attropl *next;  
};
```

The attrib list is terminated by the first entry where next is a null pointer.

The name member points to a string which is the name of the attribute.

If the attribute is one which contains a set of resources, the specific resource is specified in the structure member resource. Otherwise, the member resource is pointer to a null string.

The value member points to a string which is the new value of the attribute. For parameterized limit attributes, this string contains all parameters for the attribute.

The op member defines the manner in which the new value is assigned to the attribute. The operators are:

“enum batch_op { ..., SET, UNSET, INCR, DECR }.”

For MGR_CMD_IMPORT, specify attropl “name” as “content-type”, “content-encoding”, and “input-file” along with the corresponding “value” and an “op” of SET.

For MGR_CMD_EXPORT, specify attropl “name” as “content-type”, “content-encoding”, and “output-file” along with the corresponding “value” and an “op” of SET.

The parameter extend is reserved for implementation-defined extensions.

Privilege required for functions depends on whether those functions are used with hooks.

When not used with hooks:

Functions MGR_CMD_CREATE and MGR_CMD_DELETE require PBS Manager privilege.

Functions `MGR_CMD_SET` and `MGR_CMD_UNSET` require PBS Manager or Operator privilege.

When used with hooks:

All commands require root privilege on the server host.

Functions `MGR_CMD_IMPORT`, `MGR_CMD_EXPORT`, and `MGR_OBJ_HOOK` are used only with hooks, and therefore require root privilege on the server host.

DIAGNOSTICS

When the batch request generated by `pbs_manager()` function has been completed successfully by a batch server, the routine will return 0 (zero). Otherwise, a non zero error is returned. The error number is also set in `pbs_errno`.

SEE ALSO

The PBS Professional Programmer's Guide,
`qmgr(1B)`, `pbs_connect(3B)`

pbs_movejob

move a pbs batch job to a new destination

SYNOPSIS

```
#include <pbs_error.h>
#include <pbs_ifl.h>
```

```
int pbs_movejob(int connect, char *job_id, char *destination, char *extend)
```

DESCRIPTION

Issue a batch request to move a job to a new destination. The job is removed from the present queue and instantiated in a new queue.

A Move Job batch request is generated and sent to the server over the connection specified by connect which is the return value of pbs_connect().

The job_id parameter identifies which job is to be moved; it is specified in the form: "sequence_number.server"

The destination parameter specifies the new destination for the job. It is specified as: [queue][@server].

If destination is a null pointer or a null string, the destination will be the default queue at the current server. If destination specifies a queue but not a server, the destination will be the named queue at the current server. If destination specifies a server but not a queue, the destination will be the default queue at the named server. If destination specifies both a queue and a server, the destination is that queue at that server.

A job in the Running, Transiting, or Exiting state cannot be moved.

The parameter, extend, is reserved for implementation defined extensions.

SEE ALSO

qmove(1B), qsub(1B), and pbs_connect(3B)

DIAGNOSTICS

When the batch request generated by pbs_movejob() function has been completed successfully by a batch server, the routine will return 0 (zero). Otherwise, a non zero error is returned. The error number is also set in pbs_errno.

pbs_msgjob

record a message for a running pbs batch job

SYNOPSIS

```
#include <pbs_error.h>
#include <pbs_ifl.h>
```

```
int pbs_msgjob(int connect, char *job_id, int file, char *message,
char *extend)
```

DESCRIPTION

Issue a batch request to write a message in an output file of a batch job.

A Message Job batch request is generated and sent to the server over the connection specified by `connect` which is the return value of `pbs_connect()`.

The argument, `job_id`, identifies the job to which the message is to be sent; it is specified in the form:
“sequence_number.server”

The parameter, `file`, indicates the file or files to which the message string is to be written. See `pbs_ifl.h` for acceptable values.

The parameter, `message`, is the message string to be written.

The parameter, `extend`, is reserved for implementation defined extensions.

SEE ALSO

`qmsg(1B)` and `pbs_connect(3B)`

DIAGNOSTICS

When the batch request generated by `pbs_msgjob()` function has been completed successfully by a batch server, the routine will return 0 (zero). Otherwise, a non zero error is returned. The error number is also set in `pbs_errno`.

pbs_orderjob

reorder pbs batch jobs in a queue

SYNOPSIS

```
#include <pbs_error.h>
#include <pbs_ifl.h>
```

```
int pbs_orderjob(int connect, char *job_id1, char *job_id2,
char *extend)
```

DESCRIPTION

Issue a batch request to swap the order of two jobs with in a single queue.

An Order Job batch request is generated and sent to the server over the connection specified by connect which is the return value of pbs_connect().

The parameters job_id1 and job_id2 identify which jobs are to be swapped. They are specified in the form:
"sequence_number.server".

The parameter, extend, is reserved for implementation defined extensions.

SEE ALSO

qorder(1B), qmove(1B), qsub(1M), and pbs_connect(3B)

DIAGNOSTICS

When the batch request generated by pbs_orderjob() function has been completed successfully by a batch server, the routine will return 0 (zero). Otherwise, a non zero error is returned. The error number is also set in pbs_errno.

pbs_rerunjob

rerun a pbs batch job

SYNOPSIS

```
#include <pbs_error.h>
#include <pbs_ifl.h>
```

```
int pbs_rerunjob(int connect, char *job_id, char *extend)
```

DESCRIPTION

Issue a batch request to rerun a batch job.

A Rerun Job batch request is generated and sent to the server over the connection specified by connect which is the return value of pbs_connect().

If the job is marked as being not rerunnable, the request will fail and an error will be returned.

The argument, job_id , identifies which job is to be rerun it is specified in the form:

“sequence_number.server”

The parameter, extend , is reserved for implementation defined extensions.

SEE ALSO

qrerun(1B), qsub(1B), and pbs_connect(3B)

DIAGNOSTICS

When the batch request generated by pbs_rerunjob() function has been completed successfully by a batch server, the routine will return 0 (zero). Otherwise, a non zero error is returned. The error number is also set in pbs_errno.

pbs_resreserve, pbs_resrelease

reserve/free batch resources

SYNOPSIS

```
#include <pbs_error.h>
#include <pbs_ifl.h>
```

```
int pbs_resreserve(int connect, char **resourcelist, int arraysize,
resource_t *resource_id)
```

```
int pbs_resrelease(int connect, resource_t resource_id)
```

DESCRIPTION**pbs_resreserve**

Issue a request to the batch server to reserve specified resources. connect is the connection returned by pbs_connect(). resourcelist is an array of one or more strings specifying the resources to be queried. arraysize is the number of strings in resourcelist. resource_id is a pointer to a resource handle. The pointer cannot be null. If the present value of the resource handle is RESOURCE_T_NULL, this request is for a new reservation and if successful, a resource handle will be returned in resource_id.

If the value of resource_id as supplied by the caller is not RESOURCE_T_NULL, this is an existing (partial) reservation. Resources currently reserved for this handle will be released and the full reservation will be attempted again. If the caller wishes to release the resources allocated to a partial reservation, the caller should pass the resource handle to pbs_resrelease().

At the present time the only resources which may be specified are "nodes". It should be specified as

nodes=specification

where specification is what a user specifies in the -l option argument list for nodes, see qsub (1B).

pbs_resrelease

The pbs_resrelease() call releases or frees resources reserved with the resource handle of resource_id returned from a prior pbs_resreserve() call. connect is the connection returned by pbs_connect().

Both functions require that the issuing user have operator or administrator privilege.

SEE ALSO

qsub(1B), pbs_connect(3B), pbs_disconnect(3B) and pbs_resources(7B)

DIAGNOSTICS

pbs_resreserve() and pbs_rescrelease() return zero on success. Otherwise, a non zero error is returned. The error number is also set in pbs_erno.

PBSE_RMPART

is a special case indicating that some but not all of the requested resources could be reserved; a partial reservation was made. The reservation request should either be rerequested with the returned handle or the partial resources released.

PBSE_RMBADPARAM

a parameter is incorrect, such as a null for the pointer to the resource_id.

PBSE_RMNOPARAM

a parameter is missing, such as a null resource list.

pbs_rlsjob

release a hold on a pbs batch job

SYNOPSIS

```
#include <pbs_error.h>
#include <pbs_ifl.h>
```

```
int pbs_rlsjob(int connect, char *job_id, char *hold_type, char *extend)
```

DESCRIPTION

Issue a batch request to release a hold from a job.

A Release Job batch request is generated and sent to the server over the connection specified by `connect` which is the return value of `pbs_connect()`.

The argument, `job_id`, identifies the job from which the hold is to be released, it is specified in the form:

“sequence_number.server”

The parameter, `hold_type`, contains the type of hold to be released. The possible values are defined in `pbs_ifl.h`.

If `hold_type` is either a null pointer or points to a null string, `USER_HOLD` will be released.

The parameter, `extend`, is reserved for implementation defined extensions.

SEE ALSO

`qrls(1B)`, `qhold(1B)`, `qalter(1B)`, `pbs_alterjob(3B)`, `pbs_connect(3B)`, and `pbs_holdjob(3B)`

DIAGNOSTICS

When the batch request generated by `pbs_rlsjob()` function has been completed successfully by a batch server, the routine will return 0 (zero). Otherwise, a non zero error is returned. The error number is also set in `pbs_errno`.

pbs_runjob, pbs_asyruncjob

run a PBS batch job, asynchronous batch job

SYNOPSIS

```
#include <pbs_error.h>
#include <pbs_ifl.h>
```

```
int pbs_runjob(int connect, char *job_id, char *location, char *extend)
```

```
int pbs_asyruncjob(int connect, char *job_id, char *location, char *extend)
```

DESCRIPTION

Issue a batch request to run a batch job.

For `pbs_runjob()` a "Run Job" batch request is generated and sent to the server over the connection specified by `connect` which is the return value of `pbs_connect()`. The server will reply when the job has started execution unless file in-staging is required. In that case, the server will reply when the staging operations are started.

For `pbs_asyruncjob()` an "Asynchronous Run Job" request is generated and sent to the server over the connection. The server will validate the request and reply before initiating the execution of the job. This version of the call can be used to reduce latency in scheduling, especially when the scheduler must start a large number of jobs.

These requests requires that the issuing user have operator or administrator privilege.

The argument, `job_id`, identifies which job is to be run it is specified in the form:

sequence_number.server

The argument, `location`, if not the null pointer or null string, specifies the location where the job should be run, and optionally the resources to use. The location is the same as the `-H` option to the `qrun` command. See the description of `qrun -H`, both with and without resources specified, in the `qrun.8B` man page.

The argument, `extend`, is reserved for implementation-defined extensions.

SEE ALSO

`qrun(8B)`, `qsub(1B)`, and `pbs_connect(3B)`

DIAGNOSTICS

When the batch request generated by the `pbs_runjob()` or `pbs_asyruntimejob()` functions has been completed successfully by a batch server, the routines will return 0 (zero). Otherwise, a non zero error is returned. The error number is also set in `pbs_errno`.

pbs_selectjob

select pbs batch jobs

SYNOPSIS

```
#include <pbs_error.h>
#include <pbs_ifl.h>
```

```
char **pbs_selectjob(int connect, struct attrpl *attrib, char *extend)
```

DESCRIPTION

Issue a batch request to select jobs which meet certain criteria.

pbs_selectjob() returns an array of job identifiers which met the criteria.

The attrpl struct contains the list of selection criteria.

Initially all batch jobs are selected for which the user is authorized to query status. This set may be reduced or filtered by specifying certain attributes of the jobs.

A Select Jobs batch request is generated and sent to the server over the connection specified by connect which is the return value of pbs_connect().

The argument, attrib, is a pointer to an attrpl structure which is defined in pbs_ifl.h as:

```
struct attrpl {
    struct attrpl *next;
    char *name;
    char *resource;
    char *value;
    enum batch_op op;
};
```

The attrib list is terminated by the first entry where next is a null pointer.

The name member points to a string which is the name of the attribute. Not all of the job attributes may be used as a selection criteria. The resource member points to a string which is the name of a resource. This member is only used when name is set to ATTR_1. Otherwise, resource should be a pointer to a null string. The value member points to a string which is the value of the attribute or resource. The attribute names are listed in pbs_job_attributes.7B.

The op member defines the operator in the logical expression:

```
value operator current_value
```

The logical expression must evaluate as true for the job to be selected. The permissible values of `op` are defined in `pbs_ifl.h` as:

“enum batch_op { ..., EQ, NE, GE, GT, LE, LT, ... } ;”.

The attributes marked with (E) in the description above may only be selected with the equal, EQ, or not equal, NE, operators.

If `attrib` itself is a null pointer, then no selection is done on the basis of attributes.

The return value is a pointer to a null terminated array of character pointers. Each character pointer in the array points to a character string which is a job_identifier in the form:

sequence_number.server@server

The array is allocated by `pbs_selectjob` via `malloc()`. When the array is no longer needed, the user is responsible for freeing it by a call to `free()`.

The parameter, `extend`, is reserved for implementation defined extensions.

Finished and Moved Jobs

In order to get information on finished and moved jobs, you must add an ‘x’ character to the `extend` parameter. The `extend` parameter is a character string; set one character to be the ‘x’ character. For example:

`pbs_selectjob (..., ..., extend) ...`

To get information on finished and moved jobs only, specify the Finished (‘F’) and moved (‘M’) job states. You must also use the `extend` character string containing the ‘x’ character.

Subjobs are not considered finished until the parent array job is finished.

SEE ALSO

`qselect(1B)`, `pbs_alterjob(3B)`, and `pbs_connect(3B)`

DIAGNOSTICS

When the batch request generated by `pbs_selectjob()` function has been completed successfully by a batch server, the routine will return a pointer to the array of job identifiers. If no jobs met the criteria, the first pointer in the array will be the null pointer.

If an error occurred, a null pointer is returned and the error is available in the global integer `pbs_erno`.

pbs_selstat

obtain status of selected pbs batch jobs

SYNOPSIS

```
#include <pbs_error.h>
#include <pbs_ifl.h>
```

```
struct batch_status *pbs_selstat(int connect, struct attrpl *sel_list,
struct attrl *rattrib, char *extend)
```

```
void pbs_statfree(struct batch_status *psj)
```

DESCRIPTION

Issue a batch request to examine the status of jobs which meet certain criteria. `pbs_selstat()` returns a list of `batch_status` structures for those jobs which met the selection criteria.

The `sel_list` struct holds the selection criteria. The `rattrib` struct holds the list of attributes whose values are to be returned.

This function is a combination of `pbs_selectjobs()` and `pbs_statjob()`. It is an extension to the POSIX Batch standard.

Initially all batch jobs are selected for which the user is authorized to query status. This set may be reduced or filtered by specifying certain attributes of the jobs.

A Select Status batch request is generated and sent to the server over the connection specified by `connect` which is the return value of `pbs_connect()`.

The parameter, `sel_list`, is a pointer to an `attrpl` structure which is defined in `pbs_ifl.h` as:

```
struct attrpl {
    struct attrpl *next;
    char *name;
    char *resource;
    char *value;
    enum batch_op op;
};
```

The `sel_list` list is terminated by the first entry where `next` is a null pointer.

The `name` member points to a string which is the name of the attribute. Not all of the job attributes may be used as a selection criteria. The `resource` member points to a string which is the name of a resource. This member is only used when `name` is set to `ATTR_1`, otherwise it should be a pointer to a null string. The `value` member points to a string which is the value of the attribute or resource. The attribute names are listed in `pbs_job_attributes.7B`.

The `op` member defines the operator in the logical expression:

`value operator current_value`

The logical expression must evaluate as true for the job to be selected. The permissible values of `op` are defined in `pbs_ifl.h` as:

“enum batch_op { ..., EQ, NE, GE, GT, LE, LT, ... },”.

The attributes marked with (E) in the description above may only be selected with the equal, EQ, or not equal, NE, operators.

If `sel_list` itself is a null pointer, then no selection is done on the basis of attributes.

The `parameter`, `rattrib`, is a pointer to an `attrl` structure which is defined below. The `rattrib` list is terminated by the first entry where `next` is a null pointer. If `attrib` is given, then only the attributes in the list are returned by the server. Otherwise, all the attributes of a job are returned. When an `attrib` list is specified, the `name` member is a pointer to an attribute name as listed in `pbs_alter(3)` and `pbs_submit(3)`. The `resource` member is only used if the `name` member is `ATTR_1`, otherwise it should be a pointer to a null string. The `value` member should always be a pointer to a null string.

The `return` value is a pointer to a list of `batch_status` structures or the null pointer if no jobs can be queried for status. The `batch_status` structure is defined in `pbs_ifl.h` as

```
struct batch_status {
    struct batch_status *next;
    char                *name;
    struct attrl        *attribs;
    char                *text;
}
```

The entry, `attribs`, is a pointer to a list of `attrl` structures defined in `pbs_ifl.h` as:

```

struct attrl {
    struct attrl *next;
    char      *name;
    char      *resource;
    char      *value;
};

```

It is up to the user to free the list of `batch_status` structures when no longer needed, by calling `pbs_statfree()`.

The `extend` parameter is for optional features and or additions. Normally, this should be null pointer.

When `pbs_selstat` is used to retrieve the `Submit_arguments` job attribute, PBS returns an XML-encoded string value according to the HPCBP specification.

Finished and Moved Jobs

In order to get information on finished and moved jobs, you must add an 'x' character to the `extend` parameter. The `extend` parameter is a character string; set one character to be the 'x' character. For example:

```
pbs_selstat ( ..., ..., ..., extend) ...
```

To get information on finished and moved jobs only, specify the Finished ('F') and moved ('M') job states. You must also use the `extend` character string containing the 'x' character. For example:

```

sel_list->next = sel_list;
sel_list->name = ATTR_state;
sel_list->value = "MF";
sel_list->op = EQ;
pbs_selstat ( ..., sel_list, ..., extend) ...

```

Subjobs are not considered finished until the parent array job is finished.

SEE ALSO

`qselect(1B)`, `pbs_alterjob(3B)`, `pbs_connect(3B)`, `pbs_statjob(3B)`, and `pbs_selectjob(3B)`.

DIAGNOSTICS

When the batch request generated by `pbs_selstat()` function has been completed successfully by a batch server, the routine will return a pointer to the list of `batch_status` structures. If no jobs met the criteria or an error occurred, the return will be the null pointer. If an error occurred, the global integer `pbs_errno` will be set to a non-zero value.

pbs_sigjob

send a signal to a pbs batch job

SYNOPSIS

```
#include <pbs_error.h>
#include <pbs_ifl.h>
```

```
int pbs_sigjob(int connect, char *job_id, char *signal, char *extend)
```

DESCRIPTION

Issue a batch request to send a signal to a batch job.

A Signal Job batch request is generated and sent to the server over the connection specified by `connect` which is the return value of `pbs_connect()`. If the batch job is in the running state, the batch server will send the job the signal number corresponding to the signal named in `signal`.

The argument, `job_id`, identifies which job is to be signaled, it is specified in the form:

“sequence_number.server”

The signal argument is the name of a signal. It may be the alphabetic form with or without the SIG prefix, or it may be a numeric string for the signal number. Two special names are recognized, `suspend` and `resume`. If the name of the signal is not a recognized signal name on the execution host, no signal is sent and an error is returned. If the job is not in the running state, no signal is sent and an error is returned, except when the signal is `resume` and the job is suspended.

The parameter, `extend`, is reserved for implementation defined extensions.

SEE ALSO

`qsig(1B)` and `pbs_connect(3B)`

DIAGNOSTICS

When the batch request generated by `pbs_sigjob()` function has been completed successfully by a batch server, the routine will return 0 (zero). Otherwise, a non zero error is returned. The error number is also set in `pbs_errno`.

pbs_stagein

request that files for a pbs batch job be staged in.

SYNOPSIS

```
#include <pbs_error.h>
#include <pbs_ifl.h>
```

```
int pbs_stagein(int connect, char *job_id, char *location, char *extend)
```

DESCRIPTION

Issue a batch request to start the stage in of files specified in the stagein attribute of a batch job.

A stage in batch request is generated and sent to the server over the connection specified by connect which is the return value of pbs_connect().

This request directs the server to begin the stage in of files specified in the job's stage in attribute. This request requires that the issuing user have operator or administrator privilege.

The argument, job_id, identifies which job for which file staging is to begin. It is specified in the form:

“sequence_number.server”

The argument, location, if not the null pointer or null string, specifies the location where the job will be run and hence to where the files will be staged. The location is the name of a host in the cluster managed by the server. If the job is then directed to run at different location, the run request will be rejected.

The argument, extend, is reserved for implementation defined extensions.

SEE ALSO

qrun(8B), qsub(1B), and pbs_connect(3B)

DIAGNOSTICS

When the batch request generated by pbs_stagein() function has been completed successfully by a batch server, the routine will return 0 (zero). Otherwise, a non zero error is returned. The error number is also set in pbs_errno.

pbs_statfree

NAME

pbs_statfree - free a PBS status object

SYNOPSIS

```
#include <pbs_error.h>
#include <pbs_ifl.h>
```

```
void pbs_statfree(struct batch_status *psj)
```

DESCRIPTION

Frees the specified PBS status object returned by pbs_statque, pbs_statserver, pbs_stathook, etc.

The argument is a pointer to a batch_status structure. The batch_status structure is defined in pbs_ifl.h as

```
struct batch_status {
    struct batch_status *next;
    char                *name;
    struct attrl        *attribs;
    char                *text;
}
```

No error information is returned.

pbs_statjob

obtain status of pbs batch jobs

SYNOPSIS

```
#include <pbs_error.h>
#include <pbs_ifl.h>
```

```
struct batch_status *pbs_statjob(int connect, char *id, struct attrl *attrib, char *extend)
```

```
void pbs_statfree(struct batch_status *psj)
```

DESCRIPTION

Issue a batch request to obtain the status of a specified batch job or a set of jobs at a destination.

A Status Job batch request is generated and sent to the server over the connection specified by connect which is the return value of pbs_connect().

The parameter, id, may be either a job identifier or a destination identifier.

If id is a job identifier, it is the identifier of the job for which status is requested. It is specified in the form:

“sequence_number.server”

If id is a destination identifier, it specifies that status of all jobs at the destination (queue) which the user is authorized to see be returned. If id is the null pointer or a null string, the status of each job at the server which the user is authorized to see is returned.

The parameter, attrib, is a pointer to an attrl structure which is defined in pbs_ifl.h as:

```
struct attrl {
    struct attrl *next;
    char *name;
    char *resource;
    char *value;
};
```

The attrib list is terminated by the first entry where next is a null pointer. If attrib is given, then only the attributes in the list are returned by the server. Otherwise, all the attributes of a job are returned. When an attrib list is specified, the name member is a pointer to a attribute name as listed in pbs_alter(3) and pbs_submit(3). The resource member is only used if the name member is ATTR_l, otherwise it should be a pointer to a null string. The

value member should always be a pointer to a null string.

The parameter, extend, is reserved for implementation defined extensions.

The return value is a pointer to a list of batch_status structures or the null pointer if no jobs can be queried for status. The batch_status structure is defined in pbs_ifl.h as

```
struct batch_status {
    struct batch_status *next;
    char                *name;
    struct attrl        *attribs;
    char                *text;
}
```

It is up to the user to free the structure when no longer needed, by calling pbs_statfree().

When pbs_statjob is used to retrieve the Submit_arguments job attribute, PBS returns an XML-encoded string value according to the HPCBP specification.

Finished and Moved Jobs

When querying for multiple jobs, to get information on finished and moved jobs, you must add an 'x' character to the extend parameter. The extend parameter is a character string; set one character to be the 'x' character.

When querying for multiple jobs, to get information on finished and moved jobs only, specify the Finished ('F') and moved ('M') job states. You must also use the extend character string containing the 'x' character.

When querying for a single finished job, the extend string does not need to contain the 'x' character.

Subjobs are not considered finished until the parent array job is finished.

SEE ALSO

qstat(1B) and pbs_connect(3B)

DIAGNOSTICS

When the batch request generated by pbs_statjob() function has been completed successfully and the status of each job has been returned by the batch server, the routine will return a pointer to the list of batch_status structures. If no jobs were available to query or an error occurred, a null pointer is returned. The global integer pbs_errno should be examined to determine the cause.

pbs_statnode, pbs_statvnode, pbs_stathost

obtain status of PBS vnodes or hosts

SYNOPSIS

```
#include <pbs_error.h>
#include <pbs_ifl.h>

struct batch_status *pbs_stathost(int connect, char *id,
struct attrl *attrib, char *extend)

struct batch_status *pbs_statnode(int connect, char *id,
struct attrl *attrib, char *extend)

struct batch_status *pbs_statvnode(int connect, char *id,
struct attrl *attrib, char *extend)

void pbs_statfree(struct batch_status *psj)
```

DESCRIPTION

Issue a batch request to obtain the status of PBS execution hosts or vnodes.

`pbs_stathost` returns information about the single host named in the call or about all hosts known to the PBS Server.

`pbs_statnode` is identical to `pbs_stathost` in function. It is retained for backward compatibility.

`pbs_statvnode` returns information about the single virtual node (vnode) named in the call or about all vnodes known to the PBS Server.

A Status Node batch request is generated and sent to the server over the connection specified by `connect` which is the return value of `pbs_connect()`.

The `id` is the name of a host for `pbs_stathost`, or a vnode for `pbs_statvnode`, or the null string. If `id` specifies a name, the status of that host or vnode will be returned. If the `id` is a null string (or null pointer), the status of all hosts or vnodes at the server will be returned.

The parameter, `attrib`, is a pointer to an `attrl` structure which is defined in `pbs_ifl.h` as:

```
struct attrl {
    struct attrl *next;
    char      *name;
    char      *resource;
    char      *value;
};
```

The attrb list is terminated by the first entry where next is a null pointer. If attrb is given, then only the attributes in the list are returned by the server. Otherwise, all the attributes of a node are returned. When an attrb list is specified, the name member is a pointer to an attribute name. The resource member is not used and must be a pointer to a null string. The value member should always be a pointer to a null string.

The parameter, extend, is reserved for implementation defined extensions.

The return value is a pointer to a list of batch_status structures, which is defined in pbs_ifl.h as:

```
struct batch_status {
    struct batch_status *next;
    char      *name;
    struct attrl *attrb;
    char      *text;
}
```

It is up to the user to free the structure when no longer needed, by calling pbs_statfree().

DIAGNOSTICS

When the batch request generated by pbs_stathost(), pbs_statnode(), or pbs_statvnode() function has been completed successfully by a batch server, the routine will return a pointer to the batch_status structure. Otherwise, a null pointer is returned and the error code is set in the global integer pbs_erno.

SEE ALSO

qstat(1B), pbs_connect(3B)

pbs_statque

obtain status of pbs batch queues

SYNOPSIS

```
#include <pbs_error.h>
#include <pbs_ifl.h>

struct batch_status *pbs_statque(int connect, char *id,
struct attrl *attrib, char *extend)

void pbs_statfree(struct batch_status *psj)
```

DESCRIPTION

Issue a batch request to obtain the status of a batch queue.

A Status Queue batch request is generated and sent to the server over the connection specified by connect which is the return value of pbs_connect().

The id is the name of a queue, in the form:

```
queue_name
or the null string. If
queue_name
is specified, the status of the queue named
queue_name
at the server is returned. If the id is a null string or null pointer,
the status of all queues at the server is returned.
```

The parameter, attrib, is a pointer to an attrl structure which is defined in pbs_ifl.h as:

```
struct attrl {
    struct attrl *next;
    char *name;
    char *resource;
    char *value;
};
```

The attrib list is terminated by the first entry where next is a null pointer. If attrib is given, then only the attributes in the list are returned by the server. Otherwise, all the attributes of a queue are

returned. When an attrib list is specified, the name member is a pointer to an attribute name as listed in `pbs_alterjob(3B)` and `pbs_submit(3B)`. The resource member is only used if the name member is `ATTR_1`, otherwise it should be a pointer to a null string. The value member should always be a pointer to a null string.

When `pbs_statque` is used to get the attributes of an object, a single attrl data structure is returned for each parameterized attribute.

The parameter, `extend`, is reserved for implementation defined extensions.

The return value is a pointer to a list of `batch_status` structures, which is defined in `pbs_ifl.h` as:

```
struct batch_status {
    struct batch_status *next;
    char                *name;
    struct attrl        *attribs;
    char                *text;
}
```

It is up the user to free the structure when no longer needed, by calling `pbs_statfree()`.

SEE ALSO

`qstat(1B)` and `pbs_connect(3B)`

DIAGNOSTICS

When the batch request generated by `pbs_statque()` function has been completed successfully by a batch server, the routine will return a pointer to the `batch_status` structure. Otherwise, a null pointer is returned and the error code is set in the global integer `pbs_erno`.

pbs_statresv

obtain status information about reservations

SYNOPSIS

```
#include <pbs_error.h>
#include <pbs_ifl.h>

struct batch_status *pbs_statresv(int connect, char *id,
struct attrl *attrib, char *extend)

void pbs_statfree(struct batch_status *psj)
```

DESCRIPTION

Issue a batch request to obtain the status of a specified reservation or a set of reservations at a destination.

A Status Reservation batch request is generated and sent to the server over the connection specified by connect which is the return value of pbs_connect().

The parameter, id, is a reservation identifier. A reservation identifier is of the form:

“R<sequence_number>.<server>”

If id is the null pointer or a null string, the status of each reservation at the server which the user is authorized to see is returned.

The parameter, attrib, is a pointer to an attrl structure which is defined in pbs_ifl.h as:

```
struct attrl {
    struct attrl *next;
    char *name;
    char *resource;
    char *value;
};
```

The attrib list is terminated by the first entry where next is a null pointer. If attrib is given, then only the attributes in the list are returned by the server. Otherwise, all the attributes of a reservation are returned. When an attrib list is specified, the name member is a

pointer to a attribute name as listed in `pbs_submit_resv(3)`. The resource member is only used if the name member is `ATTR_1`, otherwise it should be a pointer to a null string. The value member should always be a pointer to a null string.

The parameter, `extend`, is reserved for implementation defined extensions.

The return value is a pointer to a list of `batch_status` structures or the null pointer if no reservations can be queried for status. The `batch_status` structure is defined in `pbs_ifl.h` as

```
struct batch_status {
    struct batch_status *next;
    char                *name;
    struct attrl        *attrs;
    char                *text;
}
```

It is up the user to free the structure when no longer needed, by calling `pbs_statfree()`.

SEE ALSO

`pbs_rstat(1B)` and `pbs_connect(3B)`

DIAGNOSTICS

When the batch request generated by `pbs_statresv()` function has been completed successfully and the status of each reservation has been returned by the batch server, the routine will return a pointer to the list of `batch_status` structures. If no reservations were available to query or an error occurred, a null pointer is returned. The global integer `pbs_errno` should be examined to determine the cause.

pbs_statsched

obtain status of PBS scheduler

SYNOPSIS

```
#include <pbs_error.h>
#include <pbs_ifl.h>
```

```
struct batch_status *pbs_statsched(int connect, struct attrl *attrib,
char *extend)
```

```
void pbs_statfree(struct batch_status *psj)
```

DESCRIPTION

Issue a batch request to obtain the status of PBS scheduler.

A Status Scheduler batch request is generated and sent to the server. The parameter `connect` is the return value of `pbs_connect()`.

The parameter, `attrib`, is a pointer to an `attrl` structure which is defined in `pbs_ifl.h` as:

```
struct attrl {
    struct attrl *next;
    char *name;
    char *resource;
    char *value;
};
```

The `attrib` list is terminated by the first entry where `next` is a null pointer. If `attrib` is given, then only the attributes in the list are returned by the server. Otherwise, all the attributes of the scheduler are returned. When an `attrib` list is specified, the `name` member is a pointer to an attribute name as listed in `pbs_alter(3)` and `pbs_submit(3)`. The `resource` member is only used if the `name` member is `ATTR_L`, otherwise it should be a pointer to a null string. The `value` member should always be a pointer to a null string.

The parameter, `extend`, is reserved for implementation-defined extensions.

The return value of `pbs_statsched()` is a pointer to a list of

batch_status structures, which is defined in pbs_ifl.h as:

```
struct batch_status {
    struct batch_status *next;
    char                *name;
    struct attrl        *attrs;
    char                *text;
}
```

It is up the user to free the batch_status structure when it is no longer needed, by calling pbs_statfree().

SEE ALSO

qstat(1B) and pbs_connect(3B)

DIAGNOSTICS

When the batch request generated by pbs_statsched() has been completed successfully by the PBS server, pbs_statsched() will return a pointer to a batch_status structure. Otherwise, a null pointer is returned and the error code is set in pbs_erno.

pbs_statserver

obtain status of a pbs batch server

SYNOPSIS

```
#include <pbs_error.h>
#include <pbs_ifl.h>
```

```
struct batch_status *pbs_statserver(int connect, struct attrl *attrib,
char *extend)
```

```
void pbs_statfree(struct batch_status *psj)
```

DESCRIPTION

Issue a batch request to obtain the status of a batch server.

A Status Server batch request is generated and sent to the server over the connection specified by `connect` which is the return value of `pbs_connect()`.

The parameter, `attrib`, is a pointer to an `attrl` structure which is defined in `pbs_ifl.h` as:

```
struct attrl {
    struct attrl *next;
    char      *name;
    char      *resource;
    char      *value;
};
```

The `attrib` list is terminated by the first entry where `next` is a null pointer. If `attrib` is given, then only the attributes in the list are returned by the server. Otherwise, all the attributes of the server are returned. When an `attrib` list is specified, the `name` member is a pointer to an attribute name as listed in `pbs_alterjob(3B)` and `pbs_submit(3B)`. The `resource` member is only used if the `name` member is `ATTR_1`, otherwise it should be a pointer to a null string. The `value` member should always be a pointer to a null string.

When `pbs_statserver` is used to get the attributes of an object, a single `attrl` data structure is returned for each parameterized attribute.

The parameter, `extend`, is reserved for implementation defined extensions.

The `return` value is a pointer to a list of `batch_status` structures, which is defined in `pbs_ifl.h` as:

```
struct batch_status {
    struct batch_status *next;
    char                *name;
    struct attrl        *attrs;
    char                *text;
}
```

It is up the user to free the space when no longer needed, by calling `pbs_statfree()`.

SEE ALSO

`qstat(1B)` and `pbs_connect(3B)`

DIAGNOSTICS

When the batch request generated by `pbs_statserver()` function has been completed successfully by a batch server, the routine will return a pointer to a `batch_status` structure. Otherwise, a null pointer is returned and the error code is set in `pbs_errno`.

pbs_submit

submit a pbs batch job

SYNOPSIS

```
#include <pbs_error.h>
#include <pbs_ifl.h>

char *pbs_submit(int connect, struct attrop1 *attrib,
char *script, char *destination, char *extend)
```

DESCRIPTION

Issue a batch request to submit a new batch job.

A Queue Job batch request is generated and sent to the server over the connection specified by connect which is the return value of pbs_connect(). The job will be submitted to the queue specified by destination .

The parameter, attrib , is a list of attrop1 structures which is defined in pbs_ifl.h as:

```
struct attrl {
    char *name;
    char *resource;
    char *value;
    struct attrl *next;
    enum batch_op op;
};
```

The attrib list is terminated by the first entry where next is a null pointer.

The name member points to a string which is the name of the attribute. The value member points to a string which is the value of the attribute. The attribute names are defined in pbs_job_attributes(7B).

If an attribute is not named in the attrib array, the default action will be taken. It will either be assigned the default value or will not be passed with the job. The action depends on the attribute. If attrib itself is a null pointer, then the default action will be taken for each attribute.

Associated with an attribute of type ATTR_1 (the letter ell) is a resource name indicated by resource in the attr1 structure. All other attribute types should have a pointer to a null string for resource .

The op member is forced to a value of
SET
by pbs_submit().

The parameter, script , is the path name to the job script. If the path name is relative, it will be expanded to the processes current working directory. If script is a null pointer or the path name pointed to is specified as the null string, no script is passed with the job.

The destination parameter specifies the destination for the job. It is specified as:

[queue]

If destination is the null string or the queue is not specified, the destination will be the default queue at the connected server.

The parameter, extend , is reserved for implementation defined extensions.

The return value is a character string which is the job_identifier assigned to the job by the server. The space for the job_identifier string is allocated by pbs_submit() and should be released via a call to free() by the user when no longer needed.

SEE ALSO

qsub(1B) and pbs_connect(3B)

DIAGNOSTICS

When the batch request generated by pbs_submit() function has been completed successfully by a batch server, the routine will return a pointer to a character string which is the job identifier of the submitted batch job. Otherwise, a null pointer is returned and the error code is set in pbs_error.

pbs_submit_resv

submit a pbs reservation

SYNOPSIS

```
#include <pbs_error.h>
#include <pbs_ifl.h>
```

```
char *pbs_submit_resv(int connect, struct attrpl *attrib, char *extend)
```

DESCRIPTION

Issue a batch request to submit a new reservation.

A Submit Reservation batch request is generated and sent to the server over the connection specified by connect which is the return value of pbs_connect().

The parameter, attrib, is a list of attrpl structures which is defined in pbs_ifl.h as:

```
struct attrl {
    char *name;
    char *resource;
    char *value;
    struct attrl *next;
    enum batch_op op;
};
```

The attrib list is terminated by the first entry where next is a null pointer.

The name member points to a string which is the name of the attribute. The value member points to a string which is the value of the attribute. The attribute names are defined in pbs_ifl.h.

If an attribute is not named in the attrib array, the default action will be taken. It will either be assigned the default value or will not be passed with the reservation. The action depends on the attribute. If attrib itself is a null pointer, then the default action will be taken for each attribute.

Associated with an attribute of type ATTR_1 (the letter ell) is a

resource name indicated by resource in the attrl structure. All other attribute types should have a pointer to a null string for resource .

The op member is forced to a value of
SET
by pbs_submit_resv().

The parameter, extend , is reserved for implementation defined extensions.

The return value is a character string which is the reservation_identifier assigned to the job by the server. The space for the reservation_identifier string is allocated by pbs_submit_resv() and should be released via a call to free() by the user when no longer needed.

SEE ALSO

pbs_rsub(1B) and pbs_connect(3B)

DIAGNOSTICS

When the batch request generated by pbs_submit_resv() function has been completed successfully by a batch server, the routine will return a pointer to a character string which is the job identifier of the submitted batch job. Otherwise, a null pointer is returned and the error code is set in pbs_error.

pbs_terminate

terminate a pbs batch server

SYNOPSIS

```
#include <pbs_error.h>
#include <pbs_ifl.h>
```

```
int pbs_terminate(int connect, int manner, char *extend)
```

DESCRIPTION

Issue a batch request to shut down a batch server. This request requires the privilege level usually reserved for batch operators and administrators.

A Server Shutdown batch request is generated and sent to the server over the connection specified by `connect` which is the return value of `pbs_connect()`.

The parameter, `manner`, specifies the manner in which the server is shut down. The available manners are defined in `pbs_ifl.h`.

The server will not respond to the batch request until the server has completed its termination procedure.

The parameter, `extend`, is reserved for implementation defined extensions.

This call requires PBS Operator or Manager privilege.

SEE ALSO

`qterm(8B)` and `pbs_connect(3B)`

DIAGNOSTICS

When the batch request generated by `pbs_terminate()` function has been completed successfully by a batch server, the routine will return 0 (zero). Otherwise, a non zero error is returned. The error number is also set in `pbs_errno`.

Chapter 5

RPP Library

This chapter discusses the Reliable Packet Protocol (RPP) used by PBS. These functions provide reliable, flow-controlled, two-way transmission of data. Each data path will be called a "stream" in this document. The advantage of RPP over TCP is that many streams can be multiplexed over one socket. This allows simultaneous connections over many streams without regard to the system imposed file descriptor limit.

5.1 RPP Library Routines

The following manual pages document the application programming interface provided by the RPP library.

rpp_open, rpp_bind, rpp_poll, rpp_io, rpp_read, rpp_write, rpp_close, rpp_getaddr, rpp_flush, rpp_terminate, rpp_shutdown, rpp_rcommit, rpp_wcommit, rpp_eom, rpp_getc, rpp_putc

reliable packet protocol

SYNOPSIS

```
#include <sys/types.h>
#include <netinet/in.h>
#include <rpp.h>
```

```
int rpp_open(addr)
    struct sockadd_in *addr;
```

```
int rpp_bind(port)
    int port;
```

```
int rpp_poll()
```

```
int rpp_io()
```

```
int rpp_read(stream, buf, len)
    u_int stream;
    char *buf;
    int len;
```

```
int rpp_write(stream, buf, len)
    u_int stream;
    char *buf;
    int len;
```

```
int rpp_close(stream)
    u_int stream;
```

```
struct sockadd_in *rpp_getaddr(stream)
    u_int stream;
```

```
int rpp_flush(stream)
    u_int stream;
```

```
int rpp_terminate()
```

```
int rpp_shutdown()
```

```
int rpp_rcommit(stream, flag)
    u_int stream;
    int flag;
```

```
int rpp_wcommit(stream, flag)
    u_int stream;
    int flag;
```

```
int rpp_eom(stream)
    u_int stream;
```

```
int rpp_getc(stream)
    u_int stream;
```

```
int rpp_putc(stream, c)
    u_int stream;
    int c;
```

DESCRIPTION

These functions provide reliable, flow-controlled, two-way transmission of data. Each data path will be called a “stream” in this document. The advantage of RPP over TCP is that many streams can be multiplexed over one socket. This allows simultaneous connections over many streams without regard to the system imposed file descriptor limit.

Data is sent and received in “messages”. A message may be of any length and is either received completely or not at all. Long messages will cause the library to use large amounts of memory in the heap by calling `malloc(3V)`.

In order to use any of the above with Windows, initialize the network library and link with `winsock2`. To do this, call `winsock_init()` before calling the function and link against the `ws2_32.lib` library.

`rpp_open()` initializes a new stream connection to `addr` and returns the stream identifier. This is an integer with a value greater than or equal to zero. A negative number indicates an error. In this case, `errno` will be set.

`rpp_bind()` is an initialization call which is used to bind the UDP socket used by RPP to a particular port. The file descriptor of the UDP socket used by the library is returned.

`rpp_poll()` returns the stream identifier of a stream with data to read. If no stream is ready to read, a -2 is returned. A -1 is returned if an error occurs.

`rpp_io()` processes any packets which are waiting to be sent or received over the UDP socket. This routine should be called if a section of code could be executing for more than a few (~10) seconds without calling any other rpp function. A -1 is returned if an error occurs, 0 otherwise.

`rpp_read()` transfers up to `len` characters of a message from stream into `buf`. If all of a message has been read, the return value will be less than `len`. The return value could be zero if all of a message had previously been read. A -1 is returned on error. A -2 is returned if the peer has closed its connection. If `rpp_poll()` is used to determine the stream is ready for reading, the call to `rpp_read()` will return immediately. Otherwise, the call will block waiting for a message to arrive.

`rpp_write()` adds information to the current message on a stream. The data in `buf` numbering `len` characters is transferred to the stream. The number of characters added to the stream are returned or a -1 on error. In this case, `errno` will be set. A -2 is returned if the peer has closed its connection.

`rpp_close()` disconnects the stream from its peer and frees all resources associated with the stream. The return value is -1 on error and 0 otherwise.

`rpp_getaddr()` returns the address which a stream is connected to. If the stream is not open, a NULL pointer is returned.

`rpp_flush()` marks the end of a message and commits all the data which has been written to the specified stream. A zero is returned if the message has been successfully committed. A -1 is returned on error.

`rpp_terminate()` is used to free all memory associated with all streams and close the UDP socket. This is done without attempting to send any final messages that may be waiting. If a process is using rpp and

calls `fork()`, the child must call `rpp_terminate()` so it will not cause a conflict with the parent's communication.

`rpp_shutdown()` is used to free all memory associated with all streams and close the UDP socket. An attempt is made to send all outstanding messages before returning.

`rpp_rcommit()` is used to "commit" or "de-commit" the information read from a message. As calls are made to `rpp_read()`, the number of characters transferred out of the message are counted. If `rpp_rcommit()` is called with flag being non-zero (TRUE), the current position in the message is marked as the commit point. If `rpp_rcommit()` is called with flag being zero (FALSE), a subsequent call to `rpp_read()` will return characters from the message following the last commit point. If an entire message has been read, `rpp_read()` will continue to return zero as the number of bytes transferred until `rpp_eom()` is called to commit the complete message.

`rpp_wcommit()` is used to "commit" or "de-commit" the information written to a stream. As calls are made to `rpp_write()`, the number of characters transferred into the message are counted. If `rpp_wcommit()` is called with flag being non-zero (TRUE), the current position in the message is marked as the commit point. If `rpp_wcommit()` is called with flag being zero (FALSE), a subsequent call to `rpp_write()` will transfer characters into the stream following the last commit point. A call to `rpp_flush()` does an automatic write commit to the current position.

`rpp_eom()` is called to terminate processing of the current message.

SEE ALSO

`tcp(4P)`, `udp(4P)`

Chapter 6

TM Library

This chapter describes the PBS Task Management library. The TM library is a set of routines used to manage multi-process, parallel, and distributed applications. The current version is an implementation of the proposed (draft) PSCHED standard sponsored by NASA. Altair has since submitted this draft to the DRAMA working group of the international Global Grid Forum standards body.

6.1 TM Library Routines

The following manual pages document the application programming interface provided by the TM library.

**tm_init, tm_nodeinfo, tm_poll, tm_notify, tm_spawn, tm_kill, tm_obit,
tm_taskinfo, tm_atnode, tm_rescinfo, tm_publish, tm_subscribe,
tm_finalize, tm_attach**

task management API

SYNOPSIS

```
#include <tm.h>

int tm_init(info, roots)
    void *info;
    struct tm_roots *roots;

int tm_nodeinfo(list, nnodes)
    tm_node_id **list;
    int *nnodes;

int tm_poll(poll_event, result_event, wait, tm_errno)
    tm_event_t poll_event;
    tm_event_t *result_event;
    int wait;
    int *tm_errno;

int tm_notify(tm_signal)
    int tm_signal;

int tm_spawn(argc, argv, envp, where, tid, event)
    int argc;
    char **argv;
    char **envp;
    tm_node_id where;
    tm_task_id *tid;
    tm_event_t *event;

int tm_kill(tid, sig, event)
    tm_task_id tid;
    int sig;
    tm_event_t *event;

int tm_obit(tid, obitval, event)
    tm_task_id tid;
```

```

int *obitval;
tm_event_t *event;

int tm_taskinfo(node, tid_list, list_size, ntasks, event)
    tm_node_id node;
    tm_task_id *tid_list;
    int list_size;
    int *ntasks;
    tm_event_t *event;

int tm_atnode(tid, node)
    tm_task_id tid;
    tm_node_id *node;

int tm_rescinfo(node, resource, len, event)
    tm_node_id node;
    char *resource;
    int len;
    tm_event_t *event;

int tm_publish(name, info, len, event)
    char *name;
    void *info;
    int len;
    tm_event_t *event;

int tm_subscribe(tid, name, info, len, info_len, event)
    tm_task_id tid;
    char *name;
    void *info;
    int len;
    int *info_len;
    tm_event_t *event;

int tm_attach(jobid, cookie, pid, tid, host, port)
    char *jobid;
    char *cookie;
    pid_t pid;
    tm_task_id *tid;
    char *host;
    int port;

```

```
int tm_finalize()
```

DESCRIPTION

These functions provide a partial implementation of the task management interface part of the PSCHED API. In PBS, MOM provides the task manager functions. This library opens a tcp socket to the MOM running on the local host and sends and receives messages using the DIS protocol (described in the PBS IDS). The tm interface can only be used by a process within a PBS job.

The PSCHED Task Management API description used to create this library was committed to paper on November 15, 1996 and was given the version number 0.1. Changes may have taken place since that time which are not reflected in this library.

The API description uses several data types that it purposefully does not define. This was done so an implementation would not be confined in the way it was written. For this specific work, the definitions follow:

```
typedef int      tm_node_id; /* job-relative node id */
#define TM_ERROR_NODE ((tm_node_id)-1)
typedef int      tm_event_t; /* > 0 for real events */
#define TM_NULL_EVENT ((tm_event_t)0)
#define TM_ERROR_EVENT ((tm_event_t)-1)
typedef unsigned long tm_task_id;
#define TM_NULL_TASK (tm_task_id)0
```

There are a number of error values defined as well: TM_SUCCESS, TM_ESYSTEM, TM_ENOEVENT, TM_ENOTCONNECTED, TM_EUNKNOWNCMD, TM_ENOTIMPLEMENTED, TM_EBADENVIRONMENT, TM_ENOTFOUND.

The functions listed here are not supported on Windows.

tm_init() initializes the library by opening a socket to the MOM on the local host and sending a TM_INIT message, then waiting for the reply. The info parameter has no use and is included to conform with the PSCHED document. The roots pointer will contain valid data after the function returns and has the following structure:

```
struct tm_roots {
    tm_task_id  tm_me;
```

```

tm_task_id  tm_parent;
int         tm_nnodes;
int         tm_ntasks;
int         tm_taskpoolid;
tm_task_id  *tm_tasklist;
};

```

tm_me	The task id of this calling task.
tm_parent	The task id of the task which spawned this task or TM_NULL_TASK if the calling task is the initial task started by PBS.
tm_nnodes	The number of nodes allocated to the job.
tm_ntasks	This will always be 0 for PBS.
tm_taskpoolid	PBS does not support task pools so this will always be -1.
tm_tasklist	This will be NULL for PBS.

The tm_ntasks, tm_taskpoolid and tm_tasklist fields are not filled with data specified by the PSCHED document. PBS does not support task pools and, at this time, does not return information about current running tasks from tm_init. There is a separate call to get information for current running tasks called tm_taskinfo which is described below. The return value from tm_init is TM_SUCCESS if the library initialization was successful, or an error is returned otherwise.

tm_nodeinfo() places a pointer to a malloc'ed array of tm_node_id's in the pointer pointed at by list. The order of the tm_node_id's in list is the same as that specified to MOM in the "exec_host" attribute. The int pointed to by nnodes contains the number of nodes allocated to the job. This is information that is returned during initialization and does not require communication with MOM. If tm_init has not been called, TM_ESYSTEM is returned, otherwise TM_SUCCESS is returned.

tm_poll() is the function which will retrieve information about the task management system to locations specified when other routines request an action take place. The bookkeeping for this is done by generating an event for each action. When the task manager (MOM) sends a message that an action is complete, the event is reported by tm_poll and information is placed where the caller requested it. The argument poll_event is meant to be used to request a specific event. This implementation does not use it and it must be set to

TM_NULL_EVENT or an error is returned. Upon return, the argument `result_event` will contain a valid event number or TM_ERROR_EVENT on error. If wait is zero and there are no events to report, `result_event` is set to TM_NULL_EVENT. If wait is non-zero and there are no events to report, the function will block waiting for an event. If no local error takes place, TM_SUCCESS is returned. If an error is reported by MOM for an event, then the argument `tm_erno` will be set to an error code.

`tm_notify()` is described in the PSCHED documentation, but is not implemented for PBS yet. It will return TM_ENOTIMPLEMENTED.

`tm_spawn()` sends a message to MOM to start a new task. The node id of the host to run the task is given by `where`. The parameters `argc`, `argv` and `envp` specify the program to run and its arguments and environment very much like `exec()`. The full path of the program executable must be given by `argv[0]` and the number of elements in the `argv` array is given by `argc`. The array `envp` is NULL terminated. The argument event points to a `tm_event_t` variable which is filled in with an event number. When this event is returned by `tm_poll`, the `tm_task_id` pointed to by `tid` will contain the task id of the newly created task.

`tm_kill()` sends a signal specified by `sig` to the task `tid` and puts an event number in the `tm_event_t` pointed to by `event`.

`tm_obit()` creates an event which will be reported when the task `tid` exits. The `int` pointed to by `obitval` will contain the exit value of the task when the event is reported.

`tm_taskinfo()` returns the list of tasks running on the node specified by `node`. The PSCHED documentation mentions a special ability to retrieve all tasks running in the job. This is not supported by PBS. The argument `tid_list` points to an array of `tm_task_id`'s which contains `list_size` elements. Upon return, `event` will contain an event number. When this event is polled, the `int` pointed to by `ntasks` will contain the number of tasks running on the node and the array will be filled in with `tm_task_id`'s. If `ntasks` is greater than `list_size`, only `list_size` tasks will be returned.

`tm_atnode()` will place the node id where the task `tid` exists in the `tm_node_id` pointed to by `node`.

`tm_resinfo()` makes a request for a string specifying the resources available on a node given by the argument `node`. The string is returned

in the buffer pointed to by resource and is terminated by a NUL character unless the number of characters of information is greater than specified by len. The resource string PBS returns is formatted as follows:

A space separated set of strings from the uname system call. The order of the strings is sysname, nodename, release, version, machine.

A comma separated set of strings giving the components of the “Resource_List” attribute of the job, preceded by a colon (:). Each component has the resource name, an equal sign, and the limit value.

tm_publish() causes len bytes of information pointed at by info to be sent to the local MOM to be saved under the name given by name.

tm_subscribe() returns a copy of the information named by name for the task given by tid. The argument info points to a buffer of size len where the information will be returned. The argument info_len will be set with the size of the published data. If this is larger than the supplied buffer, the data will have been truncated.

tm_attach() commands MOM to create a new PBS “attached task” out of a session running on MOM’s host. The jobid parameter specifies the job which is to have a new task attached. If it is NULL, the system will try to determine the correct jobid. The cookie parameter must be NULL. The pid parameter must be a non-zero process id for the process which is to be added to the job specified by jobid. If tid is non-NULL, it will be used to store the task id of the new task. The host and port parameters specify where to contact MOM. host should be NULL. The return value will be 0 if a new task has been successfully created and non-zero on error. The return value will be one of the TM error numbers defined in tm.h as follows:

TM_ESYSTEM	MOM cannot be contacted
TM_ENOTFOUND	No matching job was found
TM_ENOTIMPLEMENTED	The call is not implemented/supported
TM_ESESSION	The session specified is already attached
TM_EUSER	The calling user is not permitted to attach
TM_EOWNER	The process owner does not match the job
TM_ENOPROC	The process does not exist

tm_finalize() may be called to free any memory in use by the library and close the connection to MOM.

Chapter 7

RM Library

This chapter describes the PBS Resource Monitor library. The RM library contains functions to facilitate communication with the PBS Professional resource monitor. It is set up to make it easy to connect to several resource monitors and handle the network communication efficiently.

7.1 RM Library Routines

The following “manual” pages document the application programming interface provided by the RM library.

openrm, closerm, downrm, configrm, addreq, allreq, getreq, flushreq, activereq, fullresp

resource monitor API

SYNOPSIS

```
#include <sys/types.h>
#include <netinet/in.h>
#include <rm.h>
```

```
int openrm (host, port)
char *host;
unsigned int port;
```

```
int closerm (stream)
int stream;
```

```
int downrm (stream)
int stream;
```

```
int configrm (stream, file)
int stream;
char *file;
```

```
int addreq (stream, line)
int stream;
char *line;
```

```
int allreq (line)
char *line;
```

```
char *getreq(stream)
int stream;
```

```
int flushreq()
```

```
int activereq()
```

```
void fullresp(flag)
int flag;
```

DESCRIPTION

The resource monitor library contains functions to facilitate communication with the PBS Professional resource monitor. It is set up to make it easy to connect to several resource monitors and handle the network communication efficiently.

In all these routines, the variable `pbs_errno` will be set when an error is indicated. The lower levels of network protocol are handled by the “Data Is Strings” DIS library and the “Reliable Packet Protocol” RPP library.

`configrm()` causes the resource monitor to read the file named. Deprecated.

`addreq()` begins a new message to the resource monitor if necessary. Then adds a line to the body of an outstanding command to the resource monitor.

`allreq()` begins, for each stream, a new message to the resource monitor if necessary. Then adds a line to the body of an outstanding command to the resource monitor.

`getreq()` finishes and sends any outstanding message to the resource monitor. If `fullresp()` has been called to turn off “full response” mode, the routine searches down the line to find the equal sign just before the response value. The returned string (if it is not NULL) has been allocated by `malloc` and thus `free` must be called when it is no longer needed to prevent memory leaks.

`flushreq()` finishes and sends any outstanding messages to all resource monitors. For each active resource monitor structure, it checks if any outstanding data is waiting to be sent. If there is, it is sent and the internal structure is marked to show “waiting for response”.

`fullresp()` turns on, if `flag` is true, “full response” mode where `getreq()` returns a pointer to the beginning of a line of response. This is the default. If `flag` is false, the line returned by `getreq()` is just the answer following the equal sign.

`activereq()` Returns the stream number of the next stream with something to read or a negative number (the return from `rpp_poll`) if there is no stream to read.

In order to use any of the above with Windows, initialize the network library and link with `winsock2`. To do this, call `winsock_init()` before calling the function and link against the `ws2_32.lib` library.

SEE ALSO

`rpp(3B)`, `tcp(4P)`, `udp(4P)`

Chapter 8

TCL/tk Interface

The PBS Professional software includes a TCL/tk interface to PBS. Wrapped versions of many of the API calls are compiled into a special version of the TCL shell, called `pbs_tclsh`. (A special version of the tk window shell is also provided, called `pbs_wish`.) This chapter documents the TCL/tk interface to PBS.

The `pbs_tclapi` is a subset of the PBS external API wrapped in a TCL library. This functionality allows the creation of scripts that query the PBS system. Specifically, it permits the user to query the `pbs_server` about the state of PBS, jobs, queues, and nodes, and communicate with `pbs_mom` to get information about the status of running jobs, available resources on nodes, etc.

8.1 TCL/tk API Functions

A set of functions to communicate with the PBS Server and resource monitor have been added to those normally available with Tcl. All these calls will set the Tcl variable `pbs_errno` to a value to indicate if an error occurred. In all cases, the value "0" means no error. If a call to a Resource Monitor function is made, any error value will come from the system supplied `errno` variable. If the function call communicates with the PBS Server, any error value will come from the error number returned by the Server. This is the same TCL interface used by the `pbs_tclsh` and `pbs_wish` commands.

Note that the `pbs_tclapi pbsrescquery` command, which calls the C API `pbs_rescquery`, is deprecated. Any attempt to use it will result in a `PBSE_NOSUPPORT` error being returned.

pbs_tclapi

PBS TCL Application Programming Interface

DESCRIPTION

The `pbs_tclapi` is a subset of the PBS external API wrapped in a TCL library. This functionality allows the creation of scripts that query the PBS system. Specifically, it permits the user to query the `pbs_server` about the state of PBS, jobs, queues, and nodes, and communicate with `pbs_mom` to get information about the status of running jobs, available resources on nodes, etc.

USAGE

A set of functions to communicate with the PBS server and resource monitor have been added to those normally available with Tcl. All these calls will set the Tcl variable “`pbs_erno`” to a value to indicate if an error occurred. In all cases, the value “0” means no error. If a call to a Resource Monitor function is made, any error value will come from the system supplied `errno` variable. If the function call communicates with the PBS Server, any error value will come from the error number returned by the server. This is the same TCL interface used by the `pbs_tclsh` and `pbs_wish` commands.

`openrm host ?port?`

Creates a connection to the PBS Resource Monitor on `host` using `port` as the port number or the standard port for the resource monitor if it is not given. A connection handle is returned. If the open is successful, this will be a non-negative integer. If not, an error occurred.

`closerm connection`

The parameter `connection` is a handle to a resource monitor which was previously returned from `openrm`. This connection is closed. Nothing is returned.

`downrm connection`

Sends a command to the connected resource monitor to shutdown. Nothing is returned.

`configrm connection filename`

Sends a command to the connected resource monitor to read the configuration file given by `filename`. If this is successful, a

“0” is returned, otherwise, “-1” is returned.

addreq connection request

A resource request is sent to the connected resource monitor. If this is successful, a “0” is returned, otherwise, “-1” is returned.

getreq connection

One resource request response from the connected resource monitor is returned. If an error occurred or there are no more responses, an empty string is returned.

allreq request

A resource request is sent to all connected resource monitors. The number of streams acted upon is returned.

flushreq

All resource requests previously sent to all connected resource monitors are flushed out to the network. Nothing is returned.

activereq

The connection number of the next stream with something to read is returned. If there is nothing to read from any of the connections, a negative number is returned.

fullresp flag

Evaluates flag as a boolean value and sets the response mode used by getreq to full if flag evaluates to “true”. The full return from a resource monitor includes the original request followed by an equal sign followed by the response. The default situation is only to return the response following the equal sign. If a script needs to “see” the entire line, this function may be used.

pbsstatserv

The server is sent a status request for information about the server itself. If the request succeeds, a list with three elements is returned, otherwise an empty string is returned. The first element is the server’s name. The second is a list of attributes. The third is the “text” associated with the server (usually blank).

pbsstatjob

The server is sent a status request for information about the all jobs resident within the server. If the request succeeds, a list is returned, otherwise an empty string is returned. The list contains an entry for each job. Each element is a list with three elements. The first is the job's jobid. The second is a list of attributes. The attribute names which specify resources will have a name of the form "Resource_List:name" where "name" is the resource name. The third is the "text" associated with the job (usually blank).

`pbsstatque`

The server is sent a status request for information about all queues resident within the server. If the request succeeds, a list is returned, otherwise an empty string is returned. The list contains an entry for each queue. Each element is a list with three elements. This first is the queue's name. The second is a list of attributes similar to `pbsstatjob`. The third is the "text" associated with the queue (usually blank).

`pbsstatnode`

The server is sent a status request for information about all nodes defined within the server. If the request succeeds, a list is returned, otherwise an empty string is returned. The list contains an entry for each node. Each element is a list with three elements. This first is the node's name. The second is a list of attributes similar to `pbsstatjob`. The third is the "text" associated with the node (usually blank).

`pbsselstat`

The server is sent a status request for information about the all runnable jobs resident within the server. If the request succeeds, a list similar to `pbsstatjob` is returned, otherwise an empty string is returned.

`pbsrunjob jobid ?location?`

Run the job given by jobid at the location given by location. If location is not given, the default location is used. If this is successful, a "0" is returned, otherwise, "-1" is returned.

`pbsasyrunjob jobid ?location?`

Run the job given by jobid at the location given by location without waiting for a positive response that the job has actually

started. If location is not given, the default location is used.
If this is successful, a “0” is returned, otherwise, “-1” is returned.

pbsrerunjob jobid

Re-runs the job given by jobid. If this is successful, a “0” is returned, otherwise, “-1” is returned.

pbsdeljob jobid

Delete the job given by jobid. If this is successful, a “0” is returned, otherwise, “-1” is returned.

pbsholdjob jobid

Place a hold on the job given by jobid. If this is successful, a “0” is returned, otherwise, “-1” is returned.

pbsmovejob jobid ?location?

Move the job given by jobid to the location given by location.
If location is not given, the default location is used. If this is successful, a “0” is returned, otherwise, “-1” is returned.

pbsqenable queue

Set the “enabled” attribute for the queue given by queue to true.
If this is successful, a “0” is returned, otherwise, “-1” is returned.

pbsqdisable queue

Set the “enabled” attribute for the queue given by queue to false. If this is successful, a “0” is returned, otherwise, “-1” is returned.

pbsqstart queue

Set the “started” attribute for the queue given by queue to true.
If this is successful, a “0” is returned, otherwise, “-1” is returned.

pbsqstop queue

Set the “started” attribute for the queue given by queue to false. If this is successful, a “0” is returned, otherwise, “-1” is returned.

pbsalterjob jobid attribute_list

Alter the attributes for a job specified by `jobid`. The parameter `attribute_list` is the list of attributes to be altered. There can be more than one. Each attribute consists of a list of three elements. The first is the name, the second the resource and the third is the new value. If the alter is successful, a “0” is returned, otherwise, “-1” is returned.

`pbsresquery resource_list`

Deprecated. Obtain information about the resources specified by `resource_list`. This will be a list of strings. If the request succeeds, a list with the same number of elements as `resource_list` is returned. Each element in this list will be a list with four numbers. The numbers specify available, allocated, reserved, and down in that order.

`pbsresreserve resource_id resource_list`

Deprecated. Make (or extend) a reservation for the resources specified by `resource_list` which will be given as a list of strings. The parameter `resource_id` is a number which provides a unique identifier for a reservation being tracked by the server. If `resource_id` is given as “0”, a new reservation is created. In this case, a new identifier is generated and returned by the function. If an old identifier is used, that same number will be returned. The Tcl variable “`pbs_errno`” will be set to indicate the success or failure of the reservation.

`pbsresrelease resource_id`

Deprecated. The reservation specified by `resource_id` is released.

The two following commands are not normally used by the scheduler. They are included here because there could be a need for a scheduler to contact a server other than the one which it normally communicates with. Also, these commands are used by the Tcl tools.

`pbsconnect ?server?`

Make a connection to the named server or the default server if a parameter is not given. Only one connection to a server is allowed at any one time.

`pbsdisconnect`

Disconnect from the currently connected server.

The above Tcl functions use PBS interface library calls for communication with the server and the PBS resource monitor library to communicate with pbs_mom.

`datetime ?day? ?time?`

The number of arguments used determine the type of date to be calculated. With no arguments, the current POSIX date is returned. This is an integer in seconds.

With one argument there are two possible formats. The first is a 12 (or more) character string specifying a complete date in the following format:

YYMMDDhhmmss

All characters must be digits. The year (YY) is given by the first two (or more) characters and is the number of years since 1900. The month (MM) is the number of the month [01-12]. The day (DD) is the day of the month [01-32]. The hour (hh) is the hour of the day [00-23]. The minute (mm) is minutes after the hour [00-59]. The second (ss) is seconds after the minute [00-59]. The POSIX date for the given date/time is returned.

The second option with one argument is a relative time. The format for this is

HH:MM:SS

With hours (HH), minutes (MM) and seconds (SS) being separated by colons “:”. The number returned in this case will be the number of seconds in the interval specified, not an absolute POSIX date.

With two arguments a relative date is calculated. The first argument specifies a day of the week and must be one of the following strings: “Sun”, “Mon”, “Tue”, “Wed”, “Thr”, “Fri”, or “Sat”. The second argument is a relative time as given above. The POSIX date calculated will be the day of the week given which follows the current day, and the time given in the second argument. For example, if the current day was Monday, and the two arguments were “Fri” and “04:30:00”, the date calculated would be the POSIX date for the Friday following the current Monday, at four-thirty in the morning. If the day specified and the current day are the same, the current day is used, not the day one week later.

strftime format time

This function calls the POSIX function `strftime()`. It requires two arguments. The first is a format string. The format conventions are the same as those for the POSIX function `strftime()`. The second argument is POSIX calendar time in second as returned by `datetime`. It returns a string based on the format given. This gives the ability to extract information about a time, or format it for printing.

logmsg tag message

This function calls the internal PBS function `log_err()`. It will cause a log message to be written to the scheduler's log file. The tag specifies a function name or other word used to identify the area where the message is generated. The message is the string to be logged.

SEE ALSO

`pbs_tclsh(8B)`, `pbs_wish(8B)`, `pbs_mom(8B)`, `pbs_server(8B)`,
`pbs_sched(8B)`

Chapter 9

Hooks

This chapter describes the PBS hook APIs. For more information on hooks, see the PBS Professional Administrator's Guide.

9.1 Introduction

A hook is a block of Python code that is triggered in response to queueing a job, modifying a job, moving a job, running a job, submitting a PBS reservation, MoM receiving a job, MoM starting a job, MoM killing a job, a job finishing, and MoM cleaning up a job. Each hook can *accept* (allow) or *reject* (prevent) the action that triggers it. The hook can modify the input parameters given for the action. The hook can also make calls to functions external to PBS. PBS provides an interface for use by hooks. This interface allows hooks to read and/or modify things such as job and server attributes, the server, queues, and the event that triggered the hook.

The Administrator creates any desired hooks.

This chapter contains the following man pages:

- `pbs_module(7B)`
- `pbs_stathook(3B)`

See the following additional man pages:

- `qmgr(1B)`
- `qsub(1B)`
- `qmove(1B)`
- `qalter(1B)`
- `pbs_rsub(1B)`
- `pbs_manager(3B)`

9.2 How Hooks Work

9.2.1 Hook Contents and Permissions

A hook contains a Python script. The script is evaluated by a Python 2.5 or later interpreter, embedded in PBS.

Hooks have a default UNIX umask of 022. File permissions are inherited from the current working directory of the hook script.

9.2.2 Accepting and Rejecting Actions

The hook script always accepts the current event request action unless an unhandled exception occurs in the script, a hook alarm timeout is triggered or there's an explicit call to `“pbs.event().reject()”`.

9.2.3 Exceptions

A hook script can catch an exception and evaluate whether or not to accept or reject the event action. In this example, while referencing the non-existent attribute `pbs.event().job.interactive`, an exception is triggered, but the event action is still accepted:

```
...
try:
    e = pbs.event()
    if e.job.interactive:
        e.reject("Interactive jobs not allowed")
except SystemExit:
    pass
except:
    e.accept()
```

9.2.4 Unsupported Interfaces and Uses

Site hooks which read, write, close, or alter `stdin`, `stdout`, or `stderr`, are not supported. Hooks which use any interfaces other than those described are unsupported.

9.3 Interface to Hooks

Two PBS APIs are used with hooks. These are `pbs_manager()` and `pbs_stathook()`. The `pbs` module provides a Python interface to PBS.

9.3.1 The `pbs` Module

Hooks have access to a special module called “`pbs`”, which contains functions that perform PBS-related actions. This module must be explicitly loaded by the hook writer via the call “`import pbs`”.

The *pbs module* provides an interface to PBS and the hook environment. The interface is made up of Python objects, which have attributes and methods. You can operate on these objects using Python code.

9.3.1.1 Description of pbs Module

pbs_module

The interface is made up of Python objects, which have attributes and methods. You can operate on these objects using Python code. For a description of each object, see the PBS Professional Administrator's Guide.

9.3.1.2 pbs Module Objects

`pbs.acl`

Represents a PBS ACL type.

`pbs.args`

Represents a space-separated list of PBS arguments to commands like `qsub`, `qdel`.

`pbs.BadAttributeValueError`

Raised when setting the attribute value of a `pbs.*` object to an invalid value.

`pbs.BadAttributeValueTypeError`

Raised when setting the attribute value of a `pbs.*` object to an invalid value type.

`pbs.BadResourceValueError`

Raised when setting the resource value of a `pbs.*` object to an invalid value.

`pbs.BadResourceValueTypeError`

Raised when setting the resource value of a `pbs.*` object to an invalid value type.

`pbs.checkpoint`

Represents a job's **Checkpoint** attribute.

`pbs.depend`

Represents a job's **depend** attribute.

`pbs.duration`

Represents a time interval.

`pbs.email_list`

Represents the set of users to whom mail may be sent.

`pbs.event`

Represents a PBS event.

- `pbs.EventIncompatibleError`
Raised when referencing a non-existent attribute in `pbs.event()`.
- `pbs.EXECHOST_PERIODIC`
The `exechost_periodic` event type.
- `pbs.EXECJOB_BEGIN`
The `execjob_begin` event type.
- `pbs.EXECJOB_END`
The `execjob_end` event type.
- `pbs.EXECJOB_EPILOGUE`
The `execjob_epilogue` event type.
- `pbs.EXECJOB_PRETERM`
The `execjob_preterm` event type.
- `pbs.EXECJOB_PROLOGUE`
The `execjob_prologue` event type.
- `pbs.exec_host`
Represents a job's `exec_host` attribute.
- `pbs.exec_vnode`
Represents a job's `exec_vnode` attribute.
- `pbs.group_list`
Represents a list of group names.
- `pbs.hold_types`
Represents a job's `Hold_Types` attribute.
- `pbs.job`
Represents a PBS job.
- `pbs.job_sort_formula`
Represents the server's `job_sort_formula` attribute.
- `pbs.JOB_STATE_BEGUN`
Represents the job array state of having started.

`pbs.JOB_STATE_EXITING`

Represents the job state of exiting.

`pbs.JOB_STATE_EXPIRED`

Represents the subjob state of expiring.

`pbs.JOB_STATE_FINISHED`

Represents the job state of finished.

`pbs.JOB_STATE_HELD`

Represents the job state of held.

`pbs.JOB_STATE_MOVED`

Represents the job state of moved.

`pbs.JOB_STATE_QUEUED`

Represents the job state of queued.

`pbs.JOB_STATE_RUNNING`

Represents the job state of running.

`pbs.JOB_STATE_SUSPEND`

Represents the job state of suspended.

`pbs.JOB_STATE_SUSPEND_USERACTIVE`

Represents the job state of suspended due to user activity.

`pbs.JOB_STATE_TRANSIT`

Represents the job state of transiting.

`pbs.JOB_STATE_WAITING`

Represents the job state of waiting.

`pbs.join_path`

Represents a job's **Join_Path** attribute.

`pbs.keep_files`

Represents a job's **Keep_Files** attribute.

`pbs.license_count`

Represents a set of licensing-related counters.

pbs.LOG_DEBUG
Log level 004.

pbs.LOG_ERROR
Log level 004.

pbs.LOG_WARNING
Log level 004.

pbs.mail_points
Represents a job's `Mail_Points` attribute.

pbs.MODIFYJOB
The `modifyjob` event type.

pbs.MOVEJOB
The `movejob` event type.

pbs.ND_BUSY
Represents *busy* vnode state.

pbs.ND_DEFAULT_EXCL
Represents *default_excl* sharing vnode attribute value

pbs.ND_DEFAULT_SHARED
Represents *default_shared* sharing vnode attribute value.

pbs.ND_DOWN
Represents *down* vnode state

pbs.ND_FORCE_EXCL
Represents *force_excl* sharing vnode attribute value.

pbs.ND_FREE
Represents *free* vnode state.

pbs.ND_GLOBUS
PBS no longer supports Globus. The Globus functionality has been removed from PBS.
Represents *globus* value for vnode `ntype` attribute.

pbs.ND_IGNORE_EXCL

Represents *ignore_excl* sharing vnode attribute value.

pbs.ND_JOBBUSY

Represents *job-busy* vnode state.

pbs.ND_JOB_EXCLUSIVE

Represents *job-exclusive* vnode state.

pbs.ND_OFFLINE

Represents *offline* vnode state.

pbs.ND_PBS

Represents *pbs* value for vnode *ntype* attribute.

pbs.ND_PROV

Represents *provisioning* vnode state.

pbs.ND_RESV_EXCLUSIVE

Represents *resv-exclusive* vnode state.

pbs.ND_STALE

Represents *stale* vnode state.

pbs.ND_STATE_UNKNOWN

Represents *state-unknown*, *down* vnode state.

pbs.ND_UNRESOLVABLE

Represents *unresolvable* vnode state.

pbs.ND_WAIT_PROV

Represents *wait-provisioning* vnode state.

pbs.node_group_key

Represents the server or queue *node_group_key* attribute.

pbs.path_list

Represents a list of pathnames.

pbs.place

Represents the place job submission specification.

pbs.QTYPE_EXECUTION

The execution queue type.

pbs.QTYPE_ROUTE
The route queue type.

pbs.queue
Represents a PBS queue.

pbs.QUEUEJOB
The `queuejob` event type.

pbs.range
Represents a range of numbers referring to array indices.

pbs.resv
Represents a PBS reservation.

pbs.RESVSUB
The `resvsub` event type.

pbs.RESV_STATE_BEING_DELETED
Represents the reservation state *RESV_BEING_DELETED*.

pbs.RESV_STATE_CONFIRMED
Represents the reservation state *RESV_CONFIRMED*.

pbs.RESV_STATE_DEGRADED
Represents the reservation state *RESV_DEGRADED*.

pbs.RESV_STATE_DELETED
Represents the reservation state *RESV_DELETED*.

pbs.RESV_STATE_DELETING_JOBS
Represents the reservation state *RESV_DELETING_JOBS*.

pbs.RESV_STATE_FINISHED
Represents the reservation state *RESV_FINISHED*.

pbs.RESV_STATE_NONE
Represents the reservation state *RESV_NONE*.

pbs.RESV_STATE_RUNNING

Represents the reservation state *RESV_RUNNING*.

`pbs.RESV_STATE_TIME_TO_RUN`

Represents the reservation state *RESV_TIME_TO_RUN*.

`pbs.RESV_STATE_UNCONFIRMED`

Represents the reservation state *RESV_UNCONFIRMED*.

`pbs.RESV_STATE_WAIT`

Represents the reservation state *RESV_WAIT*.

`pbs.route_destinations`

Represents a queue's `route_destinations` attribute.

`pbs.RUNJOB`

The `runjob` event type.

`pbs.select`

Represents the `select` job submission specification.

`pbs.server`

Represents the local PBS server.

`pbs.size`

Represents a PBS `size` type.

`pbs.software`

Represents a site-dependent software specification resource.

`pbs.staging_list`

Represents a list of file stagein or stageout parameters.

`pbs.state_count`

Represents a set of job-related state counters.

`pbs.SV_STATE_ACTIVE`

Represents the server state “*Scheduling*”.

`pbs.SV_STATE_HOT`

Represents the server state “*Hot_Start*”.

`pbs.SV_STATE_IDLE`

Represents the server state “*Idle*”.

`pbs.SV_STATE_SHUTDEL`

Represents the server state “*Terminating, Delayed*”.

`pbs.SV_STATE_SHUTIMM`

Represents the server state “*Terminating*”.

`pbs.SV_STATE_SHUTSIG`

Represents the server state “*Terminating*”, when a signal has been caught.

`pbs.UnsetAttributeNameError`

Raised when referencing a non-existent name of a `pbs.*` object.

`pbs.UnsetResourceNameError`

Raised when referencing a non-existent name of a `pbs.*` object.

`pbs.user_list`

Represents a list of user names.

`pbs.vchunk`

Represents a resource chunk assigned to a job.

`pbs.version`

Represents PBS version information.

`pbs.vnode`

Represents a PBS vnode.

`SystemExit`

Raised when accepting or rejecting an action.

9.3.1.3 pbs Module Global Attribute Creation Methods

`pbs.acl("[+|-]<entity>][,...]")`

Creates an object representing a PBS ACL, using the given string parameter. Instantiation of these objects requires a formatted input string.

`pbs.checkpoint("<checkpoint_string>")`

where *<checkpoint_string>* must be one of “n”, “s”, “c”, “c=mmm”, “w”, or “w=mmm”. Creates an object representing the job’s Checkpoint attribute, using the given string. Instantiation of these objects requires a formatted input string.

`pbs.depend("<depend_string>")`
<depend_string> must be of format "*<type>.:<jobid>[,<jobid>...]*", or "*on:<count>*", and where *<type>* is one of "after", "afterok", "afterany", "before", "beforeok", and "beforenotok". Creates a PBS dependency specification object representing the job's `depend` attribute, using the given *<depend_string>*. Instantiation of these objects requires a formatted input string.

`pbs.duration("[[hours:]minutes:]seconds[.milliseconds]")`
 Creates a time specification duration instance, returning the equivalent number of seconds from the given time string. Represents an interval or elapsed time in number of seconds. Duration objects can be specified using either a time or an integer. See the "`pbs.duration(<integer>)`" creation method.

`pbs.duration(<integer>)`
 Creates an integer duration instance using the specified number of seconds. A `pbs.duration` instance can be operated on by any of the Python `int` functions. When performing arithmetic operations on a `pbs.duration` type, ensure the resulting value is a `pbs.duration()` type, before assigning to a job member that expects such a type.

`pbs.email_list("<email_address1>[,<email_address2>...]")`
 Creates an object representing a mail list. Instantiation of these objects requires a formatted input string.

`pbs.exec_host("host/N[*C][+...]")`
 Create an object representing the `exec_host` job attribute, using the given host and resource specification. Instantiation of these objects requires a formatted input string.

`pbs.exec_vnode("<vchunk>[+<vchunk>...]")`
<vchunk> is (*<vnodename:ncpus=N:mem=M>*) Creates an object representing the `exec_vnode` job attribute, using the given vnode and resource specification. When the `qrun -H` command is used, or when the scheduler runs a job, the `pbs.job.exec_vnode` object contains the vnode specification for the job. Instantiation of these objects requires a formatted input string. Example:
`pbs.exec_vnode((vnodeA:ncpus=N:mem=X)+(nodeB:ncpus=P:mem=Y+nodeC:mem=Z))`

This object is managed and accessed via the `str()` or `repr()` functions.

Example:

```
Python> ev = pbs.server().job("10").exec_vnode
Python> str(ev)
(vnodeA:ncpus=2:mem=200m)+(vnodeB:ncpus=5:mem=1g)"
```

```
pbs.group_list("<group_name>[@<host>][,<group_name>[@<host>]...]")
```

Creates an object representing a PBS group list. To use a group list object:

```
pbs.job.group_list = pbs.group_list(...)
```

Instantiation of these objects requires a formatted input string.

```
pbs.hold_types("<hold_type_str>")
```

where *<hold_type_str>* is one of "u", "o", "s", or "n". Creates an object representing the `Hold_Types` job attribute. Instantiation of these objects requires a formatted input string.

```
pbs.job_sort_formula("<formula_string>")
```

where *<formula_string>* is a string containing a math formula. Creates an object representing the `job_sort_formula` server attribute. Instantiation of these objects requires a formatted input string.

```
pbs.join_path({"oe"}"eo"}"n"})
```

Creates an object representing the `Join_Path` job attribute. Instantiation of these objects requires a formatted input string.

```
pbs.keep_files("<keep_files_str>")
```

where *<keep_files_str>* is one of "o", "e", "oe", "eo". Creates an object representing the `Keep_Files` job attribute. Instantiation of these objects requires a formatted input string.

```
pbs.license_count("Avail_Global:<W>Avail_Local:<X>Used:<Y>High_Use:<Z>")
```

Instantiates an object representing a `license_count` attribute. Instantiation of these objects requires a formatted input string.

```
pbs.mail_points("<mail_points_string>")
```

where *<mail_points_string>* is "a", "b", and/or "e", or "n". Creates an object representing a `Mail_Points` attribute. Instantiation of these objects requires a formatted input string.

```
pbs.node_group_key("<resource>")
```

Creates an object representing the resource to be used for node grouping, using the specified resource.


```
pbs.path_list("<path>[@<host>][,<path>@<host>...]")
```

Creates an object representing a PBS pathname list. To use a path list object:

```
pbs.job.Shell_Path_List = pbs.path_list(...)
```

Instantiation of these objects requires a formatted input string.

```
pbs.place("[<arrangement>[:<sharing>][:<group>]")
```

arrangement can be "pack", "scatter", "free", "vscatter"

sharing can be "shared", "excl", "exclhost"

group can be of the form "*group*=<resource>"

[*arrangement*], [*sharing*], and [*group*] can be given in any order or combination.

Creates a place object representing the job's place specification.

Instantiation of these objects requires a formatted input string. Example:

```
pl = pbs.place("pack:excl")
```

```
s = repr(pl) (or s = `pl`)
```

```
letter = pl[0] (assigns 'p' to letter)
```

```
s = s + ":group=host" (append to string)
br pl = pbs.place(s)
(update original pl)
```

```
pbs.range("<start>-<stop>:<step>")
```

Creates a PBS object representing a range of values. Example:

```
pbs.range("1-30:3")
```

Instantiation of these objects requires a formatted input string.

```
pbs.route_destinations("<queue_spec>[,<queue_spec>,...]")
```

where <queue_spec> is *queue_name*[@*server_host*[:*port*]]

Creates an object that represents a **route_destinations** routing

queue attribute. Instantiation of these objects requires a formatted input string.

```
pbs.select("[N:]res=val[:res=val][+[N:]res=val[:res=val]...]")
```

Creates a **select** object representing the job's select specification.

Instantiation of these objects requires a formatted input string. Example:

```
sel = pbs.select("2:ncpus=1:mem=5gb+3:ncpus=2:mem=5gb")
```

```
s = repr(sel) (or s = `sel`)
```

```
letter = s[3] (assigns 'c' to letter)
```

```
s = s + "+5:scratch=10gb" (append to string)
```

```
sel = pbs.select(s) (reset the value of sel)
```

```
pbs.size(<integer>)
```

Creates a PBS **size** object using the given integer value, storing the value as the

number of bytes. Size objects can be specified using either an integer or a string.

See the "**pbs.size**(<integer><suffix>)" creation method.

`pbs.size("<integer><suffix>")`

Creates a PBS `size` object out of the given string specification.

See the PBS Professional Reference Guide for suffix information.

The size of a word is the word size on the execution host. Size objects can be specified using either an integer or a string.

To operate on `pbs.size` instances, use the "+" and "-" operators.

To compare `pbs.size` instances, use the "=", "!", ">", "<", ">=", and "<=" operators. Example: the sizes are normalized to the smaller of the 2 suffixes. In this case, "10gb" becomes "10240mb" and is added to "10mb":

```
sz = pbs.size("10gb")
sz = sz + 10mb
10250mb
```

Example: the following returns True because `SZ` is greater than 100 bytes:

```
if sz > 100:
    gt100 = True
```

`pbs.staging_list("<filespec>[,<filespec>,...]")`

where *<filespec>* is *<execution_path>@<storage_host>:<storage_path>*

Creates an object representing a job file staging parameters list. To use a staging list object:

```
pbs.job.stagein = pbs.staging_list(...)
```

Instantiation of these objects requires a formatted input string.

`pbs.state_count("Transit:<U>Queued:<V>Held:<W>Running:<X>Exiting:<Y>Begun:<Z>")`

Instantiates an object representing a `state_count` attribute.

Instantiation of these objects requires a formatted input string.

`pbs.user_list("<user>[@<host>][,<user>@<host>...]")`

Creates an object representing a PBS user list. To use a user list object:

```
pbs.job.User_List = pbs.user_list(...)
```

Instantiation of these objects requires a formatted input string.

9.3.1.4 Other pbs Module Global Methods

`pbs.args("<args>")`

where *<args>* are space-separated arguments to a command such as

`qsub` or `qdel`. Creates an object representing the arguments to the command. Example:

```
pbs.args("-Wsuppress_email=N -r y")
```

Instantiation of these objects requires a formatted input string.

```
pbs.get_local_nodename()
```

This returns a Python `str` whose value is the name of the local natural vnode. If you want to refer to the vnode object representing the current host, you can pass this vnode name as the key to `pbs.event().vnode_list[]`. For example:

```
Vn = pbs.event().vnode_list[pbs.get_local_nodename()]
```

```
pbs.logjobmsg(job_ID,message)
```

where `job_ID` must be an existing or previously existing job ID and where `message` is an arbitrary string. This puts a custom string in the PBS Server log. The `tracejob` command can be used to print out the job-related messages logged by a hook script. Messages are logged at log event class `pbs.LOG_DEBUG`.

```
pbs.logmsg(log_event_class,message)
```

where `message` is an arbitrary string, and where `log_event_class` can be one of the message log event class constants:

```
pbs.LOG_WARNING
```

```
pbs.LOG_ERROR
```

```
pbs.LOG_DEBUG
```

This puts a custom string in the daemon log.

```
pbs.software("<software_info_string>")
```

Creates an object representing a site-dependent software resource. Instantiation of these objects requires a formatted input string.

```
pbs.version("<pbs_version_string>")
```

Creates an object representing the PBS version string. Instantiation of these objects requires a formatted input string.

9.3.1.5 Attributes and Resources

Hooks can read server, queue, or reservation resources. Hooks can read vnode or job attributes and resources. Hooks can modify:

- The resources requested by a job
- The resources used by a job
- The attributes of a job
- The resource arguments to `pbs_rsub`
- Vnode attributes and resources

Custom and built-in PBS resources are represented in Python dictionary-

ies, where the resource names are the dictionary keys. Built-in resources are listed in the PBS Professional Reference Guide. You reference a resource through a vnode, the Server, the event that triggered the hook, or the current job, for example:

```
pbs.server().resources_available["< resource name>"]
pbs.event().job.Resource_List["< resource name>"]
pbs.event().vnode_list[<vnode
name>].resources_available["<resource name >"]
```

The resource name must be in quotes. Example: Get the number of CPUs:

```
ncpus = Resource_List["ncpus"]
```

An instance R of a job resource can be set as follows:

```
R["<resource name>"] = <resource value>
```

For example:

```
pbs.event().job().Resource_List["mem"] = 8gb
```

9.3.1.6 Exceptions

`pbs.BadAttributeValueError`

Raised when setting the attribute value of a `pbs.*` object to an invalid value.

`pbs.BadAttributeValueTypeError`

Raised when setting the attribute value of a `pbs.*` object to an invalid value type.

`pbs.BadResourceValueError`

Raised when setting the resource value of a `pbs.*` object to an invalid value.

`pbs.BadResourceValueTypeError`

Raised when setting the resource value of a `pbs.*` object to an invalid value type.

`pbs.EventIncompatibleError`

Raised when referencing a non-existent attribute in `pbs.event()`.

`pbs.UnsetAttributeNameError`

Raised when referencing a non-existent name of an attribute.

`pbs.UnsetResourceNameError`

Raised when referencing a non-existent name of a resource.

`SystemExit`

Raised when accepting or rejecting an action.

If a hook encounters an unhandled exception, PBS rejects the corresponding action, and an error message is printed to stderr. A message is printed to the daemon log.

9.3.1.7 See Also

The PBS Professional Administrator's Guide, `pbs_hook_attributes(7B)`, `pbs_resources(7B)`, `qmgr(1B)`

9.3.2 The `pbs_manager()` API

The `pbs_manager()` API is described in ["pbs_manager" on page 42](#). The elements related to hooks are repeated here:

The `pbs_manager()` API contains an `obj_name` called "hook" defined as `MGR_OBJ_HOOK`.

To run, hooks require root privilege on UNIX, and local Administrators privilege on Windows. Hooks run only on the server host.

The `pbs_manager()` API contains the following hook commands, which operate only on hook objects:

MGR_CMD_IMPORT

This command is used for loading the hook script contents into a hook.

MGR_CMD_EXPORT

This command is used for dumping to a file the contents of a hook script.

The parameters to `MGR_CMD_IMPORT` and `MGR_CMD_EXPORT` are specified via the `attrib` parameter of `pbs_manager()`. The `attrib` parameter is a "struct `attrop1`" defined in `pbs_if1.h` as:

```
struct attrop1 {
    char    *name;
    char    *resource;
    char    *value;
    enum batch_op op;
    struct attrop1 *next;
};
```

The `attrib` list is terminated by the first entry where `next` is null.

For `MGR_CMD_IMPORT`, specify attrop1 'name' as “content-type”, “content-encoding”, and “input-file” along with the corresponding 'value' and an 'op' of SET.

For `MGR_CMD_EXPORT`, specify the attrop1 'name' as “content-type”, “content-encoding”, and “output-file” along with the corresponding 'value', and an 'op' of SET.

9.3.2.1 Examples of Using `pbs_manager()`

Example 9-1: The following:

```
# qmgr -c 'import hook hook1 application/x-python base64 hello.py.b64'
```

is programmatically equivalent to:

```
static struct attrop1 imp_attribs[] = {
    { "content-type",
      (char *)0,
      "application/x-python",
      SET,
      (struct attrop1 *)&imp_attribs[1]
    },
    { "content-encoding",
      (char *)0,
      "base64",
      SET,
      (struct attrop1 *)&imp_attribs[2]},
    { "input-file",
      (char *)0,
      "hello.py.b64",
      SET,
      (struct attrop1 *)0
    }
};

pbs_manager(con, MGR_CMD_IMPORT, MGR_OBJ_HOOK, "hook1", &imp_attribs[0],
           NULL);
```

Example 9-2: The following:

```
# qmgr -c 'export hook hook1 application/x-python default hello.py'
```

is programmatically equivalent to:

```
static struct attrop1 exp_attribs[] = {
    { "content-type",
      (char *)0,
      "application/x-python",
      SET,
      (struct attrop1 *)&exp_attribs[1]},
    { "content-encoding",
      (char *)0,
      "default",
      SET,
      (struct attrop1 *)&exp_attribs[2]},
    { "output-file",
      (char *)0,
      "hello.py",
      SET,
      (struct attrop1 *)0
    }
};

pbs_manager(con, MGR_CMD_EXPORT, MGR_OBJ_HOOK, "hook1", &exp_attribs[0],
            NULL);
```

9.3.3 The pbs_stathook() API

The PBS API called “pbs_stathook()” is used to get attributes and values for site hooks.

The prototype for pbs_stathook() is as follows:

```
struct batch_status *pbs_stathook(int connect, char *hook_name, struct
    attr1 *attrib, char *extend)
```

The call to pbs_stathook() causes a PBS_BATCH_StatusHook request to be sent to the server. In reply, the PBS server returns a batch reply status of object type MGR_OBJECT_HOOK listing the attributes and values that were requested relating to a particular hook or all hooks of type HOOK_SITE.

9.3.3.1 Example of Using pbs_stathook()

To list all site hooks using `qmgr`:

```
qmgr -c "list hook"
```

To list all site hooks using the `pbs_stathook()` API:

```
pbs_stathook()
```

The result is the same. For example, if there are two site hooks, `c3` and `c36`:

```
Hook c3
```

```
type = site
enabled = true
event = queuejob, modifyjob
user = pbsadmin
alarm = 30
order = 1
```

```
Hook c36
```

```
type = site
enabled = true
event = resvsub
user = pbsadmin
alarm = 30
order = 1
```

9.3.3.2 Description of pbs_stathook() API

pbs_stathook(3B)

SYNOPSIS

```
#include <pbs_error.h>
#include <pbs_ifl.h>

struct batch_status *pbs_stathook(int connect, char *id,
struct attrl *attrib, char *extend)

void pbs_statfree(struct batch_status *psj)
```

DESCRIPTION

Issue a batch request to obtain the status of a specified site hook or

a set of site hooks at the current server.

A Status Hook batch request is generated and sent to the server over the connection specified by `connect` which is the return value of `pbs_connect()`.

This API can be executed only by root on the local server host.

The parameter, `id`, may be either a hook name or the null string. If `id` specifies a name, the attribute-value list for that hook is returned. If `id` is a null string or a null pointer, the status of all hooks at the current server is returned.

The parameter, `attrib`, is a pointer to an `attrl` structure which is defined in `pbs_ifl.h` as:

```
struct attrl {
    struct attrl *next;
    char        *name;
    char        *resource;
    char        *value;
};
```

The `attrib` list is terminated by the first entry where `next` is a null pointer.

If an `attrib` list is given, then only the attributes in the list are returned by the server. Otherwise, all the attributes of a hook are returned.

The `resource` member is only used if the `name` member is `ATTR_1`, otherwise it should be a pointer to a null string.

The `value` member should always be a pointer to a null string.

The parameter, `extend`, is reserved for implementation defined extensions.

The return value is a pointer to a list of `batch_status` structures or the null pointer if no site hooks can be queried for status. The `batch_status` structure is defined in `pbs_ifl.h` as

```
struct batch_status {
    struct batch_status *next;
    char        *name;
    struct attrl *attrs;
    char        *text;
}
```

It is up to the user to free the structure when no longer needed, by calling `pbs_statfree()`.

SEE ALSO

`pbs_hook_attributes(7B)`, `pbs_connect(3B)`

DIAGNOSTICS

When the batch request generated by the `pbs_stathook()` function has been completed successfully and the status of each site hook has been returned by the batch server, the routine will return a pointer to the list of `batch_status` structures. If no site hooks were available to query or an error occurred, a null pointer is returned. The global integer `pbs_errno` should be examined to determine the cause.

Chapter 10

HPC Basic Profile

This chapter describes using PBS for HPC Basic Profile jobs. For more information on HPC Basic Profile, see [Chapter 7, "Metascheduling Using HPC Basic Profile", on page 573](#) in the PBS Professional Administrator's Guide.

10.1 Introduction

PBS Professional can schedule and manage jobs on one or more HPC Basic Profile Servers using the Grid Forum OGSA HPC Basic Profile web services standard.

For definitions, see section 7.1, "Definitions", on page 573 of the PBS Professional Administrator's Guide.

10.2 How PBS Works With HPC Basic Profile

10.2.1 Information Available From HPCBP Nodes

The only information available about the HPCBP nodes is that which is supplied by the HPC Basic Profile Server to the HPCBP MOM.

10.2.2 Translating Jobs for HPCBP

10.2.2.1 Translating Job Attributes for HPCBP Jobs

The HPCBP MOM converts the job's attributes to equivalent JSDL elements, and passes the resulting JSDL document to the HPC Basic Profile Server.

If the select statement includes HPCBP hostnames, those names are passed using the *<CandidateHosts>* JSDL element.

The aggregate number of CPUs is passed using the *<TotalCPUCount>* JSDL element.

See the following table for a mapping of job attributes to JSDL elements:

Table 10-1: How PBS Job Attributes Are Translated Into JSDL

PBS Attribute/Directive	JSDL element
-l select = arch = <value>	<i><OperatingSystemVersion></i> or <i><OperatingSystemName></i>
-l cput = <time>	<i><TotalCPUTime></i>
-l file = <size>	<i><IndividualDiskSpace></i>
-l select = N:host = <value> Or -l select = N:vnode=<value>	<i><HostName></i>
-l select = N:mem = <size>	<i><TotalPhysicalMemory></i>
-l select = N :ncpus = <value> :N:mpiprocs = <value>	<i><TotalCPUCount></i>
-l pcput = <time>	<i><IndividualCPUTime></i>
-l pmem = <size>	<i><IndividualPhysicalMemory></i>
-l pvmem = <size>	<i><IndividualVirtualMemory></i>
-l vmem = <size>	<i><TotalVirtualMemory></i>
-N Job_Name	<i><JobName></i>
-w stagein = <directive>	<i><DataStaging></i> + <i><Source></i>
-w stageout = <directive>	<i><DataStaging></i> + <i><Target></i>
-o <hostname> : filepath	<i><DataStaging></i> + <i><Target></i>
-e <hostname> : filepath	<i><DataStaging></i> + <i><Target></i>
-j [o][e]	<i><DataStaging></i> + <i><Target></i>
-k [o][e]	----

Table 10-1: How PBS Job Attributes Are Translated Into JSDL

PBS Attribute/Directive	JSDL element
Variable_List	<Environment>
-l ompthreads	<Environment>

10.2.2.2 Translating arch Values for HPCBP Jobs

The following table lists arch values and their *OperatingSystemName* and *OperatingSystemVersion* JSDL equivalents:

Table 10-2: Architectures Used in HPCBP Jobs

arch	OperatingSystemName	OperatingSystemVersion
<i>aix4</i>	<i>AIX</i>	----
<i>hpux10</i>	<i>HPUX</i>	----
<i>hpux11</i>	<i>HPUX</i>	----
<i>irix6</i>	<i>IRIX</i>	----
<i>linux</i>	<i>LINUX</i>	----
<i>solaris7</i>	<i>Solaris</i>	----
<i>Digitalunix</i>	<i>Tru64_UNIX</i>	----
<i>windows_2003_server</i>	<i>other</i>	<i>Windows_2003_server</i>
<i>windows_2008_server</i>	<i>other</i>	<i>Windows_2008_server</i>
<i>other</i>	<i>other</i>	----
<i>macos</i>	<i>MACOS</i>	----
<i>attunix</i>	<i>ATTUNIX</i>	----
<i>dgux</i>	<i>DGUX</i>	----
<i>decnt</i>	<i>DECNT</i>	----

Table 10-2: Architectures Used in HPCBP Jobs

arch	OperatingSystem Name	OperatingSystemVersion
<i>OpenMVS</i>	<i>OpenMVS</i>	----
<i>MVS</i>	<i>MVS</i>	----
<i>OS400</i>	<i>OS400</i>	----
<i>OS_2</i>	<i>OS_2</i>	----
<i>JavaVM</i>	<i>JavaVM</i>	----
<i>WINNT</i>	<i>WINNT</i>	----
<i>WINCE</i>	<i>WINCE</i>	----
<i>NCR3000</i>	<i>NCR3000</i>	----
<i>NetWare</i>	<i>NetWare</i>	----
<i>OSF</i>	<i>OSF</i>	----
<i>DC_OS</i>	<i>DC_OS</i>	----
<i>Reliant_UNIX</i>	<i>Reliant_UNIX</i>	----
<i>SCO_UnixWare</i>	<i>SCO_UnixWare</i>	----
<i>SCO_OpenServer</i>	<i>SCO_OpenServer</i>	----
<i>Sequent</i>	<i>Sequent</i>	----
<i>SunOS</i>	<i>SunOS</i>	----
<i>U6000</i>	<i>U6000</i>	----
<i>ASERIES</i>	<i>ASERIES</i>	----
<i>TandemNSK</i>	<i>TandemNSK</i>	----
<i>TandemNT</i>	<i>TandemNT</i>	----
<i>BS2000</i>	<i>BS2000</i>	----
<i>Lynx</i>	<i>Lynx</i>	----

Table 10-2: Architectures Used in HPCBP Jobs

arch	OperatingSystem Name	OperatingSystemVersion
<i>XENIX</i>	<i>XENIX</i>	----
<i>VM</i>	<i>VM</i>	----
<i>Interactive_UNIX</i>	<i>Interactive_UNIX</i>	----
<i>BSDUNIX</i>	<i>BSDUNIX</i>	----
<i>FreeBSD</i>	<i>FreeBSD</i>	----
<i>NetBSD</i>	<i>NetBSD</i>	----
<i>GNU_Hurd</i>	<i>GNU_Hurd</i>	----
<i>OS9</i>	<i>OS9</i>	----
<i>MACH_Kernel</i>	<i>MACH_Kernel</i>	----
<i>Inferno</i>	<i>Inferno</i>	----
<i>QNX</i>	<i>QNX</i>	----
<i>EPOC</i>	<i>EPOC</i>	----
<i>IxWorks</i>	<i>IxWorks</i>	----
<i>VxWorks</i>	<i>VxWorks</i>	----
<i>MiNT</i>	<i>MiNT</i>	----
<i>BeOS</i>	<i>BeOS</i>	----
<i>HP_MPE</i>	<i>HP_MPE</i>	----
<i>NextStep</i>	<i>NextStep</i>	----
<i>PalmPilot</i>	<i>PalmPilot</i>	----
<i>Rhapsody</i>	<i>Rhapsody</i>	----
<i>Windows_2000</i>	<i>Windows_2000</i>	----
<i>Dedicated</i>	<i>Dedicated</i>	----

Table 10-2: Architectures Used in HPCBP Jobs

arch	OperatingSystem Name	OperatingSystemVersion
<i>OS_390</i>	<i>OS_390</i>	----
<i>VSE</i>	<i>VSE</i>	----
<i>TPF</i>	<i>TPF</i>	----
<i>Windows_R_Me</i>	<i>Windows_R_Me</i>	----
<i>Caldera_Open_UNIX</i>	<i>Caldera_Open_UNI X</i>	----
<i>OpenBSD</i>	<i>OpenBSD</i>	----
<i>Not_Applicable</i>	<i>Not_Applicable</i>	----
<i>Windows_XP</i>	<i>Windows_XP</i>	----
<i>z_OS</i>	<i>z_OS</i>	----
<i><any other string></i>	<i>other</i>	<i><any other string></i>

10.3 Examples

Example 10-1: PBS job request:

```
qsub -N job1 -- /bin/executable -a -b -cd
```

JSDL document:

```
<?xml version="1.0" encoding="utf-8"?>
<jsdsl:JobDefinition xmlns:jsdl="http://schemas.ggf.org/jsdl/2005/11/jsdl"
  xmlns:jsdl-hpcpa="http://schemas.ggf.org/jsdl/2006/07/jsdl-hpcpa"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" >
  <jsdsl:JobDescription>
    <jsdsl:JobIdentification>
      <jsdsl:JobName>job1</jsdl:JobName>
      <jsdsl:JobProject> PBS-TO-HPCBP v1</jsdl:Job-
        Project>
    </jsdl:JobIdentification>
    <jsdsl:Application>
      <jsdsl-hpcpa:HPCProfileApplication name="PBS-
        toHPCBP">
      <jsdsl-hpcpa:Executable>/bin/execu-
        table</jsdl-hpcpa:Executable>
      <jsdsl-hpcpa:Argument>-a</jsdl-hpcpa:Argument>
      <jsdsl-hpcpa:Argument>b</jsdl-hpcpa:Argument>
      <jsdsl-hpcpa:Argument>-cd</jsdl-hpcpa:Argument>
      </jsdl-hpcpa:HPCProfileApplication>
    </jsdl:Application>
  </jsdl:JobDescription>
</jsdl:JobDefinition>
```

Example 10-2: PBS job request:

```
qsub -N job2 -l select=2:ncpus=2:mpiprocs=2 -- /bin/executable a b c
```

JSDL document:

```
<?xml version="1.0" encoding="utf-8"?>
<jsdsl:JobDefinition xmlns:jsdl="http://schemas.ggf.org/jsdl/2005/11/jsdl"
  xmlns:jsdl-hpcpa="http://schemas.ggf.org/jsdl/2006/07/jsdl-hpcpa"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" >
  <jsdsl:JobDescription>
    <jsdsl:JobIdentification>
      <jsdsl:JobName>job2</jsdl:JobName>
      <jsdsl:JobProject> PBS-TO-HPCBP v1</jsdl:Job-
        Project>
```

```

</jsdl:JobIdentification>
<jsdl:Application>
  <jsdl-hpcpa:HPCProfileApplication name="PBS-
    toHPCBP" >
    <jsdl-hpcpa:Executable>/bin/execu-
      table</jsdl-hpcpa:Executable>
    <jsdl-hpcpa:Argument>a</jsdl-hpcpa:Argument>
    <jsdl-hpcpa:Argument>b</jsdl-hpcpa:Argument>
    <jsdl-hpcpa:Argument>c</jsdl-hpcpa:Argument>
  </jsdl-hpcpa:HPCProfileApplication>
</jsdl:Application>
<jsdl:Resources>
  <jsdl:TotalCPUCount>
    <jsdl:UpperBoundedRange>4</jsdl:Upper-
      BoundedRange>
    <jsdl:LowerBoundedRange>4</jsdl:Lower-
      BoundedRange>
  </jsdl:TotalCPUCount>
</jsdl:Resources>
</jsdl:JobDescription>
</jsdl:JobDefinition>

```

Example 10-3: PBS job request:

```

qsub -N job3 -l select= 4 :ncpus=4 :arch=windows_2008_server
:host=winhost0+2:ncpus=2: host=winhost1 -- /bin/executable a b

```

JSDL document:

```

<?xml version="1.0" encoding="utf-8"?>
<jsdl:JobDefinition      xmlns:jsdl="http://schemas.ggf.org/jsdl/2005/11/
  jsdl" xmlns:jsdl-hpcpa="http://schemas.ggf.org/jsdl/2006/07/jsdl-
  hpcpa" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" >
  <jsdl:JobDescription>
    <jsdl:JobIdentification>
      <jsdl:JobName>job3</jsdl:JobName>
      <jsdl:JobProject> PBS-TO-HPCBP v1</jsdl:Job-
        Project>
    </jsdl:JobIdentification>

```

```

<jSDL:Application>
  <jSDL-hpcpa:HPCProfileApplication name="PBSto-
    HPCBP" >
    <jSDL-hpcpa:Executable>/bin/execu-
      table</jSDL-hpcpa:Executable>
    <jSDL-hpcpa:Argument>a</jSDL-hpcpa:Argument>
    <jSDL-hpcpa:Argument>b</jSDL-hpcpa:Argument>
    </jSDL-hpcpa:HPCProfileApplication>
  </jSDL:Application>
<jSDL:Resources>
  <jSDL:CandidateHosts>
    <jSDL:HostName>winhost0</jSDL:HostName>
    <jSDL:HostName>winhost1</jSDL:HostName>
  </jSDL:CandidateHosts>
  <jSDL:OperatingSystem>
    <jSDL:OperatingSystemType>
      <jSDL:OperatingSystemName>oth-
        er</jSDL:OperatingSystemName>
    </jSDL:OperatingSystemType>
    <jSDL:OperatingSystemVersion>Windows_2008_-
      Server</jSDL:OperatingSystemVersion>
  </jSDL:OperatingSystem>
  <jSDL:TotalCPUCount>
    <jSDL:UpperBoundedRange>20</jSDL:Upper-
      BoundedRange>
    <jSDL:LowerBoundedRange>20</jSDL:Lower-
      BoundedRange>
  </jSDL:TotalCPUCount>
</jSDL:Resources>
</jSDL:JobDescription>
</jSDL:JobDefinition>

```

Example 10-4: PBS job request:

```
qsub -N job4 -l select=4:ncpus=2:mpiprocs=2 -- /bin/executable a b
```

JSDL document:

```
<?xml version="1.0" encoding="utf-8"?>
```

```

<jSDL:JobDefinition      xmlns:jSDL="http://schemas.ggf.org/jSDL/2005/11/
  jSDL" xmlns:jSDL-hPCPA="http://schemas.ggf.org/jSDL/2006/07/jSDL-
  hPCPA" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" >
  <jSDL:JobDescription>
    <jSDL:JobIdentification>
      <jSDL:JobName>job4</jSDL:JobName>
      <jSDL:JobProject> PBS-TO-HPCBP v1</jSDL:Job-
        Project>
    </jSDL:JobIdentification>
    <jSDL:Application>
      <jSDL-hPCPA:HPCProfileApplication name="PBSto-
        HPCBP" >
        <jSDL-hPCPA:Executable>/bin/execu-
          table</jSDL-hPCPA:Executable>
        <jSDL-hPCPA:Argument>a</jSDL-hPCPA:Argu-
          ment>
        <jSDL-hPCPA:Argument>b</jSDL-hPCPA:Argu-
          ment>
      </jSDL-hPCPA:HPCProfileApplication>
    </jSDL:Application>
    <jSDL:Resources>
      <jSDL:TotalCPUCount>
        <jSDL:UpperBoundedRange>8</jSDL:UpperBound-
          edRange>
        <jSDL:LowerBoundedRange>8</jSDL:LowerBound-
          edRange>
      </jSDL:TotalCPUCount>
    </jSDL:Resources>
  </jSDL:JobDescription>
</jSDL:JobDefinition>

```

Example 10-5: Job that specifies all the PBS job attributes:

```

qsub -N job5 -l select=4:ncpus=2:mpiprocs=2:arch=windows_2008_server:
  host=winhost0:ompthreads=2 -l cput=00:01:20 -l file=2kb -l
  pcput=00:00:10 -l pmem=30kb -l pvmem=30kb -l vmem=240kb -- /bin/execut-
  able a b

```

JSDL document:

```

<?xml version="1.0" encoding="utf-8"?>
<jsd1:JobDefinition      xmlns:jsdl="http://schemas.ggf.org/jsdl/2005/11/
  jsdl" xmlns:jsdl-hpcpa="http://schemas.ggf.org/jsdl/2006/07/jsdl-
  hpcpa" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" >
  <jsd1:JobDescription>
    <jsd1:JobIdentification>
      <jsd1:JobName>job5</jsdl:JobName>
      <jsd1:JobProject> PBS-TO-HPCBP v1</jsdl:Job-
        Project>
    </jsdl:JobIdentification>
    <jsd1:Application>
      <jsd1-hpcpa:HPCProfileApplication name="PBSto-
        HPCBP" >
        <jsd1-hpcpa:Executable>/bin/execu-
          table</jsdl-hpcpa:Executable>
        <jsd1-hpcpa:Argument>a</jsdl-hpcpa:Argu-
          ment>
        <jsd1-hpcpa:Argument>b</jsdl-hpcpa:Argu-
          ment>
        <jsd1-hpcpa:Environment name="OMP_NUM_-
          THREADS"> 2</jsdl-hpcpa:Environment>
      </jsdl-hpcpa:HPCProfileApplication>
    </jsdl:Application>
    <jsd1:Resources>
      <jsd1:CandidateHosts>
        <jsd1:HostName>winhost0</jsdl:HostName>
      </jsdl:CandidateHosts>
      <jsd1:OperatingSystem>
        <jsd1:OperatingSystemType>
          <jsd1:OperatingSystemName>oth-
            er</jsdl:OperatingSystemName>
        </jsdl:OperatingSystemType>
        <jsd1:OperatingSystemVersion>Windows_2008_-
          Server</jsdl:OperatingSystemVersion>
      </jsdl:OperatingSystem>
      <jsd1:TotalCPUCount>

```

```

        <jsd1:UpperBoundedRange>8</jsdl:UpperBound-
            edRange>
        <jsd1:LowerBoundedRange>8</jsdl:LowerBound-
            edRange>
    </jsdl:TotalCPUCount>
    <jsd1:TotalCPUTime>
        <jsd1:UpperBoundedRange>80.0</jsdl:Upper-
            BoundedRange>
    </jsdl:TotalCPUTime>
    <jsd1:IndividualDiskSpace>
        <jsd1:UpperBoundedRange>2048.0</jsdl:Upper-
            BoundedRange>
    </jsdl:IndividualDiskSpace>
    <jsd1:IndividualCPUTime>
        <jsd1:UpperBoundedRange>10.0</jsdl:Upper-
            BoundedRange>
    </jsdl:IndividualCPUTime>
    <jsd1:IndividualPhysicalMemory>
        <jsd1:UpperBoundedRange>30720.0</jsdl:Up-
            perBoundedRange>
    </jsdl:IndividualPhysicalMemory>
    <jsd1:IndividualVirtualMemory>
        <jsd1:UpperBoundedRange>30720.0</jsdl:Up-
            perBoundedRange>
    </jsdl:IndividualVirtualMemory>
    <jsd1:TotalVirtualMemory>
        <jsd1:UpperBoundedRange>245760.0</jsdl:Up-
            perBoundedRange>
    </jsdl:TotalVirtualMemory>
</jsdl:Resources>
</jsdl:JobDescription>
</jsdl:JobDefinition>

```

Example 10-6: File staging example:

```

qsub -N job6 -Wstagein=test.in:server@/home/test/test.in -Wstage-
out=test.out:server@/home/test/test.out -- /bin/executable a b

```

JSDL document: If the `hpcbp_stage_protocol` attribute is set to *ftp:34* and the user with user-name *User1* has submitted the job, the HPCBP MOM generates the following JSDL document:

```
<?xml version="1.0" encoding="utf-8"?>
<jsdsl:JobDefinition      xmlns:jsdl="http://schemas.ggf.org/jsdl/2005/11/
jsdl" xmlns:jsdl-hpcpa="http://schemas.ggf.org/jsdl/2006/07/jsdl-
hpcpa" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" >
  <jsdsl:JobDescription>
    <jsdsl:JobIdentification>
      <jsdsl:JobName>job6</jsdl:JobName>
      <jsdsl:JobProject> PBS-TO-HPCBP v1</jsdl:Job-
        Project>
    </jsdl:JobIdentification>
    <jsdsl:Application>
      <jsdsl-hpcpa:HPCProfileApplication name="PBSto-
        HPCBP" >
        <jsdsl-hpcpa:Executable>/bin/execu-
          table</jsdl-hpcpa:Executable>
        <jsdsl-hpcpa:Argument>a</jsdl-hpcpa:Argu-
          ment>
        <jsdsl-hpcpa:Argument>b</jsdl-hpcpa:Argu-
          ment>
      </jsdl-hpcpa:HPCProfileApplication>
    </jsdl:Application>
    <jsdsl:DataStaging>
      <jsdsl:FileName>test.in</jsdl:FileName>
      <jsdsl:CreationFlag>overwrite</jsdl:Creation-
        Flag>
      <jsdsl:Source>
        <jsdsl:URI>ftp://ser-
          ver:34/home/test/test.in</jsdl:URI>
      </jsdl:Source>
      <jsdsl:Credential>
        <jsdsl:UsernameToken>
          <jsdsl:Username>User1</jsdl:Username>
          <jsdsl:password>User1</jsdl:password>
```

```

        </jsdl:UsernameToken>
    </jsdl:Credential>
</jsdl:DataStaging>
<jsdl:DataStaging>
    <jsdl:FileName>test.out</jsdl:FileName>
    <jsdl:CreationFlag>overwrite</jsdl:Creation-
        Flag>
    <jsdl:Target>
        <jsdl:URI>ftp://ser-
            ver:34/home/test/test.out</jsdl:URI>
    </jsdl:Target>
</jsdl:Credential>
    <jsdl:UsernameToken>
        <jsdl:Username>User1</jsdl:Username>
        <jsdl:password>User1</jsdl:password>
    </jsdl:UsernameToken>
    </jsdl:Credential>
</jsdl:DataStaging>
</jsdl:JobDescription>
</jsdl:JobDefinition>

```

10.4 Caveats

10.4.1 Unsupported Commands

For a list of unsupported PBS commands, see section 7.8.2, "Unsupported Commands", on page 588 of the PBS Professional Administrator's Guide.

10.5 See Also

10.5.1 PBS Professional Manual Pages

See the following man pages:

- `qsub(1B)`
- `pbs_mom(1B)`

10.5.2 References

1. OGSA High Performance Computing Profile Working Group (OGSA-HPCP-WG) of the Open Grid Forum
<https://forge.gridforum.org/sf/projects/ogsa-hpcp-wg>
The HPC Basic Profile specification is GFD.114:
<http://www.ogf.org/documents/GFD.114.pdf>.
2. OGSA High Performance Computing Profile Working Group (OGSA-HPCP-WG) of the Open Grid Forum
<https://forge.gridforum.org/sf/projects/ogsa-hpcp-wg>
The HPC File Staging Profile Version 1.0:
<http://forge.ogf.org/sf/go/doc15024?nav=1>
3. OGSA Job Submission Description Language Working Group (JSDL - WG) of the Open Grid Forum
http://www.ogf.org/gf/group_info/view.php?group=jsdl-wg
The JSDL HPC Profile Application Extension, Version 1.0 is GFD 111:
<http://www.ogf.org/documents/GFD.111.pdf>
4. OGSA Usage Record Working Group (UR-WG) of the Open Grid Forum
The Usage Record - Format Recommendation is GFD.98
<http://www.ogf.org/documents/GFD.98.pdf>
5. Network Working Group, Uniform Resource Identifier (URI) : Generic Syntax
<http://www.rfc-editor.org/rfc/rfc3986.txt>

Appendix A: License Agreement

CAUTION!

PRIOR TO INSTALLATION OR USE OF THE SOFTWARE YOU MUST CONSENT TO THE FOLLOWING SOFTWARE LICENSE TERMS AND CONDITIONS BY CLICKING THE “I ACCEPT” BUTTON BELOW. YOUR ACCEPTANCE CREATES A BINDING LEGAL AGREEMENT BETWEEN YOU AND ALTAIR. IF YOU DO NOT HAVE THE AUTHORITY TO BIND YOUR ORGANIZATION TO THESE TERMS AND CONDITIONS, YOU MUST CLICK “I DO NOT ACCEPT” AND THEN HAVE AN AUTHORIZED PARTY IN THE ORGANIZATION THAT YOU REPRESENT ACCEPT THESE TERMS.

IF YOU, OR THE ORGANIZATION THAT YOU REPRESENT, HAS A MASTER SOFTWARE LICENSE AGREEMENT (“MASTER SLA”) ON FILE AT THE CORPORATE HEADQUARTERS OF ALTAIR ENGINEERING, INC. (“ALTAIR”), THE MASTER SLA TAKES PRECEDENCE OVER THESE TERMS AND SHALL GOVERN YOUR USE OF THE SOFTWARE.

MODIFICATION(S) OF THESE SOFTWARE LICENSE TERMS IS EXPRESSLY PROHIBITED. ANY ATTEMPTED MODIFICATION(S) WILL BE NONBINDING AND OF NO FORCE OR EFFECT UNLESS EXPRESSLY AGREED TO IN WRITING BY AN AUTHORIZED CORPORATE OFFICER OF ALTAIR. ANY DISPUTE RELATING TO THE VALIDITY OF AN ALLEGED MODIFICATION SHALL BE DETERMINED IN ALTAIR’S SOLE DISCRETION.

Altair Engineering, Inc. - Software License Agreement

THIS SOFTWARE LICENSE AGREEMENT, including any Additional Terms (together with the “Agreement”), shall be effective as of the date of YOUR acceptance of these software license terms and conditions (the “Effective Date”) and is between ALTAIR ENGINEERING, INC., 1820 E. Big Beaver Road, Troy, MI 48083-2031, USA, a Michigan corporation (“Altair”), and YOU, or the organization on whose behalf you have authority to accept these terms (the “Licensee”). Altair and Licensee, intending to be legally bound, hereby agree as follows:

1. DEFINITIONS. In addition to terms defined elsewhere in this Agreement, the following terms shall have the meanings defined below for purposes of this Agreement:

Additional Terms. Additional Terms are those terms and conditions which are determined by an Altair Subsidiary to meet local market conditions.

Documentation. Documentation provided by Altair or its resellers on any media for use with the Products.

Execute. To load Software into a computer's RAM or other primary memory for execution by the computer.

Global Zone: Software is licensed based on three Global Zones: the Americas, Europe and Asia-Pacific. When Licensee has Licensed Workstations located in multiple Global Zones, which are connected to a single License (Network) Server, a premium is applied to the standard Software License pricing for a single Global Zone.

ISV/Independent Software Vendor. A software company providing its products, (“ISV Software”) to Altair's Licensees through the Altair License Management System using Altair License Units.

License Log File. A computer file providing usage information on the Software as gathered by the Software.

License Management System. The license management system (LMS) that accompanies the Software and limits its use in accordance with this Agreement, and which includes a License Log File.

License (Network) Server. A network file server that Licensee owns or leases located on Licensee's premises and identified by machine serial number and/or HostID on the Order Form.

License Units. A parameter used by the LMS to determine usage of the Software permitted under this Agreement at any one time.

Licensed Workstations. Single-user computers located in the same Global Zone(s) that Licensee owns or leases that are connected to the License (Network) Server via local area network or Licensee's private wide-area network.

Maintenance Release. Any release of the Products made generally available by Altair to its Licensees with annual leases, or those with perpetual licenses who have an active maintenance agreement in effect, that corrects programming errors or makes other minor changes to the Software. The fees for maintenance and support services are included in the annual license fee but perpetual licenses require a separate fee.

Order Form. Altair's standard form in either hard copy or electronic format that contains the specific parameters (such as identifying Licensee's contracting office, License Fees, Software, Support, and License (Network) Servers) of the transaction governed by this Agreement.

Products. Products include Altair Software, ISV Software, and/or Suppliers' software; and Documentation related to all of the forgoing.

Proprietary Rights Notices. Patent, copyright, trademark or other proprietary rights notices applied to the Products, packaging or media.

Software. The Altair software identified in the Order Form and any Updates or Maintenance Releases.

Subsidiary. Subsidiary means any partnership, joint venture, corporation or other form of enterprise in which a party possesses, directly or indirectly, an ownership interest of fifty percent (50%) or greater, or managerial or operational control.

Suppliers. Any person, corporation or other legal entity which may provide software or documents which are included in the Software.

Support. The maintenance and support services provided by Altair pursuant to this Agreement.

Templates. Human readable ASCII files containing machine-interpretable commands for use with the Software.

Term. The term of licenses granted under this Agreement. Annual licenses shall have a 12-month term of use unless stated otherwise on the Order Form. Perpetual licenses shall have a term of twenty-five years. Maintenance agreements for perpetual licenses have a 12-month term.

Update. A new version of the Products made generally available by Altair to its Licensees that includes additional features or functionalities but is substantially the same computer code as the existing Products.

2. LICENSE GRANT. Subject to the terms and conditions set forth in this Agreement, Altair hereby grants Licensee, and Licensee hereby accepts, a limited, non-exclusive, non-transferable license to: a) install the Products on the License (Network) Server(s) identified on the Order Form for use only at the sites identified on the Order Form; b) execute the Products on Licensed Workstations in accordance with the LMS for use solely by Licensee's employees, or its onsite Contractors who have agreed to be bound by the terms of this Agreement, for Licensee's internal business use on Licensed Workstations within the Global Zone(s) as iden-

tified on the Order Form and for the term identified on the Order Form; c) make backup copies of the Products, provided that Altair's and its Suppliers' and ISV's Proprietary Rights Notices are reproduced on each such backup copy; d) freely modify and use Templates, and create interfaces to Licensee's proprietary software for internal use only using APIs provided that such modifications shall not be subject to Altair's warranties, indemnities, support or other Altair obligations under this Agreement; and e) copy and distribute Documentation inside Licensee's organization exclusively for use by Licensee's employees and its onsite Contractors who have agreed to be bound by the terms of this Agreement. A copy of the License Log File shall be made available to Altair automatically on no less than a monthly basis. In the event that Licensee uses a third party vendor for information technology (IT) support, the IT company shall be permitted to access the Software only upon its agreement to abide by the terms of this Agreement. Licensee shall indemnify, defend and hold harmless Altair for the actions of its IT vendor(s).

3. RESTRICTIONS ON USE. Notwithstanding the foregoing license grant, Licensee shall not do (or allow others to do) any of the following: a) install, use, copy, modify, merge, or transfer copies of the Products, except as expressly authorized in this Agreement; b) use any back-up copies of the Products for any purpose other than to replace the original copy provided by Altair in the event it is destroyed or damaged; c) disassemble, decompile or “unlock”, reverse translate, reverse engineer, or in any manner decode the Software or ISV Software for any reason; d) sublicense, sell, lend, assign, rent, distribute, publicly display or publicly perform the Products or Licensee's rights under this Agreement; e) allow use outside the Global Zone(s) or User Sites identified on the Order Form; f) allow third parties to access or use the Products such as through a service bureau, wide area network, Internet location or time-sharing arrangement except as expressly provided in Section 2(b); g) remove any Proprietary Rights Notices from the Products; h) disable or circumvent the LMS provided with the Products; or (i) link any software developed, tested or supported by Licensee or third parties to the Products (except for Licensee's own proprietary software solely for Licensee's internal use).

4. OWNERSHIP AND CONFIDENTIALITY. Licensee acknowledges that all applicable rights in patents, copyrights, trademarks, service marks, and trade secrets embodied in the Products are owned by Altair and/or its Suppliers or ISVs. Licensee further acknowledges that the Products, and all copies thereof, are and shall remain the sole and exclusive property of Altair and/or its Suppliers and ISVs. This Agreement is a license and not a sale of the Products. Altair retains all rights in the Products not expressly granted to Licensee herein. Licensee acknowledges that the Products are confidential and constitute valuable assets and trade secrets of Altair and/or its Suppliers and ISVs. Licensee agrees to take the same precautions necessary to protect and maintain the confidentiality of the Products as it does to protect its own information of a confidential nature but in any event, no less than a reasonable degree of care, and shall not disclose or make them available to any person or entity except as expressly provided in this Agreement. Licensee shall promptly notify Altair in the event any unauthorized person obtains access to the Products. If Licensee is required by any governmental

authority or court of law to disclose Altair's or its ISV's or its Suppliers' confidential information, then Licensee shall immediately notify Altair before making such disclosure so that Altair may seek a protective order or other appropriate relief. Licensee's obligations set forth in Section 3 and Section 4 of this Agreement shall survive termination of this Agreement for any reason. Altair's Suppliers and ISVs, as third party beneficiaries, shall be entitled to enforce the terms of this Agreement directly against Licensee as necessary to protect Supplier's intellectual property or other rights.

Altair and its resellers providing support and training to licensed end users of the Products shall keep confidential all Licensee information provided to Altair in order that Altair may provide Support and training to Licensee. Licensee information shall be used only for the purpose of assisting Licensee in its use of the licensed Products. Altair agrees to take the same precautions necessary to protect and maintain the confidentiality of the Licensee information as it does to protect its own information of a confidential nature but in any event, no less than a reasonable degree of care, and shall not disclose or make them available to any person or entity except as expressly provided in this Agreement.

5. MAINTENANCE AND SUPPORT. **Maintenance.** Altair will provide Licensee, at no additional charge for annual licenses and for a maintenance fee for paid-up licenses, with Maintenance Releases and Updates of the Products that are generally released by Altair during the term of the licenses granted under this Agreement, except that this shall not apply to any Term or Renewal Term for which full payment has not been received. Altair does not promise that there will be a certain number of Updates (or any Updates) during a particular year. If there is any question or dispute as to whether a particular release is a Maintenance Release, an Update or a new product, the categorization of the release as determined by Altair shall be final. Licensee agrees to install Maintenance Releases and Updates promptly after receipt from Altair. Maintenance Releases and Updates are subject to this Agreement. Altair shall only be obligated to provide support and maintenance for the most current release of the Software and the most recent prior release. **Support.** Altair will provide support via telephone and email to Licensee at the fees, if any, as listed on the Order Form. If Support has not been procured for any period of time for paid-up licenses, a reinstatement fee shall apply. Support consists of responses to questions from Licensee's personnel related to the use of the then-current and most recent prior release version of the Software. Licensee agrees to provide Altair with sufficient information to resolve technical issues as may be reasonably requested by Altair. Licensee agrees to the best of its abilities to read, comprehend and follow operating instructions and procedures as specified in, but not limited to, Altair's Documentation and other correspondence related to the Software, and to follow procedures and recommendations provided by Altair in an effort to correct problems. Licensee also agrees to notify Altair of a programming error, malfunction and other problems in accordance with Altair's then current problem reporting procedure. If Altair believes that a problem reported by Licensee may not be due to an error in the Software, Altair will so notify Licensee. Questions must be directed to Altair's specially designated telephone support numbers and email addresses. Support will also be available via email at Internet addresses designated by Altair. Support is available

Monday through Friday (excluding holidays) from 8:00 a.m. to 5:00 p.m. local time in the Global Zone where licensed, unless stated otherwise on the Order Form. **Exclusions.** Altair shall have no obligation to maintain or support (a) altered, damaged or Licensee-modified Software, or any portion of the Software incorporated with or into other software not provided by Altair; (b) any version of the Software other than the current version of the Software or the immediately prior release of the Software; (c) problems caused by Licensee's negligence, abuse or misapplication of Software other than as specified in the Documentation, or other causes beyond the reasonable control of Altair; or (d) Software installed on any hardware, operating system version or network environment that is not supported by Altair. Support also **excludes** configuration of hardware, non- Altair Software, and networking services; consulting services; general solution provider related services; and general computer system maintenance.

6. WARRANTY AND DISCLAIMER. Altair warrants for a period of ninety (90) days after Licensee initially receives the Software that the Software will perform under normal use substantially as described in then current Documentation. Supplier software included in the Software and ISV Software provided to Licensee shall be warranted as stated by the Supplier or the ISV. Copies of the Suppliers' and ISV's terms and conditions of warranty are available on the Altair Support website. Support services shall be provided in a workmanlike and professional manner, in accordance with the prevailing standard of care for consulting support engineers at the time and place the services are performed.

ALTAIR DOES NOT REPRESENT OR WARRANT THAT THE PRODUCTS WILL MEET LICENSEE'S REQUIREMENTS OR THAT THEIR OPERATION WILL BE UNINTERRUPTED OR ERROR-FREE, OR THAT IT WILL BE COMPATIBLE WITH ANY PARTICULAR HARDWARE OR SOFTWARE. ALTAIR EXCLUDES AND DISCLAIMS ALL EXPRESS AND IMPLIED WARRANTIES NOT STATED HEREIN, INCLUDING THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NON-INFRINGEMENT. THE ENTIRE RISK FOR THE PERFORMANCE, NON-PERFORMANCE OR RESULTS OBTAINED FROM USE OF THE PRODUCTS RESTS WITH LICENSEE AND NOT ALTAIR. ALTAIR MAKES NO WARRANTIES WITH RESPECT TO THE ACCURACY, COMPLETENESS, FUNCTIONALITY, SAFETY, PERFORMANCE, OR ANY OTHER ASPECT OF ANY DESIGN, PROTOTYPE OR FINAL PRODUCT DEVELOPED BY LICENSEE USING THE PRODUCTS.

7. INDEMNITY. Altair will defend and indemnify, at its expense, any claim made against Licensee based on an allegation that the Software infringes a patent or copyright ("Claim"); provided, however, that this indemnification does not include claims which are based on Supplier software or ISV software, and that Licensee has not materially breached the terms of this Agreement, Licensee notifies Altair in writing within ten (10) days after Licensee first learns of the Claim; and Licensee cooperates fully in the defense of the claim. Altair shall have sole control over such defense; provided, however, that it may not enter into any settlement bind-

ing upon Licensee without Licensee's consent, which shall not be unreasonably withheld. If a Claim is made, Altair may modify the Software to avoid the alleged infringement, provided however, that such modifications do not materially diminish the Software's functionality. If such modifications are not commercially reasonable or technically possible, Altair may terminate this Agreement and refund to Licensee the prorated license fee that Licensee paid for the then current Term. Perpetual licenses shall be pro-rated over a 36-month term. Altair shall have no obligation under this Section 7, however, if the alleged infringement arises from Altair's compliance with specifications or instructions prescribed by Licensee, modification of the Software by Licensee, use of the Software in combination with other software not provided by Altair and which use is not specifically described in the Documentation, and if Licensee is not using the most current version of the Software, if such alleged infringement would not have occurred except for such exclusions listed here. This section 7 states Altair's entire liability to Licensee in the event a Claim is made. No indemnification is made for Supplier and/or ISV Software.

8. LIMITATION OF REMEDIES AND LIABILITY. Licensee's exclusive remedy (and Altair's sole liability) for Software that does not meet the warranty set forth in Section 6 shall be, at Altair's option, either (i) to correct the nonconforming Software within a reasonable time so that it conforms to the warranty; or (ii) to terminate this Agreement and refund to Licensee the license fees that Licensee has paid for the then current Term for the nonconforming Software; provided, however that Licensee notifies Altair of the problem in writing within the applicable Warranty Period when the problem first occurs. Any corrected Software shall be warranted in accordance with Section 6 for ninety (90) days after delivery to Licensee. The warranties hereunder are void if the Software has been improperly installed, misused, or if Licensee has violated the terms of this Agreement.

Altair's entire liability for all claims arising under or related in any way to this Agreement (regardless of legal theory), shall be limited to direct damages, and shall not exceed, in the aggregate for all claims, the license and maintenance fees paid under this Agreement by Licensee in the 12 months prior to the claim on a prorated basis, except for claims under Section 7. **ALTAIR AND ITS SUPPLIERS AND ISVS SHALL NOT BE LIABLE TO LICENSEE OR ANYONE ELSE FOR INDIRECT, SPECIAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES ARISING HEREUNDER (INCLUDING LOSS OF PROFITS OR DATA, DEFECTS IN DESIGN OR PRODUCTS CREATED USING THE SOFTWARE, OR ANY INJURY OR DAMAGE RESULTING FROM SUCH DEFECTS, SUFFERED BY LICENSEE OR ANY THIRD PARTY) EVEN IF ALTAIR OR ITS SUPPLIERS OR ITS ISVS HAVE BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.** Licensee acknowledges that it is solely responsible for the adequacy and accuracy of the input of data, including the output generated from such data, and agrees to defend, indemnify, and hold harmless Altair and its Suppliers and ISVs from any and all claims, including reasonable attorney's fees, resulting from, or in connection with Licensee's use of the Software. No

action, regardless of form, arising out of the transactions under this Agreement may be brought by either party against the other more than two (2) years after the cause of action has accrued, except for actions related to unpaid fees.

9. UNITED STATES GOVERNMENT RESTRICTED RIGHTS. This section applies to all acquisitions of the Products by or for the United States government. By accepting delivery of the Products except as provided below, the government or the party procuring the Products under government funding, hereby agrees that the Products qualify as “commercial” computer software as that term is used in the acquisition regulations applicable to this procurement and that the government's use and disclosure of the Products is controlled by the terms and conditions of this Agreement to the maximum extent possible. This Agreement supersedes any contrary terms or conditions in any statement of work, contract, or other document that are not required by statute or regulation. If any provision of this Agreement is unacceptable to the government, Vendor may be contacted at Altair Engineering, Inc., 1820 E. Big Beaver Road, Troy, MI 48083-2031; telephone (248) 614-2400. If any provision of this Agreement violates applicable federal law or does not meet the government's actual, minimum needs, the government agrees to return the Products for a full refund.

For procurements governed by DFARS Part 227.72 (OCT 1998), the Software, except as described below, is provided with only those rights specified in this Agreement in accordance with the Rights in Commercial Computer Software or Commercial Computer Software Documentation policy at DFARS 227.7202-3(a) (OCT 1998). For procurements other than for the Department of Defense, use, reproduction, or disclosure of the Software is subject to the restrictions set forth in this Agreement and in the Commercial Computer Software - Restricted Rights FAR clause 52.227-19 (June 1987) and any restrictions in successor regulations thereto.

Portions of Altair's PBS Professional Software and Documentation are provided with RESTRICTED RIGHTS. Use, duplication, or disclosure by the Government is subject to restrictions as set forth in subdivision(c)(1)(ii) of the rights in the Technical Data and Computer Software clause in DFARS 252.227-7013, or in subdivision (c)(1) and (2) of the Commercial Computer Software-Restricted Rights clause at 48 CFR52.227-19, as applicable.

10. CHOICE OF LAW AND VENUE. This Agreement shall be governed by and construed under the laws of the state of Michigan, without regard to that state's conflict of laws principles except if the state of Michigan adopts the Uniform Computer Information Transactions Act drafted by the National Conference of Commissioners of Uniform State Laws as revised or amended as of June 30, 2002 (“UCITA”) which is specifically excluded. This Agreement shall not be governed by the United Nations Convention on Contracts for the International Sale of Goods, the application of which is expressly excluded. Each Party waives its right to a jury trial in the event of any dispute arising under or relating to this Agreement. Each party agrees that money damages may not be an adequate remedy for breach of the provisions of

this Agreement, and in the event of such breach, the aggrieved party shall be entitled to seek specific performance and/or injunctive relief (without posting a bond or other security) in order to enforce or prevent any violation of this Agreement.

11. [RESERVED]

12. Notice. All notices given by one party to the other under the Agreement or these Additional Terms shall be sent by certified mail, return receipt requested, or by overnight courier, to the respective addresses set forth in this Agreement or to such other address either party has specified in writing to the other. All notices shall be deemed given upon actual receipt.

Written notice shall be made to:

Altair: Licensee Name & Address:

Altair Engineering, Inc. _____

1820 E. Big Beaver Rd _____

Troy, MI 48083 _____

Attn: Tom M. Perring Attn: _____

13. TERM. For annual licenses, or Support provided for perpetual licenses, renewal shall be automatic for each successive year (“Renewal Term”), upon mutual written execution of a new Order Form. All charges and fees for each Renewal Term shall be set forth in the Order Form executed for each Renewal Term. All Software licenses procured by Licensee may be made coterminous at the written request of Licensee and the consent of Altair.

14. TERMINATION. Either party may terminate this Agreement upon thirty (30) days prior written notice upon the occurrence of a default or material breach by the other party of its obligations under this Agreement (except for a breach by Altair of the warranty set forth in Section 8 for which a remedy is provided under Section 10; or a breach by Licensee of Section 5 or Section 6 for which no cure period is provided and Altair may terminate this Agreement immediately) if such default or breach continues for more than thirty (30) days after receipt of such notice. Upon termination of this Agreement, Licensee must cease using the Software and, at Altair's option, return all copies to Altair, or certify it has destroyed all such copies of the Software and Documentation.

15. GENERAL PROVISIONS. Export Controls. Licensee acknowledges that the Products may be subject to the export control laws and regulations of the United States and other countries, and any amendments thereof. Licensee agrees that Licensee will not directly or indirectly export the Products into any country or use the Products in any manner except in compliance with all applicable U.S. and other countries export laws and regulations. **Notice.** All notices given by one party to the other under this Agreement shall be sent by certified mail, return receipt requested, or by overnight courier, to the respective addresses set forth in this Agreement or to such other address either party has specified in writing to the other. All

notices shall be deemed given upon actual receipt. **Assignment.** Neither party shall assign this Agreement without the prior written consent of other party, which shall not be unreasonably withheld. All terms and conditions of this Agreement shall be binding upon and inure to the benefit of the parties hereto and their respective successors and permitted assigns. **Waiver.** The failure of a party to enforce at any time any of the provisions of this Agreement shall not be construed to be a waiver of the right of the party thereafter to enforce any such provisions. **Severability.** If any provision of this Agreement is found void and unenforceable, such provision shall be interpreted so as to best accomplish the intent of the parties within the limits of applicable law, and all remaining provisions shall continue to be valid and enforceable. **Headings.** The section headings contained in this Agreement are for convenience only and shall not be of any effect in constructing the meanings of the Sections. **Modification.** No change or modification of this Agreement will be valid unless it is in writing and is signed by a duly authorized representative of each party. **Conflict.** In the event of any conflict between the terms of this Agreement and any terms and conditions on a Licensee Purchase Order or comparable document, the terms of this Agreement shall prevail. Moreover, each party agrees any additional terms on any Purchase Order or comparable document other than the transaction items of (a) item(s) ordered; (b) pricing; (c) quantity; (d) delivery instructions and (e) invoicing directions, are not binding on the parties. In the event of a conflict between the terms of this Agreement, and the Additional Terms, the Agreement shall take precedence. **Entire Agreement.** This Agreement, the Additional Terms, and the Order Form(s) attached hereto constitute the entire understanding between the parties related to the subject matter hereto, and supersedes all proposals or prior agreements, whether written or oral, and all other communications between the parties with respect to such subject matter. This Agreement may be executed in one or more counterparts, all of which together shall constitute one and the same instrument. **Execution.** Copies of this Agreement executed via original signatures, facsimile or email shall be deemed binding on the parties.

Index

A

activereq [94](#)
addreq [94](#)
allreq [94](#)

B

batch [11](#)

C

client commands [5](#)
closerm [94](#)
commands [5](#)
configrm [94](#)
Cray
 SV1 [vii](#)
 T3e [vii](#)
credential [29](#)

D

downrm [94](#)

E

Executor [5](#)

F

flushreq [94](#)

fullresp [94](#)

G

getreq [94](#)

J

Job
 Executor (MOM) [5](#)
 Scheduler [5](#)
 Server [4](#)

M

manager [5](#)
 commands [5](#)
MOM [5](#), [5](#)
monitoring [3](#)

N

NASA [vii](#)
NQS [3](#)

O

openrm [94](#)
operator
 commands [5](#)

Index

P

[pbs_alterjob 31](#)
[pbs_asyruncjob 52](#)
[PBS_BATCH_AsyrunJob 12](#)
[PBS_BATCH_AuthenUser 11](#)
[PBS_BATCH_Commit 11](#)
[PBS_BATCH_Connect 11](#)
[PBS_BATCH_CopyFiles 11](#)
[PBS_BATCH_CopyFiles_Cred 11](#)
[PBS_BATCH_DeleteJob 11](#)
[PBS_BATCH_DeleteResv 12](#)
[PBS_BATCH_DelFiles 11](#)
[PBS_BATCH_DelFiles_Cred 11](#)
[PBS_BATCH_Disconnect 11](#)
[PBS_BATCH_FailOver 11](#)
[PBS_BATCH_GSS_Context 11](#)
[PBS_BATCH_HoldJob 11](#)
[PBS_BATCH_JobCred 11](#)
[PBS_BATCH_JobObit 11](#)
[PBS_BATCH_jobscript 11](#)
[PBS_BATCH_LocateJob 11](#)
[PBS_BATCH_Manager 11](#)
[PBS_BATCH_MessJob 11](#)
[PBS_BATCH_ModifyJob 11](#)
[PBS_BATCH_MoveJob 11](#)
[PBS_BATCH_MvJobFile 11](#)
[PBS_BATCH_OrderJob 11](#)
[PBS_BATCH_QueueJob 11](#)
[PBS_BATCH_RdytoCommit 11](#)
[PBS_BATCH_RegistDep 11](#)
[PBS_BATCH_ReleaseJob 11](#)
[PBS_BATCH_ReleaseResc 11](#)
[PBS_BATCH_Rerun 11](#)
[PBS_BATCH_Rescq 11](#)
[PBS_BATCH_ReserveResc 11](#)
[PBS_BATCH_RunJob 11](#)
[PBS_BATCH_SelectJobs 11](#)
[PBS_BATCH_SelStat 11](#)
[PBS_BATCH_Shutdown 11](#)
[PBS_BATCH_SignalJob 11](#)
[PBS_BATCH_StageIn 11](#)
[PBS_BATCH_StatusJob 11](#)
[PBS_BATCH_StatusNode 11](#)
[PBS_BATCH_StatusQue 11](#)
[PBS_BATCH_StatusResv 12](#)
[PBS_BATCH_StatusSvr 11](#)
[PBS_BATCH_SubmitResv 11](#)
[PBS_BATCH_TrackJob 12](#)
[pbs_connect 33](#)
[pbs_default 35](#)
[pbs_deljob 36](#)
[pbs_delresv 37](#)
[pbs_disconnect 38](#)
[pbs_geterrmsg 39](#)
[pbs_holdjob 40](#)
[pbs_locjob 41](#)
[pbs_manager 42](#)
[pbs_module 108](#)
[pbs_mom 5](#)
[pbs_movejob 45](#)
[pbs_msgjob 46](#)
[pbs_orderjob 47](#)
[pbs_reruncjob 48](#)
[pbs_rescrelease 49](#)
[pbs_resreserve 49](#)
[pbs_rlsjob 51](#)
[pbs_runjob 52](#)
[pbs_sched 5](#)
[pbs_selectjob 54](#)
[pbs_selstat 56](#)
[pbs_server 4](#)
[pbs_sigjob 59](#)
[pbs_stagein 60](#)
[pbs_statfree 61](#)
[pbs_stathook\(3B\) 126](#)
[pbs_stathost 64](#)
[pbs_statjob 61, 62](#)
[pbs_statnode 64](#)
[pbs_statque 66](#)
[pbs_statresv 68](#)
[pbs_statsched 70](#)
[pbs_statsserver 72](#)
[pbs_statvnode 64](#)
[pbs_submit 74](#)

Index

pbs_submit_resv [76](#)
pbs_tclapi [98](#)
pbs_tclsh [97](#)
pbs_terminate [78](#)
pbs_wish [97](#)
POSIX [5](#)

Q

queuing [3](#)
Quick Start Guide [ix](#)

R

rpp_bind [80](#)
rpp_close [80](#)
rpp_eom [80](#)
rpp_flush [80](#)
rpp_getaddr [80](#)
rpp_getc [80](#)
rpp_io [80](#)
rpp_open [79](#), [80](#)
rpp_poll [80](#)
rpp_putc [80](#)
rpp_rcommit [80](#)
rpp_read [80](#)
rpp_shutdown [80](#)
rpp_terminate [80](#)
rpp_wcommit [80](#)
rpp_write [80](#)

S

Scheduler [5](#)
scheduling [3](#)
Server [4](#)
system daemons [4](#)

T

TCL [97](#)
tm_atnode [86](#)
tm_attach [86](#)
tm_finalize [86](#)
tm_init [86](#)

tm_kill [86](#)
tm_nodeinfo [86](#)
tm_notify [86](#)
tm_obit [86](#)
tm_poll [86](#)
tm_publish [86](#)
tm_rescinfo [86](#)
tm_spawn [86](#)
tm_subscribe [86](#)
tm_taskinfo [86](#)

U

user
 commands [4](#), [5](#)
User Guide [ix](#)

W

workload management [3](#)

Index

PBS Professional® 12.0

User's Guide



PBS Works™

PBS Works is a division of  Altair

PBS Professional 12 User's Guide, updated 1/25/13.

Copyright © 2003-2012 Altair Engineering, Inc. All rights reserved.

PBS™, PBS Works™, PBS GridWorks®, PBS Professional®, PBS Analytics™, PBS Catalyst™, e-Compute™, and e-Render™ are trademarks of Altair Engineering, Inc. and are protected under U.S. and international laws and treaties. All other marks are the property of their respective owners.

ALTAIR ENGINEERING INC. Proprietary and Confidential. Contains Trade Secret Information. Not for use or disclosure outside ALTAIR and its licensed clients. Information contained herein shall not be decompiled, disassembled, duplicated or disclosed in whole or in part for any purpose. Usage of the software is only as explicitly permitted in the end user software license agreement.

Copyright notice does not imply publication.

For documentation and the PBS Works forums, go to:

Web: www.pbsworks.com

For more information, contact Altair at:

Email: pbssales@altair.com

Technical Support

Location	Telephone	e-mail
North America	+1 248 614 2425	pbssupport@altair.com
China	+86 (0)21 6117 1666	es@altair.com.cn
France	+33 (0)1 4133 0992	francesupport@altair.com
Germany	+49 (0)7031 6208 22	hwsupport@altair.de
India	+91 80 66 29 4500	pbs-support@india.altair.com
Italy	+39 0832 315573 +39 800 905595	support@altairengineering.it
Japan	+81 3 5396 2881	pbs@altairjp.co.jp
Korea	+82 31 728 8600	support@altair.co.kr
Scandinavia	+46 (0)46 286 2050	support@altair.se
UK	+44 (0)1926 468 600	pbssupport@uk.altair.com

This document is proprietary information of Altair Engineering, Inc.

Table of Contents

Acknowledgements	ix
About PBS Documentation	xi
1 Concepts and Components	1
1.1 PBS Components	2
2 Getting Started With PBS	5
2.1 New Features in PBS 12.0	5
2.2 New Features in PBS Professional 11.3	5
2.3 New Features in PBS Professional 11.2	6
2.4 New Features in PBS Professional 11.1	6
2.5 New Features in PBS Professional 11.0	6
2.6 New Features in PBS Professional 10.4	7
2.7 New Features in PBS Professional 10.2	7
2.8 New Features in Version 10.1	8
2.9 New Features in Recent Releases	8
2.10 Deprecations	9
2.11 Backward Compatibility	9
2.12 Using PBS	10
2.13 PBS Interfaces	11
2.14 User's PBS Environment	12
2.15 Usernames Under PBS	12
2.16 Setting Up Your UNIX/Linux Environment	13
2.17 Setting Up Your Windows Environment	14
2.18 Environment Variables	17
2.19 Temporary Scratch Space: TMPDIR	18

Table of Contents

3	Submitting a PBS Job	21
3.1	Vnodes: Virtual Nodes	21
3.2	PBS Resources	22
3.3	PBS Jobs	25
3.4	Submitting a PBS Job	29
3.5	Requesting Resources	32
3.6	Placing Jobs on Vnodes	43
3.7	Submitting Jobs Using Select & Place: Examples	48
3.8	Backward Compatibility	54
3.9	How PBS Parses a Job Script	57
3.10	A Sample PBS Job	57
3.11	Changing the Job's PBS Directive	59
3.12	Windows Jobs	60
3.13	Job Submission Options	62
3.14	Failed Jobs	79
4	Multiprocessor Jobs	81
4.1	Submitting Multiprocessor Jobs	81
4.2	Using MPI with PBS	88
4.3	Using PVM with PBS	126
4.4	Using OpenMP with PBS	127
4.5	Hybrid MPI-OpenMP Jobs	129
5	Using the xpbs GUI	133
5.1	Using the xpbs command	133
5.2	Using xpbs: Definitions of Terms	134
5.3	Introducing the xpbs Main Display	135
5.4	Setting xpbs Preferences	143
5.5	Relationship Between PBS and xpbs	144
5.6	How to Submit a Job Using xpbs	145
5.7	Exiting xpbs	148
5.8	The xpbs Configuration File	149
5.9	xpbs Preferences	149

Table of Contents

6	Working with PBS Jobs	153
6.1	Modifying Job Attributes	153
6.2	Holding and Releasing Jobs	156
6.3	Deleting Jobs	159
6.4	Sending Messages to Jobs	159
6.5	Sending Signals to Jobs	160
6.6	Changing Order of Jobs	161
6.7	Moving Jobs Between Queues	163
6.8	Converting a Job into a Reservation Job	164
6.9	Using Job History Information	164
7	Checking Job / System Status	169
7.1	The <code>qstat</code> Command	169
7.2	Viewing Job / System Status with <code>xpbs</code>	187
7.3	The <code>qselect</code> Command	187
7.4	Selecting Jobs Using <code>xpbs</code>	188
7.5	Using <code>xpbs</code> TrackJob Feature	189
7.6	Job Comments for Problem Jobs	190
8	Advanced PBS Features	193
8.1	UNIX Job Exit Status	193
8.2	Changing UNIX Job <code>umask</code>	194
8.3	Requesting <code>qsub</code> Wait for Job Completion	194
8.4	Specifying Job Dependencies	195
8.5	Delivery of Output Files	198
8.6	Input/Output File Staging	198
8.7	Advance and Standing Reservation of Resources	209
8.8	Dedicated Time	225
8.9	Using Comprehensive System Accounting	226
8.10	Running PBS in a UNIX DCE Environment	227
8.11	Running PBS in a UNIX Kerberos Environment	228
8.12	Support for Large Page Mode on AIX	228
8.13	Checking License Availability	229
8.14	Adjusting Job Running Time	229

Table of Contents

9	Job Arrays	235
9.1	Definitions	235
9.2	qsub: Submitting a Job Array	237
9.3	Job Array Attributes	238
9.4	Job Array States	238
9.5	PBS Environmental Variables	239
9.6	File Staging	239
9.7	PBS Commands	243
9.8	Other PBS Commands Supported for Job Arrays	251
9.9	Job Arrays and xpbs	251
9.10	More on Job Arrays	251
9.11	Job Array Caveats	254
10	HPC Basic Profile Jobs	255
10.1	Definitions	255
10.2	How HPC Basic Profile Jobs Work	256
10.3	Environmental Requirements for HPCBP	256
10.4	Submitting HPC Basic Profile Jobs	257
10.5	Managing HPCBP Jobs	262
10.6	Errors, Logging and Troubleshooting	263
10.7	Advice and Caveats	269
10.8	See Also	270
11	Submitting Cray Jobs	273
11.1	Introduction	273
11.2	PBS Jobs on the Cray	273
11.3	PBS Resources for the Cray	274
11.4	Rules for Submitting Jobs on the Cray	282
11.5	Techniques for Submitting Cray Jobs	284
11.6	Viewing Cray Job Information	290
11.7	Caveats and Advice	294
11.8	Errors and Logging	298
12	Using Provisioning	301
12.1	Definitions	301
12.2	How Provisioning Works	301
12.3	Requirements and Restrictions	303
12.4	Using Provisioning	305
12.5	Caveats and Errors	306

Table of Contents

Appendix A: Converting NQS to PBS	309
13.1 Converting Date Specifications.	309
Appendix B: License Agreement	311
Index	321

Table of Contents

Acknowledgements

PBS Professional is the enhanced commercial version of the PBS software originally developed for NASA. The NASA version had a number of corporate and individual contributors over the years, for which the PBS developers and PBS community is most grateful. Below we provide formal legal acknowledgements to corporate and government entities, then special thanks to individuals.

The NASA version of PBS contained software developed by NASA Ames Research Center, Lawrence Livermore National Laboratory, and MRJ Technology Solutions. In addition, it included software developed by the NetBSD Foundation, Inc., and its contributors as well as software developed by the University of California, Berkeley and its contributors.

Other contributors to the NASA version of PBS include Bruce Kelly and Clark Streeter of NERSC; Kent Crispin and Terry Heidelberg of LLNL; John Kochmar and Rob Pennington of *Pittsburgh Supercomputing Center*; and Dirk Grunwald of *University of Colorado, Boulder*. The ports of PBS to the Cray T3e and the IBM SP SMP were funded by *DoD USAERDC*; the port of PBS to the Cray SV1 was funded by *DoD MSIC*.

No list of acknowledgements for PBS would possibly be complete without special recognition of the first two beta test sites. Thomas Milliman of the *Space Sciences Center* of the *University of New Hampshire* was the first beta tester. Wendy Lin of *Purdue University* was the second beta tester and holds the honor of submitting more problem reports than anyone else outside of NASA.

About PBS Documentation

Where to Keep the Documentation

To make cross-references work, put all of the PBS guides in the same directory.

What is PBS Professional?

PBS is a workload management system that provides a unified batch queuing and job management interface to a set of computing resources.

The PBS Professional Documentation

The documentation for PBS Professional includes the following:

PBS Professional Administrator's Guide:

Provides the PBS administrator with the information required to configure and manage PBS Professional (PBS).

PBS Professional Quick Start Guide:

Provides a quick overview of PBS Professional installation and license file generation.

PBS Professional Installation & Upgrade Guide:

Contains information on installing and upgrading PBS Professional.

PBS Professional User's Guide:

Covers user commands and how to submit, monitor, track, delete, and manipulate jobs.

PBS Professional Programmer's Guide:

Discusses the PBS application programming interface (API).

PBS Professional Reference Guide:

Contains PBS reference material.

PBS Manual Pages:

Describe PBS commands, resources, attributes, APIs

Ordering Software and Publications

To order additional copies of this manual and other PBS publications, or to purchase additional software licenses, contact your Altair sales representative. Contact information is included on the copyright page of this book.

Document Conventions

PBS documentation uses the following typographic conventions:

abbreviation

The shortest acceptable abbreviation of a command or subcommand is underlined.

`command`

Commands such as `qmgr` and `scp`

input

Command-line instructions

`manpage (x)`

File and path names. Manual page references include the section number in parentheses appended to the manual page name.

formats

Formats

Attributes

Attributes, parameters, objects, variable names, resources, types

Values

Keywords, instances, states, values, labels

Definitions

Terms being defined

Output

Output or example code

File contents

Chapter 1

Concepts and Components

PBS is a distributed workload management system. As such, PBS handles the management and monitoring of the computational workload on a set of one or more computers. Modern workload management solutions like PBS Professional include the features of traditional batch queueing but offer greater flexibility and control than first generation batch systems (such as NQS).

Workload management systems have three primary roles:

Queuing

The collecting together of work or tasks to be run on a computer. Users submit tasks or “jobs” to the resource management system where they are queued up until the system is ready to run them.

Scheduling

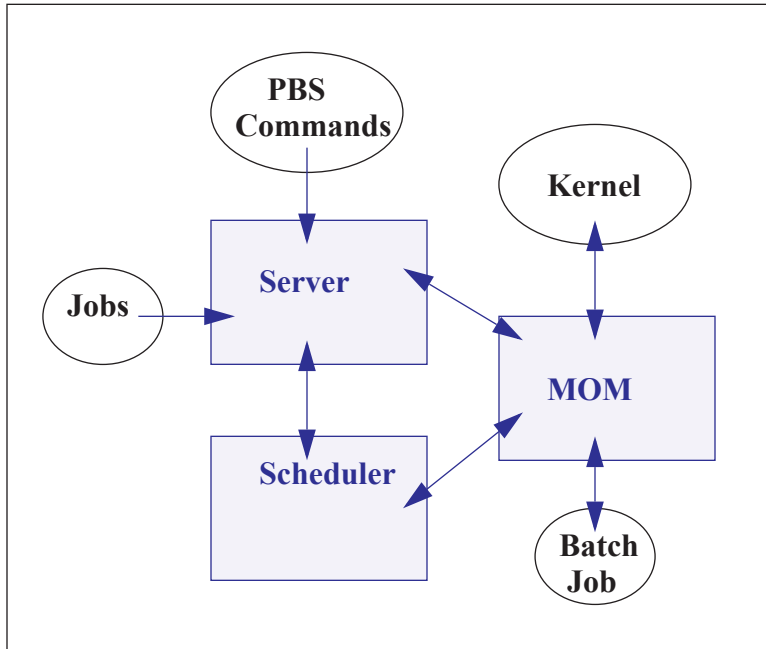
The process of selecting which jobs to run, when, and where, according to a predetermined policy. Sites balance competing needs and goals on the system(s) to maximize efficient use of resources (both computer time and people time).

Monitoring

The act of tracking and reserving system resources and enforcing usage policy. This includes both software enforcement of usage limits and user or administrator monitoring of scheduling policies to see how well they are meeting stated goals.

1.1 PBS Components

PBS consist of two major component types: user-level commands and system daemons/services. A brief description of each is given here to help you understand how the pieces fit together, and how they affect you.



Commands

PBS supplies both command line programs that are POSIX 1003.2d conforming and a graphical interface. These are used to submit, monitor, modify, and delete jobs. These *client commands* can be installed on any system type supported by PBS and do not require the local presence of any of the other components of PBS.

There are three command classifications: user commands, which any authorized user can use, operator commands, and manager (or administrator) commands. Operator and manager commands which require specific access privileges are discussed in the **PBS Professional Administrator's Guide**.

Server

The *Job Server* daemon/service is the central focus for PBS. Within this document, it is generally referred to as *the Server* or by the execution name *pbs_server*. All commands and the other daemons/services communicate with the Server via an *Internet Protocol* (IP) network. The Server's main function is to provide the basic batch services such as receiving/creating a batch job, modifying the job, and running the job. Normally, there is one Server managing a given set of resources. However if the Server Failover feature is enabled, there will be two Servers.

Job Executor (MOM)

The *Job Executor* or *MOM* is the daemon/service which actually places the job into execution. This process, *pbs_mom*, is informally called *MOM* as it is the mother of all executing jobs. (MOM is a reverse-engineered acronym that stands for Machine Oriented Mini-server.) MOM places a job into execution when it receives a copy of the job from a Server. MOM creates a new session that is as identical to a user login session as is possible. (For example under UNIX, if the user's login shell is `csh`, then MOM creates a session in which `.login` is run as well as `.cshrc`.) MOM also has the responsibility for returning the job's output to the user when directed to do so by the Server. One MOM runs on each computer which will execute PBS jobs.

Scheduler

The *Job Scheduler* daemon/service, *pbs_sched*, implements the site's policy controlling when each job is run and on which resources. The Scheduler communicates with the various MOMs to query the state of system resources and with the Server for availability of jobs to execute. The interface to the Server is through the same API as used by the client commands. Note that the Scheduler interfaces with the Server with the same privilege as the PBS manager.

Chapter 2

Getting Started With PBS

This chapter introduces the user to PBS Professional. It describes new user-level features in this release, explains the different user interfaces, introduces the concept of a PBS “job”, and shows how to set up your environment for running batch jobs with PBS.

2.1 New Features in PBS 12.0

2.1.1 Shrink-to-fit Jobs

PBS allows users to specify a variable running time for jobs. Job submitters can specify a **walltime** range for jobs where attempting to run the job in a tight time slot can be useful. Administrators can convert non-shrink-to-fit jobs into shrink-to-fit jobs in order to maximize machine use. See [section 8.14, “Adjusting Job Running Time”, on page 229](#).

2.2 New Features in PBS Professional 11.3

2.2.1 Deleting Moved and Finished Jobs

You can delete a moved or finished job. See [section 6.9.6.2, “Deleting Moved and Finished Jobs”, on page 168](#).

2.3 New Features in PBS Professional 11.2

2.3.1 Grouping Jobs by Project

You can group your jobs by project, by assigning project names. See [section 3.13.11, “Specifying a Job’s Project”, on page 71](#).

2.3.2 Support for Accelerators on Cray

You can request accelerators for Cray jobs. See [section 11.5.11, “Requesting Accelerators”, on page 289](#).

2.3.3 Support for X Forwarding for Interactive Jobs

You can receive X output from interactive jobs. See [section 3.13.22, “Receiving X Output from Interactive Jobs”, on page 78](#).

2.4 New Features in PBS Professional 11.1

2.4.1 Support for Interlagos Hardware

You can request Interlagos hardware for your jobs. See [section 11.5.10, “Requesting Interlagos Hardware”, on page 289](#).

2.5 New Features in PBS Professional 11.0

2.5.1 Improved Cray Integration

PBS is more tightly integrated with Cray systems. You can use the PBS select and place language when submitting Cray jobs. See [section , “Submitting Cray Jobs”, on page 273](#).

2.5.2 Enhanced Job Placement

PBS allows job submitters to scatter chunks by vnode in addition to scattering by host. PBS also allows job submitters to reserve entire hosts via a job’s placement request. See [section 3.6, “Placing Jobs on Vnodes”, on page 43](#).

2.6 New Features in PBS Professional 10.4

2.6.1 Estimated Job Start Times

PBS can estimate the start time and vnodes for jobs. See [section 7.1.21, “Viewing Estimated Start Times For Jobs”, on page 185](#).

2.6.2 Unified Job Submission

PBS allows users to submit jobs using the same scripts, whether the job is submitted on a Windows or UNIX/Linux system. See [section 3.3.3.1, “Python Job Scripts”, on page 26](#).

2.7 New Features in PBS Professional 10.2

2.7.1 Provisioning

PBS provides automatic provisioning of an OS or application on vnodes that are configured to be provisioned. When a job requires an OS that is available but not running, or an application that is not installed, PBS provisions the vnode with that OS or application. See [Chapter 12, “Using Provisioning”, on page 301](#) [Chapter 12, “Using Provisioning”, on page 301](#).

2.7.2 Walltime as Checkpoint Interval Measure

PBS allows a job to be checkpointed according to its walltime usage. See the `pbs_job_attributes` (7B) manual page.

2.7.3 Employing User Space Mode on IBM InfiniBand Switches

PBS allows users submitting POE jobs to use InfiniBand switches in User Space mode. See [section 4.2.5, “IBM POE with PBS”, on page 93](#).

2.8 New Features in Version 10.1

2.8.1 Submitting HPCBP Jobs

Support for HPCBP jobs is **deprecated**. PBS Professional can schedule and manage jobs on one or more HPC Basic Profile compliant servers using the Grid Forum OGSA HPC Basic Profile web services standard. You can submit a generic job to PBS, so that PBS can run it on an HPC Basic Profile Server. This chapter describes how to use PBS for HPC Basic Profile jobs. See [Chapter 10, "HPC Basic Profile Jobs", on page 255](#).

2.8.2 Using Job History Information

PBS Professional can provide job history information, including what the submission parameters were, whether the job started execution, whether execution succeeded, whether staging out of results succeeded, and which resources were used. PBS can keep job history for jobs which have finished execution, were deleted, or were moved to another server. See [section 6.9, "Using Job History Information", on page 164](#).

2.8.3 Reservation Fault Tolerance

PBS attempts to reconfirm reservations for which associated vnodes have become unavailable. See [section 8.7.8.1.i, "Reservation Fault Tolerance", on page 221](#).

2.9 New Features in Recent Releases

2.9.1 Path to Binaries (10.0)

The path to the PBS binaries may have changed for your system. If the old path was not one of `/opt/pbs`, `/usr/pbs`, or `/usr/local/pbs`, you may need to add `/opt/pbs/default` to your PATH environment variable.

2.9.2 Using `job_sort_key` (10.0)

The `sort_priority` option to `job_sort_key` is replaced with the `job_priority` option.

2.9.3 Job-Specific Staging and Execution Directories (9.2)

PBS can now provide a staging and execution directory for each job. Jobs have new attributes `sandbox` and `jobdir`, the MOM has a new option `$jobdir_root`, and there is a new environment variable called `PBS_JOBDIR`. If the job's `sandbox` attribute is set to `PRI-VATE`, PBS creates a job-specific staging and execution directory. If the job's `sandbox` attribute is unset or is set to `HOME`, PBS uses the user's home directory for staging and execution, which is how previous versions of PBS behaved. See [section 8.6, "Input/Output File Staging", on page 198](#).

2.9.4 Standing Reservations (9.2)

PBS now provides a facility for making standing reservations. A standing reservation is a series of advance reservations. The `pbs_rsub` command is used to create both advance and standing reservations. See [section 8.7, "Advance and Standing Reservation of Resources", on page 209](#).

2.10 Deprecations

For a list of deprecations, see [section 1.3, "Deprecations and Removals" on page 8 in the PBS Professional Administrator's Guide](#).

2.11 Backward Compatibility

2.11.1 Job Dependencies Affected By Job History

Enabling job history changes the behavior of dependent jobs. If a job `j1` depends on a finished job `j2` for which PBS is maintaining history than `j1` will go into the held state. If job `j1` depends on a finished job `j3` that has been purged from the historical records than `j1` will be rejected just as in previous versions of PBS where the job was no longer in the system.

2.11.2 PBS path information no longer saved in AUTOEXEC.BAT

Any value for PATH saved in AUTOEXEC.BAT may be lost after installation of PBS. If there is any path information that needs to be saved, AUTOEXEC.BAT must be edited by hand after the installation of PBS. PBS path information is no longer saved in AUTOEXEC.BAT.

2.12 Using PBS

From the user's perspective, a workload management system allows you to make more efficient use of your time. You specify the tasks you need executed. The system takes care of running these tasks and returning the results to you. If the available computers are full, then the workload management system holds your work and runs it when the resources are available.

With PBS you create a *batch job* which you then submit to PBS. A batch job is a file (a shell script under UNIX or a cmd batch file under Windows) containing the set of commands you want to run on some set of execution machines. It also contains directives which specify the characteristics (attributes) of the job, and resource requirements (e.g. memory or CPU time) that your job needs. Once you create your PBS job, you can reuse it if you wish. Or, you can modify it for subsequent runs. For example, here is a simple PBS batch job:

UNIX:

```
#!/bin/sh
#PBS -l walltime=1:00:00
#PBS -l mem=400mb,ncpus=4
./my_application
```

Windows:

```
#PBS -l walltime=1:00:00
#PBS -l mem=400mb,ncpus=4
my_application
```

Don't worry about the details just yet; the next chapter will explain how to create a batch job of your own.

2.13 PBS Interfaces

PBS provides two user interfaces: a command line interface (CLI) and a graphical user interface (GUI). The CLI lets you type commands at the system prompt. The GUI is a graphical point-and-click interface. The “user commands” are discussed in this book; the “administrator commands” are discussed in the **PBS Professional Administrator’s Guide**. The subsequent chapters of this book will explain how to use both the CLI and GUI versions of the user commands to create, submit, and manipulate PBS jobs.

Table 2-1: PBS Professional User and Manager Commands

User Commands		Administrator Commands	
Command	Purpose	Command	Purpose
nqs2pbs	Convert from NQS	pbs-report	Report job statistics
pbs_rdel	Delete a Reservation		
pbs_rstat	Status a Reservation	pbs_hostn	Report host name(s)
pbs_password	Update per user / per server password ¹	pbs_migrate_users	Migrate per user / per server passwords ¹
pbs_rsub	Submit a Reservation	pbs_probe	PBS diagnostic tool
pbsdsh	PBS distributed shell		
qalter	Alter job	pbs_telsh	TCL with PBS API
qdel	Delete job	pbsfs	Show fairshare usage
qhold	Hold a job	pbsnodes	Vnode manipulation
qmove	Move job	printjob	Report job details
qmsg	Send message to job	qdisable	Disable a queue
qorder	Reorder jobs	qenable	Enable a queue
qrls	Release hold on job	qmgr	Manager interface
qselect	Select jobs by criteria	qrerun	Requeue running job

Table 2-1: PBS Professional User and Manager Commands

User Commands		Administrator Commands	
qsig	Send signal to job	qrun	Manually start a job
qstat	Status job, queue, Server	qstart	Start a queue
qsub	Submit a job	qstop	Stop a queue
tracejob	Report job history	qterm	Shutdown PBS
xpbs	Graphical User Interface	xpbsmon	GUI monitoring tool

Notes:

1 Available on Windows only.

2.14 User's PBS Environment

In order to have your system environment interact seamlessly with PBS, there are several items that need to be checked. In many cases, your system administrator will have already set up your environment to work with PBS.

In order to use PBS to run your work, the following are needed:

- User must have access to the resources/hosts that the site has configured for PBS
- User must have a valid account (username and group) on the execution hosts
- User must be able to transfer files between hosts (e.g. via `rscp` or `scp`)
- User's time zone environment variable must be set correctly in order to use advance and standing reservations. See [section 8.7.9.1, “Setting the Submission Host's Time Zone”, on page 223](#).

The subsequent sections of this chapter discuss these requirements in detail, and provide various setup procedures.

2.15 Usernames Under PBS

By default PBS will use your login identifier as the username under which to run your job. This can be changed via the “-u” option to `qsub`. See [section 3.13.15, “Specifying Job User ID”, on page 73](#). The user submitting the job must be authorized to run the job under the execution user name (whether explicitly specified or not).

IMPORTANT:

PBS enforces a maximum username length of 15 characters. If a job is submitted to run under a username longer than this limit, the job will be rejected.

2.16 Setting Up Your UNIX/Linux Environment

2.16.1 Setting PBS_EXEC on UNIX/Linux

In order to make it easier to submit a job script, you can set up your environment so that the correct value for `PBS_EXEC` is used automatically.

Under `sh` or `bash`, do the following:

```
% . /etc/pbs.conf
```

2.16.2 Preventing Problems

A user's job may not run if the user's start-up files (i.e. `.cshrc`, `.login`, or `.profile`) contain commands which attempt to set terminal characteristics. Any such command sequence within these files should be skipped by testing for the environment variable `PBS_ENVIRONMENT`. This can be done as shown in the following sample `.login`:

```
setenv MANPATH /usr/man:/usr/local/man:$MANPATH
if ( ! $?PBS_ENVIRONMENT ) then
    do terminal settings here
endif
```

You should also be aware that commands in your startup files should not generate output when run under PBS. As in the previous example, commands that write to stdout should not be run for a PBS job. This can be done as shown in the following sample `.login`:

```
setenv MANPATH /usr/man:/usr/local/man:$MANPATH
if ( ! $?PBS_ENVIRONMENT ) then
    do terminal settings here
    run command with output here
endif
```

When a PBS job runs, the “exit status” of the last command executed in the job is reported by the job’s shell to PBS as the “exit status” of the job. (We will see later that this is important for job dependencies and job chaining.) However, the last command executed might not be the last command in your job. This can happen if your job’s shell is `csh` on the execution host

and you have a `.logout` there. In that case, the last command executed is from the `.logout` and not your job. To prevent this, you need to preserve the job's exit status in your `.logout` file, by saving it at the top, then doing an explicit `exit` at the end, as shown below:

```
set EXITVAL = $status
previous contents of .logout here
exit $EXITVAL
```

Likewise, if the user's login shell is `csh` the following message may appear in the standard output of a job:

Warning: no access to tty, thus no job control in this shell

This message is produced by many `csh` versions when the shell determines that its input is not a terminal. Short of modifying `csh`, there is no way to eliminate the message. Fortunately, it is just an informative message and has no effect on the job.

2.16.2.1 Interactive Jobs

An interactive job comes complete with a pseudotty suitable for running those commands that set terminal characteristics. But more importantly, it does not caution the user that starting something in the background that would persist after the user has exited from the interactive environment might cause trouble for some MoMs. They could believe that once the interactive session terminates, all the user's processes are gone with it. For example, applications like `ssh-agent` background themselves into a new session and would prevent a CPU set-enabled mom from deleting the CPU set for the job. This in turn might cause subsequent failed attempts to run new jobs, resulting in them being placed in a held state.

2.16.3 Setting MANPATH on SGI Systems

The PBS “man pages” (UNIX manual entries) are installed on SGI systems under `/usr/bsd`, or for the Altix, in `/usr/pbs/man`. In order to find the PBS man pages, users will need to ensure that `/usr/bsd` is set within their `MANPATH`. The following example illustrates this for the C shell:

```
setenv MANPATH /usr/man:/usr/local/man:/usr/bsd:$MANPATH
```

2.17 Setting Up Your Windows Environment

This section discusses the setup steps needed for running PBS Professional in a Microsoft Windows environment, including host and file access, passwords, and restrictions on home directories.

2.17.1 Setting PBS_EXEC on Windows

In order to make it easier to submit a job script, you can set up your environment so that the correct value for PBS_EXEC is used automatically. Under Windows, do the following:

1. Look into "C:\Program Files\PBS Pro\pbs.conf", and get the value of PBS_EXEC. It will be something like "C:\Program Files\PBS Pro\exec".
2. Set your environment accordingly:

```
cmd> set PBS_EXEC="<path>"
```

For example,

```
cmd> set PBS_EXEC="C:\Program Files\PBS Pro\exec"
```

2.17.2 Windows User's HOMEDIR

Each Windows user is assumed to have a home directory (HOMEDIR) where his/her PBS jobs are initially started.

If a user has not been explicitly assigned a home directory, then PBS will use this Windows-assigned default as the base location for the user's default home directory. More specifically, the actual home path will be:

```
[PROFILE_PATH]\My Documents\PBS Pro
```

For instance, if a *userA* has not been assigned a home directory, it will default to a local home directory of:

```
\Documents and Settings\userA\My  
Documents\PBS Pro
```

UserA's job will use the above path as its working directory.

Note that Windows can return as PROFILE_PATH one of the following forms:

```
\Documents and Settings\username  
\Documents and Settings\username.local-host  
name  
\Documents and Settings\username.local-host  
name.00N  
where N is a number  
\Documents and Settings\username.domain-name
```


2.17.3 Windows Usernames and Job Submission

A PBS job is run from a user account and the associated username string must conform to the POSIX-1 standard for portability. That is, the username must contain only alphanumeric characters, dot (.), underscore (_), and/or hyphen "-". The hyphen must not be the first letter of the username. If "@" appears in the username, then it will be assumed to be in the context of a Windows domain account: `username@domainname`. An exception to the above rule is the space character, which is allowed. If a space character appears in a username string, then it will be displayed quoted and must be specified in a quoted manner. The following example requests the job to run under account "Bob Jones".

```
qsub -u "Bob Jones" my_job
```

2.17.4 Windows rhosts File

The Windows `rhosts` file is located in the user's `[PROFILE_PATH]`, for example: `\Documents and Settings\username\.rhosts`, with the format:

hostname username

IMPORTANT:

Be sure the `.rhosts` file is owned by user or an administrator-type group, and has write access granted only to the owning user or an administrator or group.

This file can also determine if a remote user is allowed to submit jobs to the local PBS Server, if the mapped user is an Administrator account. For example, the following entry in user `susan`'s `.rhosts` file on the server would permit user `susan` to run jobs submitted from her workstation `wks031`:

```
wks031 susan
```

Furthermore, in order for Susan's output files from her job to be returned to her automatically by PBS, she would need to add an entry to her `.rhosts` file on her workstation naming the execution host `Host1`.

```
Host1 susan
```

If instead, Susan has access to several execution hosts, she would need to add all of them to her `.rhosts` file:

```
Host1 susan
Host2 susan
Host3 susan
```

Note that Domain Name Service (DNS) on Windows may return different permutations for a full hostname, thus it is important to list all the names that a host may be known. For instance, if Host4 is known as "Host4", "Host4.<subdomain>", or "Host4.<subdomain>.<domain>" you should list all three in the `.rhosts` file.

```
Host4 susan
Host4.subdomain susan
Host4.subdomain.domain susan
```

As discussed in the previous section, usernames with embedded white space must also be quoted if specified in any `hosts.equiv` or `.rhosts` files, as shown below.

```
Host5.subdomain.domain "Bob Jones"
```

2.17.5 Windows Mapped Drives and PBS

In Windows XP, when you map a drive, it is mapped "locally" to your session. The mapped drive cannot be seen by other processes outside of your session. A drive mapped on one session cannot be un-mapped in another session even if it's the same user. This has implications for running jobs under PBS. Specifically if you map a drive, `chdir` to it, and submit a job from that location, the `vnode` that executes the job may not be able to deliver the files back to the same location from which you issued `qsub`. The workaround is to use the `-o` or `-e` options to `qsub` and specify a local (non-mapped) directory location for the job output and error files. For details see [section 3.13.2.2, "Specifying Path for Output and Error Files", on page 65](#).

2.18 Environment Variables

There are a number of environment variables provided to the PBS job. Some are taken from the user's environment and carried with the job. Others are created by PBS. Still others can be explicitly created by the user for exclusive use by PBS jobs. All PBS-provided environment variable names start with the characters `"PBS_"`. Some are then followed by a capital O (`"PBS_O_"`) indicating that the variable is from the job's originating environment (i.e. the

user's). See [“PBS Environment Variables” on page 463 of the PBS Professional Reference Guide](#) for a full listing of all environment variables provided to PBS jobs and their meaning. The following short example lists some of the more useful variables, and typical values.

```
PBS_O_HOME=/u/user1
PBS_O_LOGNAME=user1
PBS_O_PATH=/usr/new/bin:/usr/local/bin:/bin
PBS_O_SHELL=/sbin/csh
PBS_O_HOST=cray1
PBS_O_WORKDIR=/u/user1
PBS_O_QUEUE=submit
PBS_JOBID=16386.cray1
PBS_QUEUE=crayq
PBS_ENVIRONMENT=PBS_INTERACTIVE
```

There are a number of ways that you can use these environment variables to make more efficient use of PBS. In the example above we see **PBS_ENVIRONMENT**, which we used earlier in this chapter to test if we were running under PBS. Another commonly used variable is **PBS_O_WORKDIR** which contains the name of the directory from which the user submitted the PBS job.

There are also two environment variables that you can set to affect the behavior of PBS. The environment variable **PBS_DEFAULT** defines the name of the default PBS Server. Typically, it corresponds to the system name of the host on which the Server is running. If **PBS_DEFAULT** is not set, the default is defined by an administrator established file (usually `/etc/pbs.conf` on UNIX, and `[PBS Destination Folder]\pbs.conf` on Windows).

The environment variable **PBS_DPREFIX** determines the prefix string which identifies directives in the job script. The default prefix string is “#PBS”; however the Windows user may wish to change this as discussed in [section 3.11, “Changing the Job’s PBS Directive”, on page 59](#).

2.19 Temporary Scratch Space: TMPDIR

PBS creates an environment variable, **TMPDIR**, which contains the full path name to a temporary “scratch” directory created for each PBS job. The directory will be removed when the job terminates.

Under Windows, **TMP** will also be set to the value of `%TMPDIR%`. The temporary directory will be created under either `\winnt\temp` or `\windows\temp`, unless an alternative directory was specified by the administrator in the MOM configuration file.

Users can access the job-specific temporary space, by changing directory to it inside their job script. For example:

UNIX:

```
cd $TMPDIR
```

Windows:

```
cd %TMPDIR%
```


Chapter 3

Submitting a PBS Job

This chapter describes virtual nodes, how to submit a PBS job, how to use resources for jobs, how to place your job on vnodes, job attributes, and several related topics.

3.1 Vnodes: Virtual Nodes

A virtual node, or vnode, is an abstract object representing a set of resources which form a usable part of a machine. This could be an entire host, or a nodeboard or a blade. A single host can be made up of multiple vnodes. Each vnode can be managed and scheduled independently. PBS views hosts as being composed of one or more vnodes. Jobs run on one or more vnodes. See the `pbs_node_attributes(7B)` man page.

3.1.1 Relationship Between Hosts, Nodes, and Vnodes

A host is any computer. Execution hosts used to be called nodes. However, some machines such as the Altix can be treated as if they are made up of separate pieces containing CPUs, memory, or both. Each piece is called a vnode. Some hosts have a single vnode and some have multiple vnodes. PBS treats all vnodes alike in most respects. Chunks cannot be split across hosts, but they can be split across vnodes on the same host.

Resources that are defined at the host level are applied to vnodes. A host-level resource is shared among the vnodes on that host. This sharing is managed by the MOM.

3.1.2 Vnode Types

What were called nodes are now called vnodes. All vnodes are treated alike, and are treated the same as what were called “time-shared nodes”. The types “time-shared” and “cluster” are deprecated. The `:ts` suffix is deprecated. It is silently ignored, and not preserved during rewrite. The vnode attribute `ntype` was only used to distinguish between PBS and Globus vnodes. Globus can still send jobs to PBS, but PBS no longer supports sending jobs to Globus. The `ntype` attribute is read-only.

3.2 PBS Resources

Resources can be available on the server and queues, and on vnodes. Jobs can request resources. Resources are allocated to jobs, and some resources such as memory are consumed by jobs. The scheduler matches requested resources with available resources, according to rules defined by the administrator. PBS can enforce limits on resource usage by jobs.

PBS provides built-in resources, and in addition, allows the administrator to define custom resources. The administrator can specify which resources are available on a given vnode, as well as at the server or queue level (e.g. floating licenses.) Vnodes can share resources. The administrator can also specify default arguments for `qsub`. These arguments can include resources. See the `qsub(1B)` man page.

Resources made available by defining them via `resources_available` at the server level are only used as job-wide resources. These resources (e.g. `walltime`, `server_dyn_res`) are requested using `-l RESOURCE=VALUE`. Resources made available at the host (vnode) level are only used as chunk resources, and can only be requested within chunks using `-l select=RESOURCE=VALUE`. Resources such as `mem` and `ncpus` can only be used at the vnode level.

Resources are allocated to jobs both by explicitly requesting them and by applying specified defaults. Jobs explicitly request resources either at the vnode level in chunks defined in a selection statement, or in job-wide resource requests. See the `pbs_resources(7B)` manual page.

Jobs are assigned limits on the amount of resources they can use. These limits apply to how much the job can use on each vnode (per-chunk limit) and to how much the whole job can use (job-wide limit). Limits are derived from both requested resources and applied default resources.

Each chunk's per-chunk limits determine how much of any resource can be used in that chunk. Per-chunk resource usage limits are the amount of per-chunk resources requested, both from explicit requests and from defaults.

Job resource limits set a limit for per-job resource usage. Job resource limits are derived in this order from:

1. explicitly requested job-wide resources (e.g. -l resource=value)
2. the select specification (e.g. -l select =...)
3. the queue's resources_default.RES
4. the server's resources_default.RES
5. the queue's resources_max.RES
6. the server's resources_max.RES

The server's default_chunk.RES does **not** affect job-wide limits.

The consumable resources requested for chunks in the select specification are summed, and this sum is used for a job-wide limit. Job resource limits from sums of all chunks override those from job-wide defaults and resource requests.

Various limit checks are applied to jobs. If a job's job resource limit exceeds queue or server restrictions, it will not be put in the queue or accepted by the server. If, while running, a job exceeds its limit for a consumable or time-based resource, it will be terminated.

A “consumable” resource is one that is reduced by being used, for example, ncpus, licenses, or mem. A “non-consumable” resource is not reduced through use, for example, walltime or a boolean resource.

Resources are tracked in server, queue, vnode and job attributes. Servers, queues and vnodes have two attributes, resources_available.RESOURCE and resources_assigned.RESOURCE. The resources_available.RESOURCE attribute tracks the total amount of the resource available at that server, queue or vnode, without regard to how much is in use. The resources_assigned.RESOURCE attribute tracks how much of that resource has been assigned to jobs at that server, queue or vnode. Jobs have an attribute called resources_used.RESOURCE which tracks the amount of that resource used by that job.

The administrator can set server and queue defaults for resources used in chunks. See the PBS Professional Administrator's Guide and the pbs_server_attributes(7B) and pbs_queue_attributes(7B) manual pages.

For a thorough discussion of PBS resources, see [“PBS Resources” on page 281 of the PBS Professional Reference Guide](#).

3.2.1 Unset Resources

When job resource requests are being matched with available resources, a numerical resource that is unset on a host is treated as if it were zero, and an unset string cannot be matched. An unset Boolean resource is treated as if it is set to “False”. An unset resource at the server or queue is treated as if it were infinite.

3.2.2 Resource Names

The resource name is any string made up of alphanumeric characters, where the first character is alphabetic. Resource names must start with an alphabetic character and can contain alphanumeric, underscore (“_”), and dash (“-”) characters.

3.2.3 Specifying Resource Values

- Resource values which contain commas, quotes, plus signs, equal signs, colons, or parentheses must be quoted to PBS. The string must be enclosed in quotes so that the command (e.g. qsub, qalter) will parse it correctly.
- When specifying resources via the command line, any quoted strings must be escaped or enclosed in another set of quotes. This second set of quotes must be different from the first set, meaning that double quotes must be enclosed in single quotes, and vice versa.
- If a string resource value contains spaces or shell metacharacters, enclose the string in quotes, or otherwise escape the space and metacharacters. Be sure to use the correct quotes for your shell and the behavior you want.

3.2.4 Resource Types

See [“Resource Data Types” on page 297 of the PBS Professional Reference Guide](#) for a description of resource types.

3.2.5 Built-in Resources

See [“Built-in Resources” on page 299 of the PBS Professional Reference Guide](#) for a list of built-in resources.

3.3 PBS Jobs

3.3.1 Rules for Submitting Jobs

- The "place" specification cannot be used without the "select" specification. See [section 3.6, “Placing Jobs on Vnodes”, on page 43](#).
- A "select" specification cannot be used with a "nodes" specification.
- A "select" specification cannot be used with old-style resource requests such as -lncpus, -lmem, -lvmem, -larch, -lhost.
- The built-in resource "software" is not a vnode-level resource. See [“Built-in Resources” on page 299 of the PBS Professional Reference Guide](#).
- A PBS job can be submitted at the command line or via `xpbs`.
- At the command line, the user can create a job script, and submit it. During submission it is possible to override elements in the job script. Alternatively, PBS will read from input typed at the command line.

3.3.2 Introduction to the PBS Job Script

3.3.2.1 Contents of a Job Script

A PBS job script consists of:

- An optional shell specification
- PBS directives
- Tasks (programs or commands)

3.3.2.2 Types of Job Scripts

PBS allows you to use various kinds of job scripts. You can use any of the following:

- A Python script that can run under Windows or UNIX/Linux
- A UNIX shell script that runs under UNIX/Linux
- Windows command batch script under Windows

3.3.2.3 Submitting a Job Script

Before submitting a job script using these instructions, be sure to set your environment appropriately. If you want the correct value for `PBS_EXEC` to be used automatically, see [section 2.16.1, “Setting PBS_EXEC on UNIX/Linux”, on page 13](#) and [section 2.17.1, “Setting PBS_EXEC on Windows”, on page 15](#).

To submit a PBS job, type the following:

UNIX/Linux shell script:

```
qsub <name of shell script>
```

UNIX/Linux Python script:

```
qsub -S $PBS_EXEC/bin/pbs_python <python job script>
```

Windows command script:

```
qsub <name of job script>
```

Windows Python script:

```
qsub -S %PBS_EXEC%\bin\pbs_python.exe <python job script>
```

If the path contains any spaces, it must be quoted, for example:

```
qsub -S "%PBS_EXEC%\bin\pbs_python.exe" <python job script>
```

3.3.3 The Job Script

3.3.3.1 Python Job Scripts

PBS allows you to submit jobs using a Python script. You can use the same Python script under Windows or UNIX/Linux. PBS includes a Python package, allowing Python job scripts to run; you do not need to install Python. To run a Python job script:

UNIX/Linux:

```
qsub -S $PBS_EXEC/bin/pbs_python <script name>
```

Windows:

```
qsub -S %PBS_EXEC%\bin\pbs_python.exe <script name>
```

If the path contains any spaces, it must be quoted, for example:

```
qsub -S "%PBS_EXEC%\bin\pbs_python.exe" <python job script>
```

You can include PBS directives in a Python job script as you would in a UNIX shell script. For example:

```
% cat myjob.py
#PBS -l select=1:ncpus=3:mem=1gb
#PBS -N HelloJob
print "Hello"
```

Python job scripts can access Win32 APIs, including the following modules:

- Win32api
- Win32con
- Pywintypes

3.3.3.1.i Windows Python Caveat

If you have Python natively installed, and you need to use the `win32api`, make sure that you import `pywintypes` before `win32api`, otherwise you will get an error. Do the following:

```
cmd> pbs_python
>> import pywintypes
>> import win32api
```

3.3.3.2 UNIX Shell Scripts

Since the job file can be a shell script, the first line of a shell script job file specifies which shell to use to execute the script. The user's login shell is the default, but you can change this. This first line can be omitted if it is acceptable for the job file to be interpreted using the login shell.

3.3.3.3 Windows Command Scripts

If the job file is a shell script, specify the shell in the first line of the job file.

3.3.3.3.i Windows Caveats

In Windows, if you use `notepad` to create a job script, the last line does not automatically get newline-terminated. Be sure to put one explicitly, otherwise, PBS job will get the following error message:

```
More?
```

when the Windows command interpreter tries to execute that last line.

3.3.3.4 Perl Scripts Under Windows

PBS jobs execute under the native cmd environment, and not under cygwin. To run a Perl script under Windows, you must specify the path to Perl in your job script.

Format:

```
<path to Perl> my_job.pl [arguments to script]
```

3.3.3.5 PBS Directives

PBS directives are at the top of the script file. They are used to request resources or set attributes. A directive begins with the default string “#PBS”. Attributes can also be set using options to the `qsub` command, which will override directives. The limit for a PBS directive is 2048 characters.

3.3.3.6 The User’s Tasks

These can be programs or commands. This is where the user specifies an application to be run.

3.3.3.7 Setting Job Attributes

Job attributes can be set by either of the following methods:

- Using PBS directives in the job script
- Giving options to the `qsub` command at the command line

These two methods have the same functionality. Options to the `qsub` command will override PBS directives, which override defaults. Some job attributes have default values preset in PBS. Some job attributes’ default values are set at the user’s site.

After the job is submitted, you can use the `qalter` command to change the job’s characteristics.

Job attributes are case-insensitive.

3.3.3.8 Debugging Job Scripts

You can run Python interactively, outside of PBS, to debug a Python job script. You use the Python interpreter to test parts of your script.

Under UNIX/Linux, use the `-i` option to the `pbs_python` command, for example:

```
/opt/pbs/default/bin/pbs_python -i <return>
```

Under Windows, the `-i` option is not necessary, but can be used. For example, either of the following will work:

```
C:\Program Files\PBS Pro\exec\bin\pbs_python.exe <return>
C:\Program Files\PBS Pro\exec\bin\pbs_python.exe -i <return>
```

When the Python interpreter runs, it presents you with its own prompt. For example:

```
% /opt/pbs/default/bin/pbs_python -i <return>
>> print "hello"
hello
```

3.3.4 Job Script Names

It is recommended to avoid using special characters in a job script name. If you must use them, on UNIX/Linux you must escape them using the backslash (“\”) character.

3.4 Submitting a PBS Job

There are a few ways to submit a PBS job using the command line. The first is to create a job script and submit it using `qsub`.

3.4.1 Submitting a Job Script

For example, with job script “myjob”, the user can submit it by typing

```
qsub myjob
16387.foo.exampledomain
```

PBS returns a *job identifier* (e.g. “16387.foo.exampledomain” in the example above.) Its format will be:

```
sequence-number.servername
```

or, for a job array,

```
sequence-number[ ].servername.domain
```

You’ll need the job identifier for any actions involving the job, such as checking job status, modifying the job, tracking the job, or deleting the job.

If “my_job” contains the following, the user is naming the job “testjob”, and running a program called “myprogram”.

```
#!/bin/sh
#PBS -N testjob
./myprogram
```

The largest possible job ID is the 7-digit number 9,999,999. After this has been reached, job IDs start again at zero.

3.4.1.1 Overriding Directives

PBS directives in a script can be overridden by using the equivalent options to qsub. For example, to override the PBS directive naming the job, and name it “newjob”, the user could type

```
qsub -N newjob my_job
```

3.4.1.2 Submitting a Simple Job

Jobs can also be submitted without specifying values for attributes. The simplest way to submit a job is to type

```
qsub myjobscript <ret>
```

If myjobscript contains

```
#!/bin/sh
./myapplication
```

the user has simply told PBS to run myapplication.

3.4.1.3 Passing Arguments to Job Scripts

If you need to pass arguments to a job script, you can either use the -v option to qsub, where you set and use environment variables, or use standard input. When using standard input, any #PBS directives in the job script will be ignored. You can replace directives with the equivalent options to qsub. To use standard input, you can either use this form:

```
echo "jobscript.sh -a foo -b bar" | qsub -l select=...
```

or you can use this form:

```
qsub [option] [option] ... <ret>
./jobscript.sh foo      <^d>
152.mymachine
```

With this form, you can type the #PBS directives on lines the name of the job script. If you do not use the -n option to qsub, or specify it via a #PBS directive (second form only), the job will be named STDIN.

3.4.2 Jobs Without a Job Script

There are two ways to submit PBS jobs without using a job script. You can run a PBS job by specifying an executable and its arguments instead of a job script. You can also specify that qsub read input from the keyboard.

3.4.2.1 Submitting Jobs by Specifying Executables

When you specify only the executable with any options and arguments, PBS starts a shell for you. To submit a job from the command line, the format is the following:

```
qsub [options] -- executable [arguments to executable] <return>
```

For example, to run myprog with the arguments a and b:

```
qsub -- myprog a b <return>
```

To run myprog with the arguments a and b, naming the job *JobA*,

```
qsub -N JobA -- myprog a b <return>
```

3.4.2.2 Submitting Jobs Using Keyboard Input

It is possible to submit a job to PBS without first creating a job script file. If you run the qsub command, with the resource requests on the command line, and then press “enter” without naming a job file, PBS will read input from the keyboard. (This is often referred to as a “here document”.) You can direct qsub to stop reading input and submit the job by typing on a line by itself a control-d (UNIX) or control-z, then enter (Windows).

Note that, under UNIX, if you enter a control-c while qsub is reading input, qsub will terminate the process and the job will not be submitted. Under Windows, however, often the control-c sequence will, depending on the command prompt used, cause qsub to submit the job to PBS. In such case, a control-break sequence will usually terminate the qsub command.

```
qsub <ret>
[directives]
[tasks]
ctrl-D
```


3.5 Requesting Resources

PBS provides built-in resources, and allows the administrator to define custom resources. The administrator can specify which resources are available on a given vnode, as well as at the queue or server level (e.g. floating licenses.) See [“Built-in Resources” on page 299 of the PBS Professional Reference Guide](#) for a listing of built-in resources.

Resources defined at the queue or server level apply to an entire job. If they are defined at the vnode level, they apply only to the part of the job running on that vnode.

Jobs request resources, which are allocated to the job, along with any defaults specified by the administrator.

Custom resources are used for application licenses, scratch space, etc., and are defined by the administrator. See section 5.14, "Custom Resources", on page 312 of the PBS Professional Administrator's Guide. Custom resources are used the same way built-in resources are used.

Jobs request resources in two ways. They can use the *select statement* to define *chunks* and specify the quantity of each chunk. A chunk is a set of resources that are to be allocated as a unit. Jobs can also use a job-wide resource request, which uses `resource=value` pairs, outside of the select statement. Format:

qsub (non-resource portion of job)

-l <resource>=<value> (this is the job-wide request)

-l select=<chunk>[+<chunk>] (this is the selection statement)

See [section 3.5.2, “Requesting Resources in Chunks”, on page 33](#) and [section 3.5.3, “Requesting Job-wide Resources”, on page 34](#).

The `qsub`, `qalter` and `pbs_rsub` commands are used to request resources. However, custom resources which were created to be invisible or unrequestable cannot be requested. See [section 3.5.15, “Resource Permissions”, on page 42](#).

The `-lnodes=` form is deprecated, and if it is used, it will be converted into a request for chunks and job-wide resources. Most jobs submitted with `"-lnodes"` will continue to work as expected. These jobs will be automatically converted to the new syntax. However, job tasks may execute in an unexpected order, because vnodes may be assigned in a different order. Jobs submitted with old syntax that ran successfully on versions of PBS Professional prior to 8.0 can fail because a limit that was per-chunk is now job-wide. This is an example of a job submitted using `-lnodes=X -lmem=M` that would fail because the mem limit is now job-wide. If the following conditions are true:

- a. PBS Professional 9.0 or later using standard MPICH
- b. The job is submitted with `qsub -lnodes=5 -lmem=10GB`
- c. The master process of this job tries to use more than 2GB

The job will be killed, where in <= 7.0 the master process could use 10GB before being killed. 10GB is now a job-wide limit, divided up into a 2GB limit per chunk.

For more information see the `qsub(1B)`, `qalter(1B)`, `pbs_rsub(1B)` and `pbs_resources(7B)` manual pages.

Do not use an old-style resource or node specification (“-lnodes=”) with “-lselect” or “-lplace”. This will produce an error.

Each kind of resource plays a specific role, which is either inside chunks or outside of them, but not both. Some resources, e.g. `ncpus`, can only be used at the host (chunk) level. The rest, e.g. `walltime`, can only be used at the job-wide level. Therefore, no resource can be requested both inside and outside of a selection statement. Keep in mind that requesting, for example, `-lncpus` is the old form, which cannot be mixed with the new form.

3.5.1 Resource Allocation

Resources are allocated to jobs both because jobs explicitly request them and because specified default resources are applied to jobs. Jobs explicitly request resources either at the vnode level in *chunks* defined in a *selection statement*, or in *job-wide* resource requests, outside of a selection statement. An explicit resource request can appear in the following, in order of precedence:

1. `qalter`
2. `qsub`
3. PBS job script directives

3.5.2 Requesting Resources in Chunks

A *chunk* declares the value of each resource in a set of resources which are to be allocated as a unit to a job. It is the smallest set of resources that will be allocated to a job. All of a chunk must be taken from a single host. A chunk request is a vnode-level request. Chunks are described in a selection statement, which specifies how many of each kind of chunk. A selection statement has this form:

```
-l select=[N:]chunk[+[N:]chunk ...]
```

If `N` is not specified, it is taken to be 1.

No spaces are allowed between chunks.

A chunk is one or more `resource_name=value` statements separated by a colon, e.g.:

```
ncpus=2:mem=10GB:host=Host1
ncpus=1:mem=20GB:arch=linux
```

Example of multiple chunks in a selection statement:

```
-l select= 2:ncpus=1:mem=10GB +3:ncpus=2:mem=8GB:arch=solaris
```

Each job submission can have only one “-l select” statement.

Host-level resources can only be requested as part of a chunk. Server or queue resources cannot be requested as part of a chunk. They must be requested outside of the selection statement.

3.5.3 Requesting Job-wide Resources

A *job-wide* resource request is for resource(s) at the server or queue level. This resource must be a server-level or queue-level resource. A job-wide resource is designed to be used by the entire job, and is available to the complex, not just one execution host.

Job-wide resources are requested outside of a selection statement, in this form:

```
-l keyword=value[,keyword=value ...]
```

where *keyword* identifies either a consumable resource or a time-based resource such as **walltime**.

Job-wide resources are used for requesting floating licenses or other resources not tied to specific vnodes, such as **cput** and **walltime**.

Job-wide resources can only be requested outside of chunks.

A resource request “outside of a selection statement” means that the resource request comes after “-l”, but not after “-lselect=”.

3.5.4 Boolean Resources

A resource request can specify whether a **boolean** resource should be true or false. For example, if some vnodes have **green=true** and some are **red=true**, a selection statement for two vnodes, each with one CPU, all green and no red, would be:

```
-l select=2:green=true:red=false:ncpus=1
```

The next example Windows script shows a job-wide request for walltime and a chunk request for ncpus and memory.

```
#PBS -l walltime=1:00:00
#PBS -l select=ncpus=4:mem=400mb
#PBS -j oe

date /t
.\my_application
date /t
```

Keep in mind the difference between requesting a vnode-level boolean and a job-wide boolean.

```
qsub -l select=1:green=True
```

will request a vnode with green set to True. However,

```
qsub -l green=True
```

will request green set to True on the server and/or queue.

3.5.5 Default Resources

Jobs get default resources, both job-wide and per-chunk, with the following order of precedence, from

1. Default `qsub` arguments
2. Default queue resources
3. Default server resources

For each chunk in the job's selection statement, first queue chunk defaults are applied, then server chunk defaults are applied. If the chunk request does not specify a resource listed in the defaults, the default is added. For a resource `RESOURCE`, a chunk default is called "default_chunk.RESOURCE".

For example, if the queue in which the job is enqueued has the following defaults defined:

```
default_chunk.ncpus=1
default_chunk.mem=2gb
```

a job submitted with this selection statement:

```
select=2:ncpus=4+1:mem=9gb
```

will have this specification after the default_chunk elements are applied:

```
select=2:ncpus=4:mem=2gb+1:ncpus=1:mem=9gb.
```

In the above, *mem=2gb* and *ncpus=1* are inherited from default_chunk.

The job-wide resource request is checked against queue resource defaults, then against server resource defaults. If a default resource is defined which is not specified in the resource request, it is added to the resource request.

3.5.6 Requesting Application Licenses

Application licenses are set up as resources defined by the administrator. PBS doesn't actually check out the licenses, the application being run inside the job's session does that.

3.5.6.1 Floating Licenses

PBS queries the license server to find out how many floating licenses are available at the beginning of each scheduling cycle. If you wish to request a site-wide floating license, it will typically have been set up as a server-level (job-wide) resource. To request an application license called AppF, use:

```
qsub -l AppF=<number of licenses> <other qsub
arguments>
```

If only certain hosts can run the application, they will typically have a host-level boolean resource set to True. To request the application license and the vnodes on which to run the application, use:

```
qsub -l AppF=<number of licenses>
      <other qsub arguments>
      -l select=haveAppF=True
```

PBS doesn't actually check out the licenses, the application being run inside the job's session does that.

3.5.6.2 Node-locked Licenses

Per-host node-locked licenses are typically set up as either a boolean resource on the vnode(s) that are licensed for the application. The resource request should include one license for each host. To request a host with a per-host node-locked license for AppA in one chunk:

```
qsub -l select=1:runsAppA=1 <jobscript>
```

Per-use node-locked licenses are typically set up so that the host(s) that run the application have the number of licenses that can be used at one time. The number of licenses the job requests should be the same as the number of instances of the application that will be run. To request a host with a per-use node-locked license for AppB, where you'll run one instance of AppB on two CPUs in one chunk:

```
qsub -l select=1:ncpus=2:AppB=1
```

Per-CPU node-locked licenses are set up so that the host has one license for each licensed CPU. You must request one license for each CPU. To request a host with a node-locked license for AppC, where you'll run a job using two CPUs in one chunk:

```
qsub -l select=1:ncpus=2:AppC=2
```

3.5.7 Requesting Scratch Space

Scratch space on a machine is set up as a host-level dynamic resource. The resource will have a name such as “dynscratch”. To request 10MB of scratch space in one chunk, a resource request would include:

```
-l select=1:ncpus=N:dynscratch=10MB
```

3.5.8 Requesting GPUs

Your PBS job can request GPUs. Your administrator can configure PBS to support any of the following:

- Job uses non-specific GPUs and exclusive use of a node
- Job uses non-specific GPUs and shared use of a node
- Job uses specific GPUs and either shared or exclusive use of a node

3.5.8.1 Binding to GPUs

PBS Professional allocates GPUs, but does not bind jobs to any particular GPU; the application itself, or the CUDA library, is responsible for the actual binding.

3.5.8.2 Requesting Non-specific GPUs and Exclusive Use of Node

If your job needs GPUs, but does not require specific GPUs, and can request exclusive use of GPU nodes, you can request GPUs the same way you request CPUs.

Your administrator can set up a resource to represent the GPUs on a node. We recommend that the GPU resource is called *ngpus*.

You submit jobs, for example “*my_gpu_job*”, requesting one node with two GPUs and one CPU, and exclusive use of the node, in the following manner:

```
qsub -lselect=1:ncpus=1:ngpus=2 -lplace=excl my_gpu_job
```

When requesting GPUs in this manner, your job should request exclusive use of the node to prevent other jobs being scheduled on its GPUs.

It is up to the application or CUDA to bind the GPUs to the application processes.

3.5.8.3 Requesting Non-specific GPUs and Shared Use of Node

Your administrator can configure PBS to allow your job to use non-specific GPUs on a node while sharing GPU nodes. In this case, your administrator puts each GPU in its own vnode.

Your administrator can configure a resource to represent GPUs. We recommend that the GPU resource is called *ngpus*.

Your administrator can configure each GPU vnode so it has a resource containing the device number of the GPU. We recommend that this resource is called *gpu_id*.

You submit jobs, for example “*my_gpu_job*”, requesting two GPUs and one CPU, and shared use of the node, in the following manner:

```
qsub -lselect=1:ncpus=1:ngpus=2 -lplace=shared my_gpu_job
```

When a job is submitted requesting any GPU, the PBS scheduler looks for a vnode with an available GPU and assigns that vnode to the job. Since there is a one-to-one correspondence between GPUs and vnodes, the job can determine the *gpu_id* of that vnode. Finally, the application can use the appropriate CUDA call to bind the process to the allocated GPU.

3.5.8.4 Requesting Specific GPUs

Your job can request one or more particular GPUs. This allows you to run applications on the GPUs for which the applications are written.

Your administrator can set up a resource to allow jobs to request specific GPUs. We recommend that the GPU resource is called *gpu_id*.

The following requests 4 vnodes, each with GPU with ID 0:

```
qsub -lselect=4:ncpus=1:gpu_id=gpu0 my_gpu_job
```

When a job is submitted requesting specific GPUs, the PBS scheduler assigns the vnode with the resource containing that *gpu_id* to the job. The application can use the appropriate CUDA call to bind the process to the allocated GPU.

3.5.8.5 Viewing GPU Information for Nodes

You can find the number of GPUs available and assigned on execution hosts via the `pbsnodes` command. See [“pbsnodes” on page 105 of the PBS Professional Reference Guide](#).

3.5.9 Note About Submitting Jobs

The default for walltime is 5 years. The scheduler uses walltime to predict when resources will become available. Therefore it is useful to request a reasonable walltime for each job.

3.5.10 Submitting Jobs with Resource Specification (Old Syntax)

If neither a node specification nor a selection directive is specified, then a selection directive will be created requesting 1 chunk with resources specified by the job, and with those from the queue or server default resource list. These are: `ncpus`, `mem`, `arch`, `host`, and `software`, as well as any other default resources specified by the administrator.

For example, a job submitted with

```
qsub -l ncpus=4:mem=123mb:arch=linux
```

will have the following selection directive created:

```
select=1:ncpus=4:mem=123mb:arch=linux
```

Do not mix old style resource or node specification with the `select` and `place` statements. Do not use one in a job script and the other on the command line. This will result in an error.

3.5.11 Moving Jobs From One Queue to Another

If the job is moved from the current queue to a new queue, any default resources in the job's resource list that were contributed by the current queue are removed. This includes a `select` specification and `place` directive generated by the rules for conversion from the old syntax. If a job's resource is unset (undefined) and there exists a default value at the new queue or server, that default value is applied to the job's resource list. If either `select` or `place` is missing from the job's new resource list, it will be automatically generated, using any newly inherited default values.

Example:

Given the following set of queue and server default values:

Server

`resources_default.ncpus=1`

Queue QA

`resources_default.ncpus=2`

`default_chunk.mem=2gb`

Queue QB

`default_chunk.mem=1gb`

no default for ncpus

The following illustrate the equivalent select specification for jobs submitted into queue QA and then moved to (or submitted directly to) queue QB:

`qsub -l ncpus=1 -lmem=4gb`

In QA: `select=1:ncpus=1:mem=4gb`

No defaults need be applied

In QB: `select=1:ncpus=1:mem=4gb`

No defaults need be applied

`qsub -l ncpus=1`

In QA: `select=1:ncpus=1:mem=2gb`

Picks up 2gb from queue default chunk and 1 ncpus from qsub

In QB: `select=1:ncpus=1:mem=1gb`

Picks up 1gb from queue default chunk and 1 ncpus from qsub

`qsub -lmem=4gb`

In QA: `select=1:ncpus=2:mem=4gb`

Picks up 2 ncpus from queue level job-wide resource default and 4gb mem from qsub

In QB: `select=1:ncpus=1:mem=4gb`

Picks up 1 ncpus from server level job-wide default and 4gb mem from qsub

qsub -lnodes=4

In QA: `select=4:ncpus=1:mem=2gb`

Picks up a queue level default memory chunk of 2gb. (This is not 4:ncpus=2 because in prior versions, "nodes=x" implied 1 CPU per node unless otherwise explicitly stated.)

In QB: `select=4:ncpus=1:mem=1gb` (In prior versions, "nodes=x" implied 1 CPU per node unless otherwise explicitly stated, so the ncpus=1 is not inherited from the server default.)

qsub -l mem=16gb -lnodes=4

In QA: `select=4:ncpus=1:mem=4gb` (This is not 4:ncpus=2 because in prior versions, "nodes=x" implied 1 CPU per node unless otherwise explicitly stated.)

In QB: `select=4:ncpus=1:mem=4gb` (In prior versions, "nodes=x" implied 1 CPU per node unless otherwise explicitly stated, so the ncpus=1 is not inherited from the server default.)

3.5.12 Resource Request Conversion Dependent on Where Resources are Defined

A job's resource request is converted from old-style to new according to various rules, one of which is that the conversion is dependent upon where resources are defined. For example: The boolean resource "Red" is defined on the server, and the boolean resource "Blue" is defined at the host level. A job requests "`qsub -l Blue=True`". This looks like an old-style resource request, and PBS checks to see where Blue is defined. Since Blue is defined at the host level, the request is converted into "`-l select=1:Blue=True`". However, if a job requests "`qsub -l Red=True`", while this looks like an old-style resource request, PBS does not convert it to a chunk request because Red is defined at the server.

3.5.13 Jobs Submitted with Undefined Resources

Any job submitted with undefined resources, specified either with "-l select" or with "-lnodes", will not be rejected at submission. The job will be aborted upon being enqueued in an execution queue if the resources are still undefined. This preserves backward compatibility.

3.5.14 Limits on Resource Usage

Each chunk's per-chunk limits determine how much of any resource can be used in that chunk. Per-chunk resource usage limits are established by per-chunk resources, both from explicit requests and from defaults.

Job resource limits set a limit for per-job resource usage. Job resource limits are established both by requesting job-wide resources and by summing per-chunk consumable resources. Job resource limits from sums of all chunks, including defaults, override those from job-wide defaults. Limits include both explicitly requested resources and default resources.

If a job's job resource limit exceeds queue or server restrictions, it will not be put in the queue or accepted by the server. If, while running, a job exceeds its limit for a consumable or time-based resource, it will be terminated. See **The PBS Professional Administrator's Guide**.

Job limits are created from the directive for each consumable resource.

For example,

```
qsub -lselect=2:ncpus=3:mem=4gb:arch=linux
```

will have the following job limits set:

- ncpus=6
- mem=8gb

3.5.15 Resource Permissions

Custom resources can be created so that they are invisible, or cannot be requested or altered. If a resource is invisible it also cannot be requested or altered. The function of some PBS commands depends upon whether a resource can be viewed, requested or altered. These commands are those which view or request resources or modify resource requests:

pbsnodes

Users cannot view restricted host-level custom resources.

pbs_rstat

Users cannot view restricted reservation resources.

pbs_rsub	Users cannot request restricted custom resources for reservations.
qalter	Users cannot alter a restricted resource.
qmgr	Users cannot print or list a restricted resource.
qselect	Users cannot specify restricted resources via <code>-l resource_list</code> .
qsub	Users cannot request a restricted resource.
qstat	Users cannot view a restricted resource.

3.6 Placing Jobs on Vnodes

The *place statement* and the `vnode sharing` attribute controls how the job is placed on the vnodes from which resources may be allocated for the job. The *place statement* can be specified, in order of precedence, via:

1. Explicit placement request in `qalter`
2. Explicit placement request in `qsub`
3. Explicit placement request in PBS job script directives
4. Default `qsub` place statement
5. Queue default placement rules
6. Server default placement rules
7. Built-in default conversion and placement rules

The *place statement* may be not be used without the `select` statement.

The *place statement* has this form:

```
-l place=[arrangement][: sharing][: grouping]
```

where

arrangement is one of `free` | `pack` | `scatter` | `vscatter`

sharing is one of `excl` | `shared` | `exclhost`

grouping can have only one instance of `group=resource`

and where

Table 3-1: Placement Modifiers

Modifier	Meaning
free	Place job on any vnode(s)
pack	All chunks will be taken from one host
scatter	Only one chunk is taken from any host
vscatter	Only one chunk is taken from any vnode. Each chunk must fit on a vnode.
excl	Only this job uses the vnodes chosen
exclhost	The entire host is allocated to the job
shared	This job can share the vnodes chosen
group=resource	Chunks will be placed on vnodes according to a resource shared by those vnodes. This resource must be a string or string array. All vnodes in the group must have a common value for the resource, which can be either the built-in resource host or a site-defined vnode-level resource

Grouping by resource name will override `node_group_key`. To run a job on a single host, use “-lplace=pack”.

3.6.1 Inheriting Placement Specifications

If the administrator has defined default values for arrangement, sharing, and grouping, each job inherits these unless it explicitly requests at least one. That means that if your job requests arrangement, but not sharing or grouping, it will not inherit values for sharing or grouping. For example, the administrator sets a default of `place=pack:exclhost:group=host`. Job A requests `place=free`, but doesn’t specify sharing or grouping, so Job A does not inherit sharing or grouping. Job B does not request any placement, so it inherits all three.

3.6.2 Using *place=group*

When a job requests `place=group=<resource>`, PBS looks for enough vnodes to satisfy the request, where all of the selected vnodes share a common value for the specified resource.

The specified resource must be a string or string array.

The value of the resource can be one or more strings at each vnode, but there must be one string that is the same for each vnode. For example, if the resource is *router*, the value can be “*r1i0,r1*” at one vnode, and “*r1i1,r1*” at another vnode, but these vnodes can be grouped because they share the string “*r1*”.

3.6.3 Specifying Primary Execution Host

The job’s primary execution host is the host that supplies the vnode to satisfy first chunk requested by the job. You can see which host is the primary execution host in the following ways:

- The primary execution host is the first host listed in the job’s node file; see [section 3.6.5, “The Job’s Node File”, on page 47](#).
- The primary execution host is the first host listed in the job’s `exec_vnode` attribute; see [section 7.1.19, “Viewing Resources Allocated to a Job”, on page 182](#).

3.6.4 Vnodes Allocated to a Job

3.6.4.1 Hosts Allocated to a Job

The node file contains the names of the vnodes allocated to a job. The name of the node file is given in the environment variable `PBS_NODEFILE`. The order in which hosts appear in the file is the order in which chunks are specified in the selection directive. The order in which hostnames appear in the file is `hostA X times, hostB Y times`, where `X` is the number of MPI processes on `hostA`, `Y` is the number of MPI processes on `hostB`, etc. See [section 3.6.5, “The Job’s Node File”, on page 47](#). See also the definition of the resources “`mpiprocs`” and “`ompthreads`” in [“Built-in Resources” on page 299 of the PBS Professional Reference Guide](#), and [section 4.1.4, “Specifying Number of MPI Processes Per Chunk”, on page 83](#).

3.6.4.2 Specifying Shared or Exclusive Use of Vnodes

Each vnode can be allocated exclusively to one job, or its resources can be shared among jobs. Hosts can also be allocated exclusively to one job, or shared among jobs.

How vnodes are allocated to jobs is determined by a combination of the vnode's **sharing** attribute and the job's resource request. The possible values for the vnode **sharing** attribute, and how they interact with a job's placement request, are described in [“sharing” on page 371 of the PBS Professional Reference Guide](#). The following table expands on this:

Table 3-2: How Vnode sharing Attribute Affects Vnode Allocation

Value of Vnode sharing Attribute	Effect on Allocation
not set	The job's arrangement request determines how vnodes are allocated to the job. If there is no specification, vnodes are shared.
<i>default_share</i>	Vnodes are shared unless the job explicitly requests exclusive use of the vnodes.
<i>default_excl</i>	Vnodes are allocated exclusively to the job unless the job explicitly requests shared allocation.
<i>default_exclhost</i>	All vnodes from this host are allocated exclusively to the job, unless the job explicitly requests shared allocation.
<i>ignore_excl</i>	Vnodes are shared, regardless of the job's request.
<i>force_excl</i>	Vnodes are allocated exclusively to the job, regardless of the job's request.
<i>force_exclhost</i>	All vnodes from this host are allocated exclusively to the job, regardless of the job's request.

If a vnode is allocated exclusively to a job, all of its resources are assigned to the job. The state of the vnode becomes *job-exclusive*. No other job can use the vnode.

If a host is to be allocated exclusively to one job, all of the host must be used: if any vnode from a host has its sharing attribute set to either *default_exclhost* or *force_exclhost*, all vnodes on that host must have the same value for the sharing attribute.

3.6.4.3 Placing Jobs on Vnodes

Jobs can be placed on vnodes according to the job's placement request. Each chunk from a job can be placed on a different host, or a different vnode. Alternatively, all chunks can be taken from a single host, or from chunks sharing the same value for a specified resource. The job can request exclusive use of each vnode, or shared use with other jobs. The job can

request exclusive use of its hosts. If a job does not request *vscluster* for its arrangement, any chunk can be broken across vnodes. This means that one chunk could be taken from more than one vnode.

3.6.4.4 Restrictions on Placement

If the job requests *vscluster* for its arrangement, no chunk can be larger than a vnode. No chunk can be split across vnodes. This behavior is different from other values for arrangement, where chunks can be split across vnodes.

3.6.5 The Job's Node File

The file containing the vnodes allocated to a job lists the names of the hosts from which the job's vnodes are taken. See [section 4.1.3, “The Job's Node File”, on page 82](#).

3.6.6 Resources Allocated from a Vnode

The resources allocated from a vnode are only those specified in the job's *schedselect*. This job attribute is created internally by starting with the select specification and applying any server and queue default_chunk resource defaults that are missing from the select statement. The schedselect job attribute contains only vnode-level resources. The *exec_vnode* job attribute shows which resources are allocated from which vnodes. See [“Job Attributes” on page 375 of the PBS Professional Reference Guide](#).

3.6.6.1 Resources Assigned to a Job

The *Resource_List* attribute is the list of resources requested via qsub, with job-wide defaults applied. Vnode-level resources from *Resource_List* are used in the converted select when the user doesn't specify a select statement. The converted select statement is used to fill in gaps in schedselect.

Values for ncpus or mem in the job's *Resource_List* come from three places:

1. Resources specified via qsub,
2. the sum of the values in the select specification (not including default_chunk), or
3. resources inherited from queue and/or server resources_default.

Case 3 applies only when the user does not specify -l select, but uses -lnodes or -lncpus instead.

The *Resource_List.mem* is a job-wide memory limit which, if memory enforcement is enabled, the entire job (the sum of all of the job's usage) cannot exceed.

Examples:

The queue has the following:

```
resources_default.mem=200mb
default_chunk.mem=100mb
```

A job requesting `-l select=2:ncpus=1:mem=345mb` will take 345mb from each of two vnodes and have a job-wide limit of 690mb ($2 * 345$). The job's `Resource_List.mem` will show 690mb.

A job requesting `-l select=2:ncpus=2` will take 100mb (`default_chunk`) value from each vnode and have a job wide limit of 200mb ($2 * 100mb$). The job's `Resource_List.mem` will show 200mb.

A job requesting `-l ncpus=2` will take 200mb (inherited from `resources_default` and used to create the select spec) from one vnode and a job-wide limit of 200mb. The job's `Resource_List.mem` will show 200mb.

A job requesting `-lnodes=2` will inherit the 200mb from `resources_default.mem` which will be the job-wide limit. The memory will be taken from the two vnodes, half (100mb) from each. The generated select spec is `2:ncpus=1:mem=100mb`. The job's `Resource_List.mem` will show 200mb.

3.7 Submitting Jobs Using Select & Place: Examples

Unless otherwise specified, the vnodes allocated to the job will be allocated as shared or exclusive based on the setting of the vnode's sharing attribute. Each of the following shows how you would use `-l select=` and `-l place=`.

1. A job that will fit in a single host such as an Altix but not in any of the vnodes, packed into the fewest vnodes:


```
-l select=1:ncpus=10:mem=20gb
-l place=pack
```

In earlier versions, this would have been:

```
-lncpus=10,mem=20gb
```
2. Request four chunks, each with 1 CPU and 4GB of memory taken from anywhere.


```
-l select=4:ncpus=1:mem=4GB
-l place=free
```
3. Allocate 4 chunks, each with 1 CPU and 2GB of memory from between

one and four vnodes which have an arch of “linux”.

```
-l select=4:ncpus=1:mem=2GB:arch=linux -l place=free
```

4. Allocate four chunks on 1 to 4 vnodes where each vnode must have 1 CPU, 3GB of memory and 1 node-locked dyna license available for each chunk.

```
-l select=4:dyna=1:ncpus=1:mem=3GB -l place=free
```

5. Allocate four chunks on 1 to 4 vnodes, and 4 floating dyna licenses. This assumes “dyna” is specified as a server dynamic resource.

```
-l dyna=4 -l select=4:ncpus=1:mem=3GB -l place=free
```

6. This selects exactly 4 vnodes where the arch is linux, and each vnode will be on a separate host. Each vnode will have 1 CPU and 2GB of memory allocated to the job.

```
-lselect=4:mem=2GB:ncpus=1:arch=linux -lplace=scatter
```

7. This will allocate 3 chunks, each with 1 CPU and 10GB of memory. This will also reserve 100mb of scratch space if scratch is to be accounted . Scratch is assumed to be on

a file system common to all hosts. The value of “place” depends on the default which is “place=free”.

```
-l scratch=100mb -l select=3:ncpus=1:mem=10GB
```

8. This will allocate 2 CPUs and 50GB of memory on a host named zooland. The value of “place” depends on the default which defaults to “place=free”:

```
-l select=1:ncpus=2:mem=50gb:host=zooland
```

9. This will allocate 1 CPU and 6GB of memory and one host-locked swlicense from each of two hosts:

```
-l select=2:ncpus=1:mem=6gb:swlicense=1  
-l place=scatter
```

10. Request free placement of 10 CPUs across hosts:

```
-l select=10:ncpus=1  
-l place=free
```

11. Here is an odd-sized job that will fit on a single Altix, but not on any one node-board. We request an odd number of CPUs that are not shared, so they must be “rounded up”:

```
-l select=1:ncpus=3:mem=6gb  
-l place=pack:excl
```

12. Here is an odd-sized job that will fit on a single Altix, but not on any one node-board. We are asking for small number of CPUs but a large amount of memory:

```
-l select=1:ncpus=1:mem=25gb  
-l place=pack:excl
```

13. Here is a job that may be run across multiple Altix systems, packed into the fewest vnodes:

```
-l select=2:ncpus=10:mem=12gb  
-l place=free
```

14. Submit a job that must be run across multiple Altix systems, packed into the fewest vnodes:

```
-l select=2:ncpus=10:mem=12gb  
-l place=scatter
```

15. Request free placement across nodeboards within a single host:

```
-l select=1:ncpus=10:mem=10gb  
-l place=group=host
```

-
16. Request free placement across vnodes on multiple Altixes:

```
-l select=10:ncpus=1:mem=1gb  
-l place=free
```

17. Here is a small job that uses a shared cpuset:

```
-l select=1:ncpus=1:mem=512kb  
-l place=pack:shared
```

18. Request a special resource available on a limited set of nodeboards, such as a graphics card:

```
-l select= 1:ncpus=2:mem=2gb:graphics=True +  
  1:ncpus=20:mem=20gb:graphics=False  
-l place=pack:excl
```

19. Align SMP jobs on c-brick boundaries:

```
-l select=1:ncpus=4:mem=6gb  
-l place=pack:group=cbrick
```

20. Align a large job within one router, if it fits within a router:

```
-l select=1:ncpus=100:mem=200gb  
-l place=pack:group=router
```

21. Fit large jobs that do not fit within a single router into as few available routers as possible. Here, RES is the resource used for node grouping:

```
-l select=1:ncpus=300:mem=300gb  
-l place=pack:group=<RES>
```

22. To submit an MPI job, specify one chunk per MPI task. For a 10-way MPI job with 2gb of memory per MPI task:

```
-l select=10:ncpus=1:mem=2gb
```

23. To submit a non-MPI job (including a 1-CPU job or an OpenMP or shared memory) job, use a single chunk. For a 2-CPU job requiring 10gb of memory:

```
-l select=1:ncpus=2:mem=10gb
```

3.7.1 Examples Using Old Syntax

1. Request CPUs and memory on a single host using old syntax:
`-l ncpus=5,mem=10gb`
will be converted into the equivalent:
`-l select=1:ncpus=5:mem=10gb`
`-l place=pack`
2. Request CPUs and memory on a named host along with custom resources including a floating license using old syntax:
`-l ncpus=1,mem=5mb,host=sunny,opti=1,arch=solaris`
is converted to the equivalent:
`-l select=1:ncpus=1:mem=5gb:host=sunny:arch=solaris`
`-l place=pack`
`-l opti=1`
3. Request one host with a certain property using old syntax:
`-lnodes=1:property`
is converted to the equivalent:
`-l select=1:ncpus=1:property=True`
`-l place=scatter`
4. Request 2 CPUs on each of four hosts with a given property using old syntax:
`-lnodes=4:property:ncpus=2`
is converted to the equivalent:
`-l select=4:ncpus=2:property=True`

`-l place=scatter`

5. Request 1 CPU on each of 14 hosts asking for certain software, licenses and a job limit amount of memory using old syntax:

```
-lnodes=14:mpi-fluent:ncpus=1 -lfluent=1,fluent-all=1, fluent-par=13
-l mem=280mb
```

is converted to the equivalent:

```
-l select=14:ncpus=1:mem=20mb:mpi_fluent=True
-l place=scatter
-l fluent=1,fluent-all=1,fluent-par=13
```

6. Requesting licenses using old syntax:

```
-lnodes=3:dyna-mpi-Linux:ncpus=2 -ldyna=6,mem=100mb, software=dyna
```

is converted to the equivalent:

```
-l select=3:ncpus=2:mem=33mb: dyna-mpi-Linux=True
-l place=scatter
-l software=dyna
-l dyna=6
```

7. Requesting licenses using old syntax:

```
-l ncpus=2,app_lic=6,mem=200mb -l software=app
```

is converted to the equivalent:

```
-l select=1:ncpus=2:mem=200mb
-l place=pack
-l software=app
-l app_lic=6
```

8. Additional example using old syntax:

```
-lnodes=1:fserver+15:noserver
```

is converted to the equivalent:

```
-l select=1:ncpus=1:fserver=True + 15:ncpus=1:noserver=True
-l place=scatter
```

but could also be more easily specified with something like:

```
-l select=1:ncpus=1:fserver=True + 15:ncpus=1:fserver=False
-l place=scatter
```

9. Allocate 4 vnodes, each with 6 CPUs with 3 MPI processes per vnode, with each

vnode on a separate host. The memory allocated would be one-fourth of the memory specified by the queue or server default if one existed. This results in a different placement of the job from version 5.4:

```
-lnodes=4:ppn=3:ncpus=2
```

is converted to:

```
-l select=4:ncpus=6:mpiprocs=3 -l place=scatter
```

10. Allocate 4 vnodes, from 4 separate hosts, with the property blue. The amount of memory allocated from each vnode is 2560MB (= 10GB / 4) rather than 10GB from each vnode.

```
-lnodes=4:blue:ncpus=2 -l mem=10GB
```

is converted to:

```
-l select=4:blue=True:ncpus=2:mem=2560mb -lplace=scatter
```

3.8 Backward Compatibility

For backward compatibility, a legal node specification or resource specification will be converted into selection and placement directives. Specifying “cpp” is part of the old syntax, and should be replaced with “ncpus”. Do not mix old style resource or node specification syntax with select and place statements. If a job is submitted using -l select on the command line, and it contains an old-style specification in the job script, that will result in an error.

When a nodespec is converted into a select statement, the job will have the environment variables NCPUS and OMP_NUM_THREADS set to the value of ncpus in the first piece of the nodespec. This may produce incompatibilities with prior versions when a complex node specification using different values of ncpus and ppn in different pieces is converted.

3.8.1 Node Specification Conversion

Node specification format:

```
-lnodes=[N:spec_list | spec_list]
```

```
[[+N:spec_list | +spec_list] ...]
```

```
[#suffix ...][-lncpus=Z]
```

where:

spec_list has syntax: *spec[:spec ...]*

spec is any of: hostname | property | ncpus=X | cpp=X | ppn=P

suffix is any of: property | excl | shared

N and P are positive integers

X and Z are non-negative integers

The node specification is converted into selection and placement directives as follows:

Each *spec_list* is converted into one chunk, so that N:*spec_list* is converted into N chunks.

If *spec* is hostname :

The chunk will include host=hostname

If *spec* matches any vnode's resources_available.host value:

The chunk will include host=hostname

If *spec* is property :

The chunk will include property=true

Property must be a site-defined vnode-level boolean resource.

If *spec* is ncpus=X or cpp=X :

The chunk will include ncpus=X

If no *spec* is ncpus=X and no *spec* is cpp=X :

The chunk will include ncpus=P

If *spec* is ppn=P :

The chunk will include mpiprocs=P

If the nodespec is

-lnodes=N:ppn=P

It is converted to

-lselect=N:ncpus=P:mpiprocs=P

Example:

-lnodes=4:ppn=2

is converted into

-lselect=4:ncpus=2:mpiprocs=2

If `-lncpus=Z` is specified and no spec contains `ncpus=X` and no spec is `cpp=X` :

Every chunk will include `ncpus=W`, where `W` is `Z` divided by the total number of chunks.
(Note: `W` must be an integer; `Z` must be evenly divisible by the number of chunks.)

If `property` is a suffix :

All chunks will include `property=true`

If `excl` is a suffix :

The placement directive will be `-lplace=scatter:excl`

If `shared` is a suffix :

The placement directive will be `-lplace=scatter:shared`

If neither `excl` nor `shared` is a suffix :

The placement directive will be `-lplace=scatter`

Example:

```
-lnodes=3:green:ncpus=2:ppn=2+2:red
```

is converted to:

```
-l select=3:green=true:ncpus=4:mpiprocs=2+ 2:red=true:ncpus=1
-l place=scatter
```

Node specification syntax for requesting properties is deprecated. The boolean resource syntax "`property=true`" is only accepted in a selection directive. It is erroneous to mix old and new syntax.

3.8.2 Resource Specification Conversion

The resource specification is converted to select and place statements after any defaults have been applied.

Resource specification format:

```
-lresource=value[:resource=value ...]
```

The resource specification is converted to:

```
-lselect=1[:resource=value ...]
-lplace=pack
```

with one instance of *resource=value* for each of the following vnode-level resources in the resource request:

built-in resources: ncpus | mem | vmem | arch | host

site-defined vnode-level resources

3.9 How PBS Parses a Job Script

The `qsub` command scans the lines of the script file for directives. Scanning will continue until the first executable line, that is, a line that is not blank, not a directive line, nor a line whose first non white space character is “#”. If directives occur on subsequent lines, they will be ignored.

A line in the script file will be processed as a directive to `qsub` if and only if the string of characters starting with the first non white space character on the line and of the same length as the directive prefix matches the directive prefix (i.e. “#PBS”). The remainder of the directive line consists of the options to `qsub` in the same syntax as they appear on the command line. The option character is to be preceded with the “-” character.

If an option is present in both a directive and on the command line, that option and its argument, if any, will be ignored in the directive. The command line takes precedence. If an option is present in a directive and not on the command line, that option and its argument, if any, will be taken from there.

3.10 A Sample PBS Job

The following is an example of a job script written in Python. This script calculates the 10th Fibonacci number.

```
% cat job.py
#PBS -l select=1:ncpus=3:mem=1gb
#PBS -N myjob
def fibo(n):
    global fibo
    if n < 2:
        return n
    else:
        return fibo(n - 1) + fibo(n - 2)
print ("fibo(10)=%d" % fibo(10))
```

Note that this script contains PBS directives.

Let's look at an example PBS job in detail:

UNIX/Linux:

```
#!/bin/sh
#PBS -l walltime=1:00:00
#PBS -l select=mem=400mb
#PBS -j oe

date
./my_application
date
```

Windows:

```
#PBS -l walltime=1:00:00
#PBS -l select=mem=400mb
#PBS -j oe

date /t
my_application
date /t
```

On line one in the example above Windows does not show a shell directive. (The default on Windows is the batch command language.) Also note that it is possible under both Windows and UNIX to specify to PBS the scripting language to use to interpret the job script (see the “-S” option to `qsub` in [section 3.13.9, “Specifying Scripting Language to Use”, on page 70](#)). The Windows script will be a .exe or .bat file.

Lines 2-8 of both files are almost identical. The primary differences will be in file and directory path specification (such as the use of drive letters and slash vs. backslash as the path separator).

Lines 2-4 are PBS directives. PBS reads down the shell script until it finds the first line that is not a valid PBS directive, then stops. It assumes the rest of the script is the list of commands or tasks that the user wishes to run. In this case, PBS sees lines 6-8 as being user commands.

The section ["Job Submission Options" on page 62](#) describes how to use the `qsub` command to submit PBS jobs. Any option that you specify to the `qsub` command line (except “-I”) can also be provided as a PBS directive inside the PBS script. PBS directives come in two types: resource requirements and attribute settings.

In our example above, lines 2-3 specify the “-l” resource list option, followed by a specific resource request. Specifically, lines 2-3 request 1 hour of wall-clock time as a job-wide request, and 400 megabytes (MB) of memory in a chunk. .

Line 4 requests that PBS *join* the `stdout` and `stderr` output streams of the job into a single stream.

Finally lines 6-8 are the command lines for executing the program(s) we wish to run. You can specify as many programs, tasks, or job steps as you need.

3.11 Changing the Job’s PBS Directive

By default, the text string “#PBS” is used by PBS to determine which lines in the job file are PBS directives. The leading “#” symbol was chosen because it is a comment delimiter to all shell scripting languages in common use on UNIX systems. Because directives look like comments, the scripting language ignores them. The limit for a PBS directive is 2048 characters.

Under Windows, however, the command interpreter does not recognize the ‘#’ symbol as a comment, and will generate a benign, non-fatal warning when it encounters each “#PBS” string. While it does not cause a problem for the batch job, it can be annoying or disconcerting to the user. Therefore Windows users may wish to specify a different PBS directive, via either the `PBS_DPREFIX` environment variable, or the “-C” option to `qsub`. For example, we can direct PBS to use the string “REM PBS” instead of “#PBS” and use this directive string in our job script:

```
REM PBS -l walltime=1:00:00
REM PBS -l select=mem=400mb
REM PBS -j oe
date /t
.\my_application
date /t
```

Given the above job script, we can submit it to PBS in one of two ways:

```
set PBS_DPREFIX=REM PBS
qsub my_job_script
```

or

```
qsub -C "REM PBS" my_job_script
```

For additional details on the “-C” option to `qsub`, see [section 3.13, “Job Submission Options”, on page 62](#).

3.12 Windows Jobs

3.12.1 Submitting Windows Jobs

Any .bat files that are to be executed within a PBS job script have to be prefixed with "call" as in:

```
@echo off
call E:\step1.bat
call E:\step2.bat
```

Without the "call", only the first .bat file gets executed and it doesn't return control to the calling interpreter.

An example:

A job script that contains:

```
@echo off
E:\step1.bat
E:\step2.bat
```

should now be:

```
@echo off
call E:\step1.bat
call E:\step2.bat
```

Under Windows, comments in the job script must be in ASCII characters.

3.12.2 Passwords

When running PBS in a password-protected Windows environment, you will need to specify to PBS the password needed in order to run your jobs. There are two methods of doing this: (1) by providing PBS with a password once to be used for all jobs ("single signon method"), or (2) by specifying the password for each job when submitted ("per job method"). Check with your system administrator to see which method was configured at your site.

3.12.2.1 Single-Signon Password Method

To provide PBS with a password to be used for all your PBS jobs, use the `pbs_password` command. This command can be used whether or not you have jobs enqueued in PBS. The command usage syntax is:

```
pbs_password [-s server] [-r] [-d] [user]
```

When no options are given to `pbs_password`, the password credential on the default PBS server for the current user, i.e. the user who executes the command, is updated to the prompted password. Any user jobs previously held due to an invalid password are not released.

The available options to `pbs_password` are:

- r** Any user jobs previously held due to an invalid password are released.
- s server** Allows user to specify server where password will be changed.
- d** Deletes the password.
- user** The password credential of user *user* is updated to the prompted password. If *user* is not the current user, this action is only allowed if:
 1. The current user is root or admin.
 2. User *user* has given the current user explicit access via the `ruse-rok ()` mechanism:
 - a The hostname of the machine from which the current user is logged in appears in the server's `hosts.equiv` file, or
 - b The current user has an entry in user's `HOMEDIR\.rhosts` file.

Note that `pbs_password` encrypts the password obtained from the user before sending it to the PBS Server. The `pbs_password` command does not change the user's password on the current host, only the password that is cached in PBS.

The `pbs_password` command is supported only on Windows and all supported Linux platforms on x86 and x86_64.

The `pbs_password` command has no effect on running jobs. Queued jobs use the new password.

3.12.2.2 Per-job Password Method

If you are running in a password-protected Windows environment, but the single-signon method has not been configured at your site, then you will need to supply a password with the submission of each job. You can do this via the `qsub` command, with the `-Wpwd` option, and supply the password when prompted.

```
qsub -Wpwd <job script>
```

You will be prompted for the password, which is passed on to the program, then encrypted and saved securely for use by the job. The password should be enclosed in double quotes.

Keep in mind that in a multi-host job, the password supplied will be propagated to all the sister hosts. This requires that the password be the same on the user's accounts on all the hosts. The use of domain accounts for a multi-host job will be ideal in this case.

Accessing network share drives/resources within a job session also requires that you submit the job with a password via `qsub -W pwd`.

The `-Wpwd` option to the `qsub` command is supported only on Windows and all supported Linux platforms on x86 and x86_64.

3.13 Job Submission Options

There are many options to the `qsub` command. The table below gives a quick summary of the available options; the rest of this chapter explains how to use each one.

Table 3-3: Options to the `qsub` Command

Option	Function and Page Reference
<code>-A account_string</code>	"Specifying a Local Account" on page 76
<code>-a date_time</code>	"Deferring Execution" on page 71
<code>-C "DPREFIX"</code>	"Changing the Job's PBS Directive" on page 59
<code>-c interval</code>	"Specifying Job Checkpoint Interval" on page 72
<code>-e path</code>	"Specifying Path for Output and Error Files" on page 65
<code>-h</code>	"Holding a Job (Delaying Execution)" on page 72
<code>-I</code>	"Interactive-batch Jobs" on page 77

Table 3-3: Options to the qsub Command

Option	Function and Page Reference
-J X-Y[:Z]	“Job Array” on page 235
-j <i>join</i>	"Merging Output and Error Files" on page 66
-k <i>keep</i>	"Retaining Output and Error Files on Execution Host" on page 66
-l resource_list	section 3.3.1, “Rules for Submitting Jobs”, on page 25
-M user_list	"Setting Email Recipient List" on page 68
-m MailOptions	"Specifying Email Notification" on page 67
-N name	"Specifying a Job Name" on page 69
-o path	"Specifying Path for Output and Error Files" on page 65
-p priority	"Setting a Job’s Priority" on page 70
-P project	"Specifying a Job’s Project" on page 71
-q destination	"Specifying Queue and/or Server" on page 64
-r value	"Marking a Job as “Rerunnable” or Not” on page 69
-S path_list	"Specifying Scripting Language to Use" on page 70
-u user_list	"Specifying Job User ID" on page 73
-V	"Exporting Environment Variables" on page 67
-v variable_list	"Expanding Environment Variables" on page 67
-W <attribute>=<value>	"Setting Job Attribute Values" on page 75
-W depend=list	"Specifying Job Dependencies" on page 195
-W group_list=list	"Specifying Job Group ID" on page 76
-W stagein=list	"Input/Output File Staging" on page 198
-W stageout=list	"Input/Output File Staging" on page 198

Table 3-3: Options to the `qsub` Command

Option	Function and Page Reference
<code>-W cred=dce</code>	"Running PBS in a UNIX DCE Environment" on page 227
<code>-W block=opt</code>	"Requesting qsub Wait for Job Completion" on page 194
<code>-W pwd="password"</code>	"Per-job Password Method" on page 62 and "Running PBS in a UNIX DCE Environment" on page 227
<code>-W sandbox=<value></code>	"Staging and Execution Directory: User's Home vs. Job-specific" on page 198
<code>-W umask=nnn</code>	"Changing UNIX Job umask" on page 194
<code>-X</code>	"Receiving X Output from Interactive Jobs" on page 78
<code>-z</code>	"Suppressing Job Identifier" on page 76

3.13.1 Specifying Queue and/or Server

The “`-q destination`” option to `qsub` allows you to specify a particular destination to which you want the job submitted. The *destination* names a queue, a Server, or a queue at a Server. The `qsub` command will submit the script to the Server defined by the *destination* argument. If the *destination* is a routing queue, the job may be routed by the Server to a new destination. If the `-q` option is not specified, the `qsub` command will submit the script to the default queue at the default Server. (See also the discussion of **PBS_DEFAULT** in ["Environment Variables" on page 17](#).) The destination specification takes the following form:

```
-q [queue[@host]]
```

Examples:

```
qsub -q queue my_job
qsub -q @server my_job
#PBS -q queueName
qsub -q queueName@serverName my_job
qsub -q queueName@serverName.domain.com my_job
```

3.13.2 Managing Output and Error Files

3.13.2.1 Default Behavior

PBS, by default, always copies the standard output (stdout) and standard error (stderr) files back to `$PBS_O_WORKDIR` on the submission host when a job finishes. When `qsub` is run, it sets `$PBS_O_WORKDIR` to the current working directory where the `qsub` command is executed.

3.13.2.2 Specifying Path for Output and Error Files

The “`-o path`” and “`-e path`” options to `qsub` allows you to specify the name of the files to which the stdout and the stderr file streams should be written. The path argument is of the form: `[hostname:]path_name` where *hostname* is the name of a host to which the file will be returned and *path_name* is the path name on that host. You may specify relative or absolute paths. If you specify only a file name, it is assumed to be relative to your home directory. Do not use variables in the path. The following examples illustrate these various options.

```
#PBS -o /u/user1/myOutputFile
#PBS -e /u/user1/myErrorFile

qsub -o myOutputFile my_job
qsub -o /u/user1/myOutputFile my_job
qsub -o myWorkstation:/u/user1/myOutputFile my_job
qsub -e myErrorFile my_job
qsub -e /u/user1/myErrorFile my_job
qsub -e myWorkstation:/u/user1/myErrorFile my_job
```

Note that if the PBS client commands are used on a Windows host, then special characters like spaces, backslashes (`\`), and colons (`:`) can be used in command line arguments such as for specifying pathnames, as well as drive letter specifications. The following are allowed:

```
qsub -o \temp\my_out job.scr
qsub -e "host:e:\Documents and Settings\user\Desktop\output"
```

The error output of the above job is to be copied onto the `e:` drive on *host* using the path `"\Documents and Settings\user\Desktop\output"`. The quote marks are required when arguments to `qsub` contain spaces.

3.13.2.3 Output Appended When Job is Rerun

If your job runs and produces output, and then is rerun, meaning that another job with the same name is run, PBS appends the output of the second run to that of the first. The first output is preserved.

3.13.2.4 Merging Output and Error Files

The “-j *join*” option declares if the standard error stream of the job will be merged with the standard output stream of the job. A *join* argument value of *oe* directs that the two streams will be merged, intermixed, as standard output. A *join* argument value of *eo* directs that the two streams will be merged, intermixed, as standard error. If the *join* argument is *n* or the option is not specified, the two streams will be two separate files.

```
qsub -j oe my_job
#PBS -j eo
```

3.13.2.5 Retaining Output and Error Files on Execution Host

The “-k *keep*” option defines which (if either) of standard output (STDOUT) or standard error (STDERR) of the job will be retained in the job’s staging and execution directory on the primary execution host. If set, this option overrides the path name for the corresponding file. If not set, neither file is retained on the execution host. The argument is either the single letter “e” or “o”, or the letters “e” and “o” combined in either order. Or the argument is the letter “n”. If “-k” is not specified, neither file is retained.

e

The standard error file is to be retained in the job’s staging and execution directory on the primary execution host. The job’s name will be the default file name given by: *job_name.e**sequence* where *job_name* is the name specified for the job, and *sequence* is the sequence number component of the job identifier.

o

The standard output file is to be retained in the job’s staging and execution directory on the primary execution host. The file name will be the default file name given by: *job_name.o**sequence* where *job_name* is the name specified for the job, and *sequence* is the sequence number component of the job identifier.

eo, oe

Both standard output and standard streams are retained on the primary execution host, in the job's staging and execution directory.

n

Neither file is retained.

```
qsub -k oe my_job
#PBS -k eo
```

3.13.3 Exporting Environment Variables

The “-V” option declares that all environment variables in the `qsub` command’s environment are to be exported to the batch job.

```
qsub -V my_job
#PBS -V
```

3.13.4 Expanding Environment Variables

The “-v *variable_list*” option to `qsub` allows you to specify additional environment variables to be exported to the job. *variable_list* names environment variables from the `qsub` command environment which are made available to the job when it executes. These variables and their values are passed to the job. These variables are added to those already automatically exported. Format: comma-separated list of strings in the form:

`-v variable`

or

`-v variable=value`

If a *variable=value* pair contains any commas, the value must be enclosed in single or double quotes, and the *variable=value* pair must be enclosed in the kind of quotes not used to enclose the value. For example:

```
qsub -v DISPLAY,myvariable=32 my_job
qsub -v "var1='A,B,C,D'" job.sh
qsub -v a=10, "var2='A,B'", c=20, HOME=/home/zzz job.sh
```

3.13.5 Specifying Email Notification

The “-m *MailOptions*” defines the set of conditions under which the execution server will send a mail message about the job. The *MailOptions* argument is a string which consists of either the single character “n”, or one or more of the characters “a”, “b”, and “e”. If no email notification is specified, the default behavior will be the same as for “-m a”.

a

Send mail when job is *aborted* by batch system

Example:

Job to be deleted at request "root@host."

b

Send mail when job *begins* execution

Example:

 Begun execution

e

Send mail when job *ends* execution

Example:

 Execution terminated

 Exit_status=0

 resources_used.cput=0

 resources_used.cput=00:00:00

 resources_used.mem=2464kb

 resources_used.ncpus=1

 resources_used.vmem=2455kb

 resources_used.walltime=00:00:07

n

Do not send mail

Examples:

```
qsub -m ae my_job
```

```
#PBS -m b
```

3.13.6 Setting Email Recipient List

The “-M *user_list*” option declares the list of users to whom mail is sent by the execution server when it sends mail about the job. The *user_list* argument is of the form:

```
user[@host][,user[@host],...]
```

If unset, the list defaults to the submitting user at the `qsub` host, i.e. the job owner.

```
qsub -M user1@mydomain.com my_job
```

3.13.6.1 Caveats

PBS on Windows can only send email to addresses that specify an actual hostname that accepts port 25 (sendmail) requests. For the above example on Windows you will need to specify:

```
qsub -M user1@host.mydomain.com
```

where `host.mydomain.com` accepts port 25 connections.

3.13.7 Specifying a Job Name

The “-N *name*” option declares a name for the job. The *name* specified may be up to 15 characters in length. The first character must be alphabetic, numeric, hyphen, underscore, or plus sign. If the -N option is not specified, the job name will be the base name of the job script file specified on the command line. If no script file name was specified and the script was read from the standard input, then the job name will be set to STDIN.

```
qsub -N myName my_job  
#PBS -N myName
```

3.13.8 Marking a Job as “Rerunnable” or Not

The “-r *y|n*” option declares whether the job is rerunnable. To rerun a job is to terminate the job and requeue it in the execution queue in which the job currently resides. The *value* argument is a single character, either “y” or “n”. If the argument is “y”, the job is rerunnable. If the argument is “n”, the job is not rerunnable. The default value is “n”, not rerunnable.

```
qsub -r n my_job  
#PBS -r n
```

Marking your job as non-rerunnable will not affect how PBS treats it in the case of startup failure. If a job that is marked non-rerunnable has an error during startup, before it begins execution, that job is requeued for another attempt. The purpose of marking a job as non-rerunnable is to prevent it from running twice and using data that undergoes a change during execution. However, if the job never actually starts execution, the data isn’t altered before the job uses it, so PBS requeues it.

PBS requeues some jobs that are terminated before execution. Two examples of this are multi-host jobs where the job did not start on one or more execution hosts, and provisioning jobs for which the provisioning script failed.

Interactive jobs are not rerunnable.

3.13.9 Specifying Scripting Language to Use

The “-S *path_list*” option declares the path and name of the scripting language to be used in interpreting the job script. The option argument *path_list* is in the form: *path[@host] [, path[@host] , . . .]* Only one path may be specified for any host named, and only one path may be specified without the corresponding host name. The path selected will be the one with the host name that matched the name of the execution host. If no matching host is found, then the path specified without a host will be selected, if present. If the -S option is not specified, the option argument is the null string, or no entry from the *path_list* is selected, then PBS will use the user’s login shell on the execution host.

Example 3-1: Using bash via a directive:

```
#PBS -S /bin/bash@mars,/usr/bin/bash@jupiter
```

Example 3-2: Running a Python script from the command line on UNIX/Linux:

```
qsub -S /opt/pbs/default/bin/pbs_python <script name>
```

Example 3-3: Running a Python script from the command line on Windows:

```
qsub -S "C:\Program Files\PBS Pro\exec\bin\pbs_python.exe" <script name>
```

3.13.9.1 Windows Caveats

Using this option under Windows is more complicated because if you change from the default shell of cmd, then a valid PATH is not automatically set. Thus if you use the “-S” option under Windows, you must explicitly set a valid PATH as the first line of your job script.

3.13.10 Setting a Job’s Priority

The “-p *priority*” option defines the priority of the job. The *priority* argument must be an integer between -1024 (lowest priority) and +1023 (highest priority) inclusive. The default is no priority which is equivalent to a priority of zero.

This option allows the user to specify a priority for their jobs. However, this option is dependant upon the local scheduling policy. By default the “sort jobs by job-priority” feature is disabled. If your local PBS administrator has enabled it, then all queued jobs will be sorted based on the user-specified priority. (If you need an absolute ordering of your own jobs, see ["Specifying Job Dependencies" on page 195.](#))

```
qsub -p 120 my_job
#PBS -p -300
```

3.13.11 Specifying a Job's Project

In PBS, a project is a way to organize jobs independently of users and groups. A project is a tag that identifies a set of jobs. Each job's `project` attribute specifies the job's project. Each job can be a member of up to one project.

Projects are not tied to users or groups. One user or group may run jobs in more than one project. For example, user Bob runs JobA in ProjectA and JobB in ProjectB. User Bill runs JobC in ProjectA. User Tom runs JobD in ProjectB. Bob and Tom are in Group1, and Bill is in Group2.

A job's project can be set in the following ways:

- At submission, using the `qsub -P` option; see [“qsub” on page 210 of the PBS Professional Reference Guide](#)
- After submission, via the `qalter -P` option; see [“qalter” on page 128 of the PBS Professional Reference Guide](#)

3.13.12 Deferring Execution

The `-a date_time` option declares the time after which the job is eligible for execution. The `date_time` argument is in the form:

```
[[[[CC]YY]MM]DD]hhmm[.SS]
```

where

CC is the first two digits of the year (the century)

YY is the second two digits of the year

MM is the two digits for the month

DD is the day of the month

hh is the hour

mm is the minute

The optional *SS* is the seconds

If the month, *MM*, is not specified, it will default to the current month if the specified day *DD*, is in the future. Otherwise, the month will be set to next month. Likewise, if the day, *DD*, is not specified, it will default to today if the time *hhmm* is in the future. Otherwise, the day will be set to tomorrow. For example, if you submit a job at 11:15am with a time of “1110”, the job will be eligible to run at 11:10am tomorrow. Other examples include:

```
qsub -a 0700 my_job
#PBS -a 10220700
```

The job is in the wait (W) state from the time it is submitted until the time it is eligible for execution.

3.13.13 Holding a Job (Delaying Execution)

The “-h” option specifies that a *user hold* be applied to the job at submission time. The job will be submitted, then placed in a hold state. The job will remain ineligible to run until the hold is released. (For details on releasing a held job see ["Holding and Releasing Jobs" on page 156.](#))

```
qsub -h my_job
#PBS -h
```

3.13.14 Specifying Job Checkpoint Interval

3.13.14.1 Checkpointable Jobs

A job is checkpointable if any of the following is true:

- Its application supports checkpointing and there are checkpoint scripts
- There is a third-party checkpointing application available
- The OS supports checkpointing

Checkpoint scripts are set up by the local system administrator.

3.13.14.2 Queue Checkpoint Intervals

The execution queue in which the job resides controls the minimum interval at which a job can be checkpointed. The interval is specified in CPU minutes or walltime minutes. The same value is used for both, so for example if the minimum interval is specified as 12, then a job using the queue’s interval for CPU time will be checkpointed every 12 minutes of CPU time, and a job using the queue’s interval for walltime will be checkpointed every 12 minutes of walltime.

3.13.14.3 Checkpoint Interval

The “-c *checkpoint-spec*” option defines the interval, in CPU minutes, or in walltime minutes, at which the job will be checkpointed.

The *checkpoint-spec* argument is specified as:

c

Checkpointing is to be performed according to the interval, measured in CPU time, set on the execution queue in which the job resides.

c=<minutes of CPU time>

Checkpointing is to be performed at intervals of the specified number of minutes of CPU time used by the job. This value must be greater than zero. If the interval specified is less than that set on the execution queue in which the job resides, the queue's interval is used.

Format: Integer

w

Checkpointing is to be performed according to the interval, measured in walltime, set on the execution queue in which the job resides.

w=<minutes of walltime>

Checkpointing is to be performed at intervals of the specified number of minutes of walltime used by the job. This value must be greater than zero. If the interval specified is less than that set on the execution queue in which the job resides, the queue's interval is used.

Format: Integer

n

No checkpointing is to be performed.

s

Checkpointing is to be performed only when the Server executing the job is shut down.

u

Checkpointing is unspecified, thus resulting in the same behavior as "s".

If "-c" is not specified, the checkpoint attribute is set to the value "u".

```
qsub -c c my_job
```

```
#PBS -c c=10
```

Checkpointing is not supported for job arrays.

3.13.15 Specifying Job User ID

PBS requires that a user's name be consistent across a server and its execution hosts, but not across a submission host and a server. A user may have access to more than one server, and may have a different username on each server. In this environment, if a user wishes to submit

a job to any of the available servers, the user specifies the username to be used at each server. The wildcard username will be used if the job ends up at yet another server not specified, but only if that wildcard username is valid.

Example 3-4: Our user is UserS on the submission host HostS, UserA on server ServerA, and UserB on server ServerB, and is UserC everywhere else. Note that this user must be UserA on all ExecutionA and UserB on all ExecutionB machines. Then our user can use “qsub -u UserA@ServerA,UserB@ServerB,UserC” for the job. The job owner will always be UserS. On UNIX, UserA, UserB, and UserC must each have `.rhosts` files at their servers that list UserS.

Username are limited to 15 characters.

3.13.15.1 qsub -u: User ID with UNIX

The server’s `flatuid` attribute determines whether it assumes that identical usernames mean identical users. If true, it assumes that if UserS exists on both the submission host and the server host, then UserS can run jobs on that server. If not true, the server calls `ruserok()` which uses `/etc/hosts.equiv` or `.rhosts` to authorize UserS to run as UserS. In this case, the user whose name is specified in with the `-u` option must have a `.rhosts` file on the server’s host listing the job owner, meaning that UserS at the server must have a `.rhosts` file listing UserS.

Example 3-5: Our user is UserA on the submission host, but is userB at the server. In order to submit jobs as UserA and run jobs as UserB, UserB must have a `.rhosts` file on the server’s host that lists UserA.

Table 3-4: UNIX User ID and flatuid

Value of flatuid	Submission host username/server host username	
	Same: UserS/UserS	Different: UserS/UserA
True	Server assumes user has permission to run job	Server checks whether UserS can run job as UserA
Not true	Server checks whether UserS can run job as UserS	Server checks whether UserS can run job as UserA

Note that if different names are listed via the `-u` option, then they are checked regardless of the value of `flatuid`.

Using `hosts.equiv` is not recommended.

3.13.15.2 **qsub -u: User ID with Windows**

Under Windows, if a user has a non-admin account, the server's `hosts.equiv` file is used to determine whether that user can run a job on a given server. For an admin account, `[PROFILE_PATH] \rhosts` is used, and the server's `acl_roots` attribute must be set to allow job submissions. Usernames containing spaces are allowed as long as the username length is no more than 15 characters, and the usernames are quoted when used in the command line.

Table 3-5: Requirements for Admin User to Submit Job

Location/Action	Submission host username/Server host username	
	Same: UserS/UserS	Different: UserS/UserA
<code>[PROFILE_PATH] \rhosts</code> contains	For UserS on ServerA, add <code><HostS> UserS</code>	For UserA on ServerA, add <code><HostS> UserS</code>
set ServerA's <code>acl_roots</code> attribute	<code>qmgr> set server acl_roots=UserS</code>	<code>qmgr> set server acl_roots=UserA</code>

Table 3-6: Requirements for Non-admin User to Submit Job

File	Submission host username/Server host username	
	Same: UserS/UserS	Different: UserS/UserA
<code>hosts.equiv</code> on ServerA	<code><HostS></code>	<code><HostS> UserS</code>

3.13.16 **Setting Job Attribute Values**

You can use the `-W <attribute>=<value>` option to the `qsub` command to set any job attribute. This option duplicates the function of several other `qsub` options. For example, using `"-e <path>"` or `"-W Error_Path=<path>"` has the same effect.

Avoid duplicating other `qsub` options when using the `-W` option.

3.13.17 Specifying Job Group ID

The “-W *group_list=g_list*” option defines the group name under which the job is to run on the execution system. The *g_list* argument is of the form:

```
group[@host][,group[@host],...]
```

Only one group name may be given per specified host. Only one of the group specifications may be supplied without the corresponding host specification. That group name will be used for execution on any host not named in the argument list. If not set, the *group_list* defaults to the primary group of the user under which the job will be run. Under Windows, the primary group is the first group found for the user by PBS when querying the accounts database.

```
qsub -W group_list=grpA,grpB@jupiter my_job
```

3.13.18 Specifying a Local Account

The “-A *account_string*” option defines the account string associated with the job. The *account_string* is an opaque string of characters and is not interpreted by the Server which executes the job. This value is often used by sites to track usage by locally defined account names.

IMPORTANT:

Under Unicos, if the Account string is specified, it must be a valid account as defined in the system “User Data Base”, UDB.

```
qsub -A Math312 my_job
#PBS -A accountNumber
```

3.13.19 Suppressing Job Identifier

The “-z” option directs the *qsub* command to not write the job identifier assigned to the job to the command’s standard output.

```
qsub -z my_job
#PBS -z
```

3.13.20 Specifying Staging and Execution Directory

The -W *sandbox=<value>* option allows you to specify where PBS will stage files and execute the job script. See [section 8.6, “Input/Output File Staging”, on page 198](#).

3.13.21 Interactive-batch Jobs

PBS provides a special kind of batch job called *interactive-batch*. An interactive-batch job is treated just like a regular batch job (in that it is queued up, and has to wait for resources to become available before it can run). Once it is started, however, the user's terminal input and output are connected to the job in a manner similar to a `login` session. It appears that the user is logged into one of the available execution machines, and the resources requested by the job are reserved for that job. Many users find this useful for debugging their applications or for computational steering. The “`qsub -I`” option declares that the job is an interactive-batch job.

Interactive jobs can use provisioning.

If the `qsub -I` option is specified on the command line, the job is an interactive job. If a script is given, it will be processed for directives, but any executable commands will be discarded. When the job begins execution, all input to the job is from the terminal session in which `qsub` is running. The `-I` option is ignored in a script directive.

When an interactive job is submitted, the `qsub` command will not terminate when the job is submitted. `qsub` will remain running until the job terminates, is aborted, or the user interrupts `qsub` with a SIGINT (the control-C key). If `qsub` is interrupted prior to job start, it will query if the user wishes to exit. If the user responds “yes”, `qsub` exits and the job is aborted.

Once the interactive job has started execution, input to and output from the job pass through `qsub`. Keyboard-generated interrupts are passed to the job. Lines entered that begin with the tilde (~) character and contain special sequences are interpreted by `qsub` itself. The recognized special sequences are:

- ~. `qsub` terminates execution. The batch job is also terminated.
- ~susp If running under the UNIX C shell, suspends the `qsub` program. “susp” is the suspend character, usually CNTL-Z.
- ~asusp If running under the UNIX C shell, suspends the input half of `qsub` (terminal to job), but allows output to continue to be displayed. “asusp” is the auxiliary suspend character, usually control-Y.

3.13.21.1 Caveats

- Interactive-batch jobs are not supported on Windows.
- Interactive-batch jobs do not support job arrays.
- Interactive jobs are not rerunnable.

3.13.22 Receiving X Output from Interactive Jobs

You can receive X output from an interactive job.

3.13.22.1 How to Receive X Output

To receive X output, use `qsub -X -I`.

3.13.22.2 Requirements for Receiving X Output

- The job must be interactive: you must also specify either `-I` or `-W interactive = true`.
- An X server must be running on the system where you want to see the X output.
- The `DISPLAY` variable in the job's submission environment must be set to the display where the X output is desired.
- Your administrator must configure MOM's `PATH` to include the `xauth` utility.

3.13.22.3 Viewing X Output Job Attributes

Each job has two read-only attributes containing X forwarding information. These are the following:

`forward_x11_cookie`

This attribute contains the X authorization cookie.

`forward_x11_port`

This attribute contains the number of the port being listened to by the port forwarder on the submission host.

You can view these attributes using `qstat -f <job ID>`.

3.13.22.4 Caveats for Receiving X Output

- This option is not available under Windows.
- If you use the `qsub -v` option, PBS will handle the `DISPLAY` variable correctly.
- If you use the `qsub -v DISPLAY` option, you will get an error.
- At most 25 concurrent X applications can run using the same job session.

3.13.22.5 X Forwarding Errors

- If the `DISPLAY` environment variable is pointing to a display number that is correctly

formatted but incorrect, submitting an interactive X forwarding job results in the following error message:

```
"cannot read data from 'xauth list <display number>', errno=<errno>"
```

- If the `DISPLAY` environment variable is pointing to an incorrectly formatted value, submitting an interactive X forwarding job results in the following error message:

```
"qsub: Failed to get xauth data (check $DISPLAY variable)"
```
- If the X authority utility (`xauth`) is not found on the submission host, the following error message is displayed:

```
"execution of xauth failed: sh: xauth: command not found"
```
- When the execution of the `xauth` utility results in an error, the error message displayed by the `xauth` utility is preceded by the following:

```
"execution of xauth failed: "
```
- When the `qsub -X` option is used without `-I` or `-W interactive=true`, the following error message is displayed:

```
"qsub: X11 forwarding possible only for Interactive Jobs"
```

3.14 Failed Jobs

Once a job has experienced a certain number of failures, PBS holds the job.

If requeueing a job fails, the job is deleted.

Chapter 4

Multiprocessor Jobs

4.1 Submitting Multiprocessor Jobs

4.1.1 Assigning the Chunks You Want

PBS assigns chunks to job processes in the order in which the chunks appear in the select statement. PBS takes the first chunk from the primary execution host; this is where the top task of the job runs.

Example 4-1: You want three chunks, where the first has two CPUs and 20 GB of memory, the second has four CPUs and 100 GB of memory, and the third has one CPU and five GB of memory:

```
-lselect=1:ncpus=2:mem=20gb+ncpus=4:mem=100gb+mem=5gb
```

4.1.2 Placing Your Job on Specific Vnodes

Placement sets allow partitioning of the vnodes in the complex according to the values of one or more resources, so that a vnode may be in multiple placement sets: one set that share a value for one resource, and another set that share a different value for a different resource. By default, placement sets attempt to group vnodes that are “close to” each other. If your job doesn’t request a specific placement, it may be run in a placement set. See [section 4.8.32, “Placement Sets” on page 205 in the PBS Professional Administrator’s Guide](#).

Your job can request grouping according to a specific resource; see [section 3.6.2, “Using `place=group`”, on page 45](#). If your job requests grouping by a resource, i.e. `place=group=resource`, then the chunks are placed as requested and placement sets are ignored.

If a job requests grouping but no group contains the required number of vnodes, grouping is ignored.

4.1.3 The Job's Node File

For each job, PBS creates a job-specific “host file” or “node file”, which is a text file containing the name(s) of the host(s) containing the vnode(s) allocated to that job. The file is created by the MOM on the primary execution host, and is available only on that host.

4.1.3.1 Node File Format and Contents

The node file contains a list of host names, one per line. The name of the host is the value in `resources_available.host` of the allocated vnode(s). The order in which hosts appear in the PBS node file is the order in which chunks are specified in the selection directive.

The node file contains one line per MPI process with the name of the host on which that process should execute. The number of MPI processes for a job, and the contents of the node file, are controlled by the value of the resource `mpiprocs`.

For each chunk requesting `mpiprocs=M`, the name of the host from which that chunk is allocated is written in the node file *M* times. Therefore the number of lines in the node file is the sum of requested `mpiprocs` for all chunks requested by the job.

Example 4-2: Two MPI processes run on HostA and one MPI process runs on HostB. The node file looks like this:

```
HostA
HostA
HostB
```

4.1.3.2 Name and Location of Node File

The file is created by the MoM on the primary execution host, in `PBS_HOME/aux/JOB_ID`, where *JOB_ID* is the job identifier for that job.

The full path and name for the node file is set in the job's environment, in the environment variable `PBS_NODEFILE`.

4.1.3.3 Node File for Old-style Requests

For jobs which request resources using the old *-lnodes=nodespec* format, the host for each vnode allocated to the job is listed *N* times, where *N* is the number of MPI ranks on the vnode. The number of MPI ranks is specified via the `ppn` resource.

Example 4-3: Request four vnodes, each with two MPI processes, where each process has three threads, and each thread has a CPU:

```
qsub -lnodes=4:ncpus=3:ppn=2
```

This results in each of the four hosts being written twice, in the order in which the vnodes are assigned to the job.

4.1.3.4 Using and Modifying the Node File

You can use `$PBS_NODEFILE` in your job script.

You can modify the node file. You can remove entries or sort the entries. PBS does not use the contents of the node file.

4.1.3.5 Node File Caveats

Do not add entries for new hosts; PBS may terminate processes on those hosts because PBS does not expect the processes to be running there. Adding entries on the same host may cause the job to be terminated because it is using more CPUs than it requested.

4.1.4 Specifying Number of MPI Processes Per Chunk

How you request chunks matters. First, the number of MPI processes per chunk defaults to *1* for chunks with CPUs, and *0* for chunks without CPUs, unless you specify this value using the `mpiprocs` resource. Second, you can specify whether MPI processes share CPUs. For example, requesting one chunk with four CPUs and four MPI processes is not the same as requesting four chunks each with one CPU and one MPI process. In the first case, all four MPI processes are sharing all four CPUs. In the second case, each process gets its own CPU.

You request the number of MPI processes you want for each chunk using the `mpiprocs` resource. For example, to request two MPI processes for each of four chunks, where each chunk has two CPUs:

```
-lselect=4:ncpus=2:mpiprocs=2
```

If you don't explicitly request a value for the `mpiprocs` resource, it defaults to `1` for each chunk requesting CPUs, and `0` for chunks not requesting CPUs.

Example 4-4: To request one chunk with two MPI processes and one chunk with one MPI process, where both chunks have two CPUs:

```
-lselect=ncpus=2:mpiprocs=2+ncpus=2
```

Example 4-5: A request for three vnodes, each with one MPI process:

```
qsub -l select=3:ncpus=2
```

This results in the following node file:

```
<hostname for 1st vnode>
```

```
<hostname for 2nd vnode>
```

```
<hostname for 3rd vnode>
```

Example 4-6: If you want to run two MPI processes on each of three hosts and have the MPI processes share a single processor on each host, request the following:

```
-lselect=3:ncpus=1:mpiprocs=2
```

The node file then contains the following list:

```
hostname for VnodeA
```

```
hostname for VnodeA
```

```
hostname for VnodeB
```

```
hostname for VnodeB
```

```
hostname for VnodeC
```

```
hostname for VnodeC
```

Example 4-7: If you want three chunks, each with two CPUs and running two MPI processes, use:

```
-l select=3:ncpus=2:mpiprocs=2...
```

The node file then contains the following list:

```
hostname for VnodeA
```

```
hostname for VnodeA
```

```
hostname for VnodeB
```

```
hostname for VnodeB
```

```
hostname for VnodeC
```

```
hostname for VnodeC
```

Notice that the node file is the same as the previous example, even though the number of CPUs used is different.

Example 4-8: If you want four MPI processes, where each process has its own CPU:

```
-lselect=4:ncpus=1
```

See [“Built-in Resources” on page 299 of the PBS Professional Reference Guide](#) for a definitions of the `mpiprocs` resource.

4.1.4.1 Chunks With No MPI Processes

If you request a chunk that has no MPI processes, PBS may take that chunk from a vnode which has already supplied another chunk. You request a chunk that has no MPI processes using either of the following:

```
-lselect=1:ncpus=0
```

```
-lselect=1:ncpus=2:mpiprocs=0
```

4.1.5 Caveats and Advice for Multiprocessor Jobs

4.1.5.1 Requesting Uniform Processors

Some MPI jobs require the work on all vnodes to be at the same stage before moving to the next stage. For these applications, the work can proceed only at the pace of the slowest vnode, because faster vnodes must wait while it catches up. In this case, you may find it useful to ensure that the job’s vnodes are homogeneous.

If there is a resource that identifies the architecture, type, or speed of the vnodes, you can use it to ensure that all chunks are taken from vnodes with the same value. You can either request a specific value for this resource for all chunks, or you can group vnodes according to the value of the resource. See [section 3.6.2, “Using `place=group`”, on page 45](#).

Example 4-9: The resource that identifies the speed is named *speed*, and your job requests 16 chunks, each with two CPUs, two MPI processes, all with *speed* equal to *fast*:

```
-lselect=16:ncpus=2:mpiprocs=2:speed=fast
```

Example 4-10: Request 16 chunks where each chunk has two CPUs, using grouping to ensure that all chunks share the same speed. The resource that identifies the speed is named *speed*:

```
-lselect=16:ncpus=2:mpiprocs=2:place=group=speed
```

4.1.5.2 Requesting Storage on NFS Server

One of the vnodes in your complex may act as an NFS server to the rest of the vnodes, so that all vnodes have access to the storage on the NFS server.

Example 4-11: The `scratch` resource is shared among all the vnodes in the complex, and is requested from a central location, called the “*nfs_server*” vnode. To request two vnodes, each with two CPUs to do calculations, and one vnode with 10gb of memory and no MPI processes:

```
-l select=2:ncpus=2+1:host=nfs_server:scratch=10gb:ncpus=0
```

With this request, your job has one MPI process on each chunk containing CPUs, and no MPI processes on the memory-only chunk. The job shows up as having a chunk on the “*nfs_server*” host.

4.1.6 File Staging for Multiprocessor Jobs

PBS stages files to and from the primary execution host only.

4.1.7 Prologue and Epilogue

The prologue is run as root on the primary host, with the current working directory set to `PBS_HOME/mom_priv`, and with `PBS_JOBDIR` and `TMPDIR` set in its environment.

PBS runs the epilogue as root on the primary host. The epilogue is executed with its current working directory set to the job's staging and execution directory, and with `PBS_JOBDIR` and `TMPDIR` set in its environment.

4.1.8 MPI Environment Variables

NCPUS

PBS sets the `NCPUS` environment variable in the job's environment on the primary execution host. PBS sets `NCPUS` to the value of `ncpus` requested for the first chunk.

OMP_NUM_THREADS

PBS sets the `OMP_NUM_THREADS` environment variable in the job's environment on the primary execution host. PBS sets this variable to the value of `ompthreads` requested for the first chunk, which defaults to the value of `ncpus` requested for the first chunk.

4.1.9 Examples of Multiprocessor Jobs

Example 4-12: For a 10-way MPI job with 2gb of memory per MPI task:

```
qsub -l select=10:ncpus=1:mem=2gb
```

Example 4-13: If you have a cluster of small systems with for example two CPUs each, and you wish to submit an MPI job that will run on four separate hosts:

```
qsub -l select=4:ncpus=1 -l place=scatter
```

In this example, the node file contains one entry for each of the hosts allocated to the job, which is four entries.

The variables NCPUS and OMP_NUM_THREADS are set to one.

Example 4-14: If you do not care where the four MPI processes are run:

```
qsub -l select=4:ncpus=1 -l place=free
```

Here, the job runs on two, three, or four hosts depending on what is available.

For this example, the node file contains four entries. These are either four separate hosts, or three hosts, one of which is repeated once, or two hosts, etc.

NCPUS and OMP_NUM_THREADS are set to 1, the number of CPUs allocated from the first chunk.

4.1.10 Submitting SMP Jobs

To submit an SMP job, simply request a single chunk containing all of the required CPUs and memory, and if necessary, specify the hostname. For example:

```
qsub -l select=ncpus=8:mem=20gb:host=host1
```

When the job is run, the node file will contain one entry, the name of the selected execution host.

The job will have two environment variables, NCPUS and OMP_NUM_THREADS, set to the number of CPUs allocated.

4.2 Using MPI with PBS

4.2.1 Using an Integrated MPI

Many MPIs are integrated with PBS. PBS provides tools to integrate most of them; a few MPIs supply the integration. When a job is run under an integrated MPI, PBS can track resource usage, signal job processes, and perform accounting for all processes of the job.

When a job is run under an MPI that is not integrated with PBS, PBS is limited to managing the job only on the primary vnode, so resource tracking, job signaling, and accounting happen only for the processes on the primary vnode.

The instructions that follow are for integrated MPIs. Check with your administrator to find out which MPIs are integrated at your site. If an MPI is not integrated with PBS, you use it as you would outside of PBS.

Some of the integrated MPIs have slightly different command lines. See the instructions for each MPI.

The following table lists the supported MPIs and gives links to instructions for using each MPI:

Table 4-1: List of Supported MPIs

MPI Name	Versions	Instructions for Use
HP MPI	1.08.03 2.0.0	See section 4.2.4, “HP MPI with PBS”, on page 92
IBM POE	AIX 5.x, 6.x	See section 4.2.5, “IBM POE with PBS”, on page 93
Intel MPI	2.0.022 3 4	See section 4.2.6, “Intel MPI with PBS”, on page 100
LAM MPI	6.5.9	Deprecated. See section 4.2.7.2, “Using LAM 6.5.9 with PBS”, on page 105
LAM MPI	7.0.6 7.1.1	See section 4.2.7.1, “Using LAM 7.x with PBS”, on page 105

Table 4-1: List of Supported MPIs

MPI Name	Versions	Instructions for Use
MPICH-P4	1.2.5 1.2.6 1.2.7	See section 4.2.8, “MPICH-P4 with PBS”, on page 106
MPICH-GM		See section 4.2.9, “MPICH-GM with PBS”, on page 108
MPICH-MX		See section 4.2.10, “MPICH-MX with PBS”, on page 111
MPICH2	1.0.3 1.0.5 1.0.7	See section 4.2.11, “MPICH2 with PBS”, on page 115
MVAPICH	1.2	See section 4.2.12, “MVAPICH with PBS”, on page 119
MVAPICH2	1.8	See section 4.2.13, “MVAPICH2 with PBS”, on page 121
Open MPI	1.4.x	See section 4.2.14, “Open MPI with PBS”, on page 124
Platform MPI	8.0	See section 4.2.15, “Platform MPI with PBS”, on page 124
SGI MPT	Any	See section 4.2.16, “SGI MPT with PBS”, on page 124

4.2.1.1 Integration Caveats

- Under Windows, MPIs are not integrated with PBS. PBS is limited to tracking resources, signaling jobs, and performing accounting only for job processes on the primary vnode.
- Some MPI command lines are slightly different; the differences for each are described.

4.2.1.2 Integrating an MPI on the Fly using the **pbs_tmrsh** Command

The PBS administrator can perform the steps to integrate the supported MPIs. For non-integrated MPIs, you can integrate them on the fly using the **pbs_tmrsh** command. You should not use **pbs_tmrsh** with an integrated MPI.

This command emulates **rsh**, but uses the PBS TM interface to talk directly to **pbs_mom** on sister vnodes. The **pbs_tmrsh** command informs the primary and sister MoMs about job processes on sister vnodes. When the job uses **pbs_tmrsh**, PBS can track resource usage for all job processes.

You use **pbs_tmrsh** as your **rsh** or **ssh** command. To use **pbs_tmrsh**, set the appropriate environment variable to *pbs_tmrsh*. For example, to integrate MPICH, set the **P4_RSHCOMMAND** environment variable to *pbs_tmrsh*, and to integrate HP MPI, set **MPI_REMSH** to *pbs_tmrsh*.

The following figure illustrates how the `pbs_tmsh` command can be used to integrate an MPI on the fly:

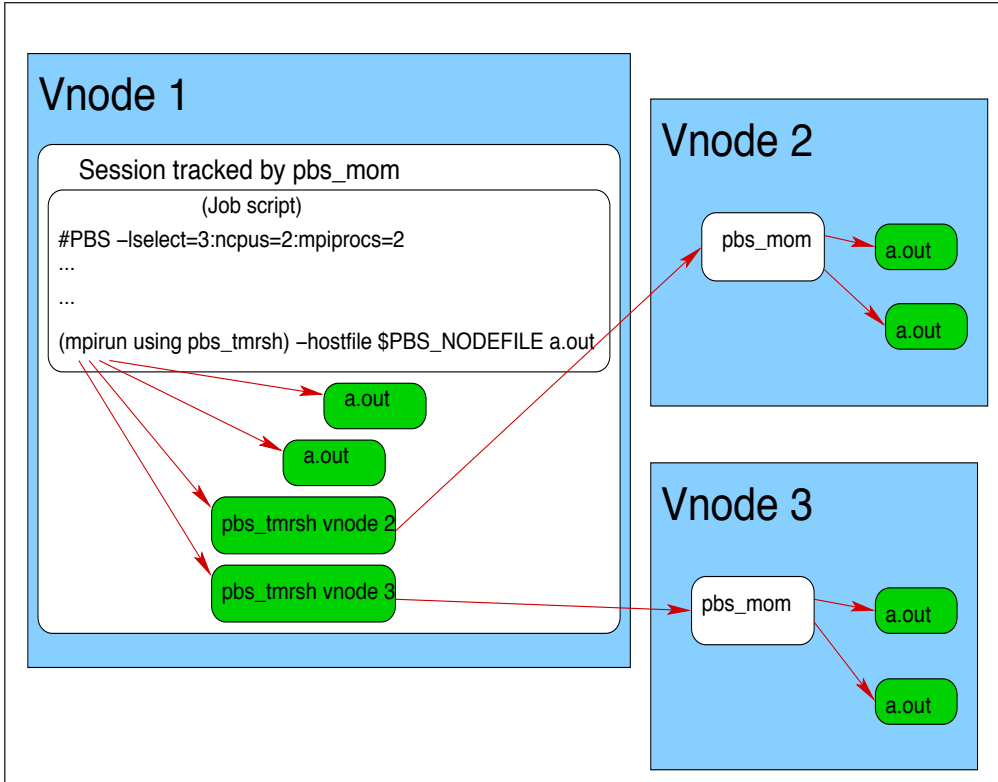


Figure 4-1: PBS knows about processes on vnodes 2 and 3, because `pbs_tmsh` talks directly to `pbs_mom`, and `pbs_mom` starts the processes on vnodes 2 and 3

4.2.1.2.i Caveats for the `pbs_tmsh` Command

- This command cannot be used outside of a PBS job; if used outside a PBS job, this command will fail.
- The `pbs_tmsh` command does not perform exactly like `rsh`. For example, you cannot pipe output from `pbs_tmsh`; this will fail.

4.2.2 Prerequisites to Using MPI with PBS

The MPI that you intend to use with PBS must be working before you try to use it with PBS. You must be able to run an MPI job outside of PBS.

4.2.3 Caveats for Using MPIs

Some applications write scratch files to a temporary location in `tmpdir`. The location of `tmpdir` is host-dependent. If you are using an MPI other than LAM MPI or Open MPI, and your application needs scratch space, the location of `tmpdir` for the job should be consistent across execution hosts. Your PBS administrator can specify `tmpdir` for each host. The job's `TMPDIR` environment variable can also affect `tmpdir`, but `TMPDIR` is overridden by the administrator's setting.

4.2.4 HP MPI with PBS

HP MPI can be integrated with PBS on UNIX and Linux so that PBS can track resource usage, signal processes, and perform accounting, for all job processes. Your PBS administrator can integrate HP MPI with PBS.

4.2.4.1 Setting up Your Environment for HP MPI

In order to override the default `rsh`, set `PBS_RSHCOMMAND` in your job script:

```
export PBS_RSHCOMMAND=<rsh choice>
```

4.2.4.2 Using HP MPI with PBS

You can run jobs under PBS using HP MPI without making any changes to your MPI command line.

4.2.4.3 Options

When running a PBS HP MPI job, you can use the same arguments to the `mpirun` command as you would outside of PBS. The following options are treated differently under PBS:

- `-h <host>`
Ignored
- `-l <user>`
Ignored
- `-np <number>`
Modified to fit the available resources

4.2.4.4 Caveats for HP MPI with PBS

Under the integrated HP MPI, the job's working directory is changed to the user's home directory.

4.2.5 IBM POE with PBS

When you are using AIX machines running IBM's Parallel Operating Environment, or POE, you can run PBS jobs using either the HPS or InfiniBand, whichever is available. You can use either IP or US mode. PBS manages InfiniBand or the HPS. LoadLeveler is not required in order to use InfiniBand switches in User Space mode.

PBS can track the resources for MPI, LAPI programs or a mix of MPI and LAPI programs.

Any job that can run under IBM `poe` can run under PBS. There are some exceptions and differences; under PBS, the `poe` command is slightly different. See [section 4.2.5.5, “poe Options and Environment Variables”](#), on page 95.

4.2.5.1 Using the InfiniBand Switch

To ensure that a job uses the InfiniBand switch, make sure that the job's environment has `PBS_GET_IBWINS` set to 1. This can be accomplished the following ways:

- The administrator sets this value for all jobs.
- You can set the environment variable for each job: set `PBS_GET_IBWINS = 1` in your shell environment, and use the `-V` option to every `qsub` command. See the previous section.
 - `csch`:

```
setenv PBS_GET_IBWINS 1
```
 - `bash`:

```
PBS_GET_IBWINS = 1
export PBS_GET_IBWINS
```
- You can set the environment variable for one job; use the `“-v PBS_GET_IBWINS = 1”` option to the `qsub` command.

4.2.5.2 Using the HPS

If an HPS is available on the AIX machine where your job runs, PBS runs your jobs so that they use the HPS.

In order to make sure that your job runs on this machine, you can request the resource representing the HPS. We recommend that this resource is called *hps*. We recommend that this resource is a host-level Boolean defined on each host on the HPS; check with your administrator.

4.2.5.3 Specifying Number of Ranks

Make sure that you request the number of MPI ranks that you want, since PBS calculates the number of windows based on the number of ranks. You can use the `mpiprocs` resource to specify the number of MPI processes for each chunk. See [section 4.1.4, “Specifying Number of MPI Processes Per Chunk”, on page 83](#).

Example 4-15: To request two vnodes, each with eight CPUs and one MPI rank, for a total of 16 CPUs and two ranks:

```
select=2:ncpus=8
```

Example 4-16: To request two vnodes, each with eight CPUs and eight MPI ranks, for a total of 16 CPUs and 16 ranks:

```
select=2:ncpus=8:mpiprocs=8
```

4.2.5.3.i If Your Complex Contains HPS and Non-HPS Machines

If your complex contains machines on the HPS and machines that are not on the HPS, and you wish to run on the HPS, you must specify machines on the HPS. To specify machines on the HPS, you must request the HPS resource in your select statement. This resource is configured by your PBS administrator. We recommend that this resource is a host-level Boolean, but it could be an integer; check with your PBS administrator.

Example 4-17: Request four chunks using `place=scatter`. The HPS resource is a Boolean called `hps`. Each host must have `hps=True`:

```
% qsub -l select=4:ncpus=2:hps=true -lplace=scatter
```

Example 4-18: Same placement as previous example; request four chunks using `place=pack`. Only one host is used, and you can have each chunk request the HPS. The HPS resource is a Boolean called `hps`:

```
% qsub -l select=4:ncpus=2:hps=true -l place=pack
```

If your PBS administrator has configured a host-level integer resource instead of a Boolean resource, make sure that you request the correct value for this resource; see your PBS administrator.

4.2.5.4 Restrictions on `poe` Jobs

- Outside of PBS, you can run `poe`, but you will see this warning:
`pbsrun.poe: Warning, not running under PBS`
- Inside PBS, you cannot run `poe` jobs without arguments. Attempting to do this will give

the following error:

```
pbsrun.poe: Error, interactive program name entry not supported under PBS
poe exits with a value of 1.
```

- Some environment variables and options to `poe` behave differently under PBS. These differences are described in the next section.
- The maximum number of ranks that can be launched is the number of entries in `$PBS_NODEFILE`.

4.2.5.5 `poe` Options and Environment Variables

The usage for `poe` is:

```
poe [program] [program_options] [poe options]
```

When submitting jobs to `poe`, you can set environment variables instead of using options to `poe`. The equivalent environment variable is listed with its `poe` option. All options and environment variables except the following are passed to `poe`:

`-devtype`, `MP_DEVTYPE`

If InfiniBand is not specified in either the option or the environment variable, US mode is not used for the job.

`-euiddevice`, `MP_EUIDEVICE`

Ignored by PBS.

`-eulib {ip|us}`, `MP_EULIB`

If set to `us`, the job runs in User Space mode.

If set to any other value, that value is passed to IBM `poe`.

If the command line option `-eulib` is set, it takes precedence over the `MP_EULIB` environment variable.

`-hostfile`, `-hfile`, `MP_HOSTFILE`

Ignored. If this is specified, PBS prints the following:

```
pbsrun.poe: Warning, -hostfile value replaced by PBS
```

or

```
pbsrun.poe: Warning -hfile value replaced by PBS
```

If this environment variable is set when a `poe` job is submitted, PBS prints the following error message:

```
pbsrun.poe: Warning MP_HOSTFILE value replaced by PBS
```

`-instances`, `MP_INSTANCES`

The option and the environment variable are treated differently:

-instances

If the option is set, PBS prints a warning:

```
pbsrun.poe: Warning, -instances cmd line option removed by
PBS
```

MP_INSTANCES

If the environment variable is set, PBS uses it to calculate the number of network windows for the job.

The maximum value allowed can be requested by using the string “max” for the environment variable.

If the environment variable is set to a value greater than the maximum allowed value, it is replaced with the maximum allowed value.

The default maximum value is 4.

-procs, MP_PROCS

This option or environment variable should be set to the total number of mpiprocs requested by the job when using US mode.

If neither this option nor the MP_PROCS environment variable is set, PBS uses the number of entries in \$PBS_NODEFILE.

If this option is set to N , and the job is submitted with a total of M mpiprocs:

If $N \geq M$: The value N is passed to IBM poe.

If $N < M$ and US mode is not being used: The value N is passed to poe.

If $N < M$ and US mode is being used: US mode is turned off and a warning is printed:

```
pbsrun.poe: Warning, user mode disabled due to MP_PROCS setting
```

4.2.5.6 Caveats for POE**4.2.5.6.i Multi-host Jobs on POE**

If you wish to run a multi-host job, it must not run on a mix of InfiniBand and non-InfiniBand hosts. It can run entirely on hosts that are non-InfiniBand, or on hosts that are all using InfiniBand, but not both.

4.2.5.6.ii Maximum Number of Ranks on POE

The maximum number of ranks that can be launched under integrated POE is the number of entries in \$PBS_NODEFILE.

4.2.5.6.iii Run Jobs in Foreground on POE

Since PBS is tracking tasks started by `poe`, these tasks are counted towards your run limits. Running multiple `poe` jobs in the background will not work. Instead, run `poe` jobs one after the other or submit separate jobs. Otherwise switch windows will be used by more than one task. The `tracejob` command will show any of various error messages.

4.2.5.6.iv Job Submission Format on POE

Do not submit InfiniBand jobs in which the select statement specifies only a number, for example:

```
$ export PBS_GET_IBWINS=1
$ qsub -koe -mn -l select=1 -V jobname
```

Instead, use the equivalent request which specifies a resource:

```
$ export PBS_GET_IBWINS=1
$ qsub -koe -mn -l select=1:ncpus=1 -V jobname
```

4.2.5.6.v Environment Variables under POE

Do not set the `PBS_O_HOST` environment variable. If you do so, using the `qsub` command with the `-V` option will fail.

4.2.5.7 Useful Information

4.2.5.7.i IBM Documentation

For more information on using IBM's Parallel Operating Environment, see "IBM Parallel Environment for AIX 5L Hitchhiker's Guide".

4.2.5.7.ii Sources for Sample Code

When installing the `ppe.poe` fileset there are three directories containing sample code that may be of interest (from "How installing the POE fileset alters your system"):

- Directory containing sample code for running User Space POE jobs without LoadLeveler:
`/usr/lpp/ppe.poe/samples/swtbl`
- Directory containing sample code for running User Space jobs without LoadLeveler, using the network table API:
`/usr/lpp/ppe.poe/samples/ntbl`
- Directory that contains the sample code for running User Space jobs on InfiniBand interconnects, without LoadLeveler, using the network resource table API:
`/usr/lpp/ppe.poe/samples/nrt`

4.2.5.8 Examples Using poe

Example 4-19: Using IP mode, run a single executable poe job with four ranks on hosts spread across the PBS-allocated hosts listed in \$PBS_NODEFILE:

```
% cat $PBS_NODEFILE
host1
host2
host3
host4

% cat job.script
poe /path/mpiprogram -eulib ip

% qsub -l select=4:ncpus=1 -lplace=scatter
job.script
```

Example 4-20: Using US mode, run a single executable poe job with four ranks on hosts spread across the PBS-allocated hosts listed in \$PBS_NODEFILE:

```
% cat $PBS_NODEFILE
host1
host2
host3
host4

% cat job.script
poe /path/mpiprogram -eulib us
```

```
% qsub -l select=4:ncpus=1 -lplace=scatter
    job.script
```

Example 4-21: Using IP mode, run executables prog1 and prog2 with two ranks of prog1 on host1, two ranks of prog2 on host2 and two ranks of prog2 on host3:

```
% cat $PBS_NODEFILE
host1
host1
host2
host2
host3
host3

% cat job.script
echo prog1 > /tmp/poe.cmd
echo prog1 >> /tmp/poe.cmd
echo prog2 >> /tmp/poe.cmd
echo prog2 >> /tmp/poe.cmd
echo prog2 >> /tmp/poe.cmd
echo prog2 >> /tmp/poe.cmd
poe -cmdfile /tmp/poe.cmd -eulib ip
rm /tmp/poe.cmd
```

```
% qsub -l select=3:ncpus=2:mpiprocs=2 -l place=scatter job.script
```

Example 4-22: Using US mode, run executables prog1 and prog2 with two ranks of prog1 on host1, two ranks of prog2 on host2 and two ranks of prog2 on host3:

```
% cat $PBS_NODEFILE
host1
host1
host2
host2
host3
host3
```

```
% cat job.script
echo prog1 > /tmp/poe.cmd
echo prog1 >> /tmp/poe.cmd
echo prog2 >> /tmp/poe.cmd
echo prog2 >> /tmp/poe.cmd
echo prog2 >> /tmp/poe.cmd
echo prog2 >> /tmp/poe.cmd
poe -cmdfile /tmp/poe.cmd -eulib us
rm /tmp/poe.cmd

% qsub -l select=3:ncpus=2:mpiprocs=2 -l place=scatter job.script
```

4.2.6 Intel MPI with PBS

PBS provides an interface to Intel MPI's `mpirun`. If executed inside a PBS job, this allows for PBS to track all Intel MPI processes so that PBS can perform accounting and have complete job control. If executed outside of a PBS job, it behaves exactly as if standard Intel MPI's `mpirun` was used.

4.2.6.1 Using Intel MPI Integrated with PBS

You use the same `mpirun` command as you would use outside of PBS.

When submitting PBS jobs that invoke the PBS-supplied interface to `mpirun` for Intel MPI, be sure to explicitly specify the actual number of ranks or MPI tasks in the `qsub select` specification. Otherwise, jobs will fail to run with "too few entries in the machinefile".

For an example of this problem, specification of the following:

```
#PBS -l select=1:ncpus=1:host=hostA+1:ncpus=2:host=hostB
mpirun -np 3 /tmp/mytask
```

results in the following node file:

```
hostA
hostB
```

which conflicts with the "`-np 3`" specification in `mpirun` since only two MPD daemons are started.

The correct way is to specify either of the following:

```
#PBS -l select=1:ncpus=1:host=hostA+2:ncpus=1:host=hostB
#PBS -l select=1:ncpus=1:host=hostA+1:ncpus=2:host=hostB:mpiprocs=2
```

which causes the node file to contain:

```
hostA
hostB
hostB
```

and is consistent with "mpirun -np 3".

4.2.6.2 Options to Integrated Intel MPI

If executed inside a PBS job script, all of the options to the PBS interface are the same as for Intel MPI's `mpirun` except for the following:

-host, -ghost

For specifying the execution host to run on. Ignored.

-machinefile <file>

The file argument contents are ignored and replaced by the contents of `$PBS_NODEFILE`.

mpdboot option --totalnum=*

Ignored and replaced by the number of unique entries in `$PBS_NODEFILE`.

mpdboot option --file=*

Ignored and replaced by the name of `$PBS_NODEFILE`. The argument to this option is replaced by `$PBS_NODEFILE`.

Argument to `mpdboot option -f <mpd_hosts_file>` replaced by `$PBS_NODEFILE`.

-s

If the PBS interface to Intel MPI's `mpirun` is called inside a PBS job, Intel MPI's `mpirun -s` argument to `mpdboot` is not supported as this closely matches the `mpirun` option "`-s <spec>`". You can simply run a separate `mpdboot -s` before calling `mpirun`. A warning message is issued by the PBS interface upon encountering a `-s` option describing the supported form.

-np

If you do not specify a `-np` option, then no default value is provided by the PBS interface. It is up to the standard `mpirun` to decide what the reason-

able default value should be, which is usually 1. The maximum number of ranks that can be launched is the number of entries in \$PBS_NODEFILE.

4.2.6.3 MPD Startup and Shutdown

Intel MPI's `mpirun` takes care of starting and stopping the MPD daemons. The PBS interface to Intel MPI's `mpirun` always passes the arguments `-totalnum=<number of mpds to start>` and `-file=<mpd_hosts_file>` to the actual `mpirun`, taking its input from unique entries in \$PBS_NODEFILE.

4.2.6.4 Examples

Example 4-23: Run a single-executable Intel MPI job with six processes spread out across the PBS-allocated hosts listed in \$PBS_NODEFILE:

Node file:

```
pbs-host1
pbs-host1
pbs-host2
pbs-host2
pbs-host3
pbs-host3
```

Job script:

```
# mpirun takes care of starting the MPD
# daemons on unique hosts listed in
# $PBS_NODEFILE, and also runs the 6 processes
# on the 6 hosts listed in
# $PBS_NODEFILE; mpirun takes care of
# shutting down MPDs.
mpirun /path/myprog.x 1200
```

Run job script:

```
qsub -l select=3:ncpus=2:mpiprocs=2 job.script
<job-id>
```

Example 4-24: Run an Intel MPI job with multiple executables on multiple hosts using \$PBS_NODEFILE and mpiexec arguments to mpirun:

\$PBS_NODEFILE:

```
hostA
hostA
hostB
hostB
hostC
hostC
```

Job script:

```
# mpirun runs MPD daemons on hosts listed in $PBS_NODEFILE
# mpirun runs 2 instances of mpitest1
# on hostA; 2 instances of mpitest2 on
# hostB; 2 instances of mpitest3 on hostC.
# mpirun takes care of shutting down the
# MPDs at the end of MPI job run.
mpirun -np 2 /tmp/mpitest1 : -np 2 /tmp/mpitest2 : -np 2 /tmp/mpitest3
```


Run job script:

```
qsub -l select=3:ncpus=2:mpiprocs=2 job.script  
<job-id>
```

Example 4-25: Run an Intel MPI job with multiple executables on multiple hosts via the `-configfile` option and `$PBS_NODEFILE`:

`$PBS_NODEFILE`:

```
hostA  
hostA  
hostB  
hostB  
hostC  
hostC
```

Job script:

```
echo "-np 2 /tmp/mpitest1" >> my_config_file  
echo "-np 2 /tmp/mpitest2" >> my_config_file  
echo "-np 2 /tmp/mpitest3" >> my_config_file  
  
# mpirun takes care of starting the MPD daemons  
# config file says run 2 instances of mpitest1  
# on hostA; 2 instances of mpitest2 on  
# hostB; 2 instances of mpitest3 on hostC.  
# mpirun takes care of shutting down the MPD daemons.  
mpirun -configfile my_config_file  
  
# cleanup  
rm -f my_config_file
```

Run job script:

```
qsub -l select=3:ncpus=2:mpiprocs=2 job.script  
<job-id>
```

4.2.6.5 Restrictions

The maximum number of ranks that can be launched under integrated Intel MPI is the number of entries in `$PBS_NODEFILE`.

4.2.7 LAM MPI with PBS

LAM MPI can be integrated with PBS on UNIX and Linux so that PBS can track resource usage, signal processes, and perform accounting, for all job processes. Your PBS administrator can integrate LAM MPI with PBS.

4.2.7.1 Using LAM 7.x with PBS

You can run jobs under PBS using LAM 7.x without making any changes to your `mpirun` call.

4.2.7.2 Using LAM 6.5.9 with PBS

Support for LAM 6.5.9 is **deprecated**. You can run jobs under PBS using LAM 6.5.9.

4.2.7.2.i Caveats for LAM 6.5.9 with PBS

- If you specify the `bhost` argument, PBS will print a warning saying that the `bhost` argument is ignored by PBS.
- If you do not specify the `where` argument, `pbs_mpi1am` will try to run the your program on all available CPUs using the `C` keyword.

4.2.7.3 Example Job Submission Script

The following is a simple PBS job script for use with LAM MPI:

```
#!/bin/bash

# Job Name
#PBS -N LamSubTest
# Merge output and error files
#PBS -j oe
# Select 2 nodes with 1 CPU each
#PBS -l select=2:ncpus=1
# Export Users Environmental Variables to Execution Host
#PBS -V
# Send email on abort, begin and end
#PBS -m abe
# Specify mail recipient
#PBS -M username@example.com

cd $PBS_O_WORKDIR

date
lamboot -v $PBS_NODEFILE
mpirun -np $(cat $PBS_NODEFILE|wc -l) ./ANY_C_MPI_CODE_HERE
date
```

When using the integrated `lamboot` in a job script, `lamboot` takes input from `$PBS_NODEFILE` automatically, so the argument is not necessary.

4.2.7.4 See Also

For information on LAM MPI, see www.lam-mpi.org/.

4.2.8 MPICH-P4 with PBS

MPICH-P4 can be integrated with PBS on UNIX and Linux so that PBS can track resource usage, signal processes, and perform accounting, for all job processes. Your PBS administrator can integrate MPICH-P4 with PBS.

4.2.8.1 Options for MPICH-P4 with PBS

Under PBS, the syntax and arguments for the MPICH-P4 `mpirun` command on Linux are the same except for one option, which you should not set:

`-machinefile file`

PBS supplies the machinefile. If you try to specify it, PBS prints a warning that it is replacing the machinefile.

4.2.8.2 Example of Using MPICH-P4 with PBS

Example of using `mpirun`:

```
#PBS -l select=arch=linux
#
mpirun a.out
```

4.2.8.3 MPICH Under Windows

Under Windows, you may need to use the `-localroot` option to MPICH's `mpirun` command in order to allow the job's processes to run more efficiently, or to get around the error "failed to communicate with the barrier command". Here is an example job script:

```
C:\DOCUME~1\user1>type job.scr
echo begin
type %PBS_NODEFILE%
"\Program Files\MPICH\mpd\bin\mpirun" -localroot -np 2 -machinefile
    %PBS_NODEFILE% \winnt\temp\netpipe -reps 3
echo done
```

4.2.8.3.i Caveats for MPICH Under Windows

Under Windows, MPICH is not integrated with PBS. Therefore, PBS is limited to tracking and controlling processes and performing accounting only for job processes on the primary vnode.

4.2.9 MPICH-GM with PBS

4.2.9.1 Using MPICH-GM and MPD with PBS

PBS provides an interface to MPICH-GM's `mpirun` using MPD. If executed inside a PBS job, this allows for PBS to track all MPICH-GM processes started by the MPD daemons so that PBS can perform accounting and have complete job control. If executed outside of a PBS job, it behaves exactly as if standard `mpirun` with MPD had been used.

You use the same `mpirun` command as you would use outside of PBS. If the MPD daemons are not already running, the PBS interface will take care of starting them for you.

4.2.9.1.i Options

Inside a PBS job script, all of the options to the PBS interface are the same as `mpirun` with MPD except for the following:

`-m <file>`

The `file` argument contents are ignored and replaced by the contents of `$PBS_NODEFILE`.

`-np`

If not specified, the number of entries found in `$PBS_NODEFILE` is used. The maximum number of ranks that can be launched is the number of entries in `$PBS_NODEFILE`

`-pg`

The use of the `-pg` option, for having multiple executables on multiple hosts, is allowed but it is up to you to make sure only PBS hosts are specified in the process group file; MPI processes spawned on non-PBS hosts are not guaranteed to be under the control of PBS.

4.2.9.1.ii MPD Startup and Shutdown

The script starts MPD daemons on each of the unique hosts listed in `$PBS_NODEFILE`, using either the `rsh` or `ssh` method based on the value of the environment variable `RSH-COMMAND`. The default is `rsh`. The script also takes care of shutting down the MPD daemons at the end of a run.

If the MPD daemons are not running, the PBS interface to `mpirun` will start GM's MPD daemons as you on the allocated PBS hosts. The MPD daemons may have been started already by the administrator or by you. MPD daemons are not started inside a PBS prologue script since it won't have the path of `mpirun` that you executed (GM or MX), which would determine the path to the MPD binary.

4.2.9.1.iii Examples

Example 4-26: Run a single-executable MPICH-GM job with 3 processes spread out across the PBS-allocated hosts listed in `$PBS_NODEFILE`:

```
$PBS_NODEFILE:
pbs-host1
pbs-host2
pbs-host3
qsub -l select=3:ncpus=1
[MPICH-GM-HOME]/bin/mpirun -np 3 /path/myprog.x 1200
^D
<job-id>
```

If the GM MPD daemons are not running, the PBS interface to `mpirun` will start them as you on the allocated PBS hosts. The daemons may have been previously started by the administrator or by you.

Example 4-27: Run an MPICH-GM job with multiple executables on multiple hosts listed in the process group file `procgrp`:

```
Job script:
qsub -l select=2:ncpus=1
echo "host1 1 user1 /x/y/a.exe arg1 arg2" > procgrp
echo "host2 1 user1 /x/x/b.exe arg1 arg2" >> procgrp

[MPICH-GM-HOME]/bin/mpirun -pg procgrp /path/mypro.x 1200
rm -f procgrp
^D
<job-id>
```

When the job runs, `mpirun` gives the warning message:

```
warning: "-pg" is allowed but it is up to user to make sure only PBS hosts
are specified; MPI processes spawned are not guaranteed to be under
PBS-control.
```

The warning is issued because if any of the hosts listed in `procgrp` are not under the control of PBS, then the processes on those hosts will not be under the control of PBS.

4.2.9.2 Using MPICH-GM and `rsh/ssh` with PBS

PBS provides an interface to MPICH-GM's `mpirun` using `rsh/ssh`. If executed inside a PBS job, this lets PBS track all MPICH-GM processes started via `rsh/ssh` so that PBS can perform accounting and have complete job control. If executed outside of a PBS job, it behaves exactly as if standard `mpirun` had been used.

You use the same `mpirun` command as you would use outside of PBS.

4.2.9.2.i Options

Inside a PBS job script, all of the options to the PBS interface are the same as `mpirun` except for the following:

`-machinefile <file>`

The `file` argument contents are ignored and replaced by the contents of `$PBS_NODEFILE`.

`-np`

If not specified, the number of entries found in `$PBS_NODEFILE` is used. The maximum number of ranks that can be launched is the number of entries in `$PBS_NODEFILE`.

`-pg`

The use of the `-pg` option, for having multiple executables on multiple hosts, is allowed but it is up to you to make sure only PBS hosts are specified in the process group file; MPI processes spawned on non-PBS hosts are not guaranteed to be under the control of PBS.

4.2.9.2.ii Examples

Example 4-28: Run a single-executable MPICH-GM job with 64 processes spread out across the PBS-allocated hosts listed in `$PBS_NODEFILE`:

`$PBS_NODEFILE:`

`pbs-host1`

`pbs-host2`

`...`

`pbs-host64`

```
qsub -l select=64:ncpus=1 -l place=scatter
mpirun -np 64 /path/myprog.x 1200
^D
<job-id>
```

Example 4-29: Run an MPICH-GM job with multiple executables on multiple hosts listed in the process group file `procgrp`:

```
qsub -l select=2:ncpus=1
echo "host1 1 user1 /x/y/a.exe arg1 arg2" > procgrp
echo "host2 1 user1 /x/x/b.exe arg1 arg2" >> procgrp
mpirun -pg procgrp /path/mypro.x
rm -f procgrp
^D
<job-id>
```

When the job runs, `mpirun` gives this warning message:

```
warning: "-pg" is allowed but it is up to user to make sure only PBS hosts
are specified; MPI processes spawned are not guaranteed to be under the
control of PBS.
```

The warning is issued because if any of the hosts listed in `procgrp` are not under the control of PBS, then the processes on those hosts will not be under the control of PBS.

4.2.9.3 Restrictions

The maximum number of ranks that can be launched under integrated MPICH-GM is the number of entries in `$PBS_NODEFILE`.

4.2.10 MPICH-MX with PBS

4.2.10.1 Using MPICH-MX and MPD with PBS

PBS provides an interface to MPICH-MX's `mpirun` using MPD. If executed inside a PBS job, this allows for PBS to track all MPICH-MX processes started by the MPD daemons so that PBS can perform accounting and have complete job control. If executed outside of a PBS job, it behaves exactly as if standard MPICH-MX `mpirun` with MPD was used.

You use the same `mpirun` command as you would use outside of PBS. If the MPD daemons are not already running, the PBS interface will take care of starting them for you.

4.2.10.1.i Options

Inside a PBS job script, all of the options to the PBS interface are the same as `mpirun` with MPD except for the following:

- `-m <file>`
The `file` argument contents are ignored and replaced by the contents of `$PBS_NODEFILE`.
- `-np`
If not specified, the number of entries found in `$PBS_NODEFILE` is used. The maximum number of ranks that can be launched is the number of entries in `$PBS_NODEFILE`.
- `-pg`
The use of the `-pg` option, for having multiple executables on multiple hosts, is allowed but it is up to you to make sure only PBS hosts are specified in the process group file; MPI processes spawned on non-PBS hosts are not guaranteed to be under the control of PBS.

4.2.10.1.ii MPD Startup and Shutdown

The PBS `mpirun` interface starts MPD daemons on each of the unique hosts listed in `$PBS_NODEFILE`, using either the `rsh` or `ssh` method, based on value of environment variable `RSHCOMMAND`. The default is `rsh`. The interface also takes care of shutting down the MPD daemons at the end of a run.

If the MPD daemons are not running, the PBS interface to `mpirun` starts MX's MPD daemons as you on the allocated PBS hosts. The MPD daemons may already have been started by the administrator or by you. MPD daemons are not started inside a PBS prologue script since it won't have the path of `mpirun` that you executed (GM or MX), which would determine the path to the MPD binary.

4.2.10.1.iii Examples

Example 4-30: Run a single-executable MPICH-MX job with 64 processes spread out across the PBS-allocated hosts listed in `$PBS_NODEFILE`:

```
$PBS_NODEFILE:
pbs-host1
pbs-host2
...
pbs-host64
```

```
qsub -l select=64:ncpus=1 -lplace=scatter
[MPICH-MX-HOME]/bin/mpirun -np 64 /path/myprog.x 1200
^D
<job-id>
```

If the MPD daemons are not running, the PBS interface to `mpirun` starts MX's MPD daemons as you on the allocated PBS hosts. The MPD daemons may be already started by the administrator or by you.

Example 4-31: Run an MPICH-MX job with multiple executables on multiple hosts listed in the process group file `procgrp`:

```
qsub -l select=2:ncpus=1
echo "pbs-host1 1 username /x/y/a.exe arg1 arg2" > procgrp
echo "pbs-host2 1 username /x/x/b.exe arg1 arg2" >> procgrp
[MPICH-MX-HOME]/bin/mpirun -pg procgrp /path/myprog.x 1200
rm -f procgrp
^D
<job-id>
```

`mpirun` prints a warning message:

```
warning: "-pg" is allowed but it is up to user to make sure only PBS hosts
are specified; MPI processes spawned are not guaranteed to be under
PBS-control
```

The warning is issued because if any of the hosts listed in `procgrp` are not under the control of PBS, then the processes on those hosts will not be under the control of PBS.

4.2.10.2 Using MPICH-MX and `rsh/ssh` with PBS

PBS provides an interface to MPICH-MX's `mpirun` using `rsh/ssh`. If executed inside a PBS job, this allows for PBS to track all MPICH-MX processes started by `rsh/ssh` so that PBS can perform accounting and has complete job control. If executed outside of a PBS job, it behaves exactly as if standard `mpirun` had been used.

You use the same `mpirun` command as you would use outside of PBS.

4.2.10.2.i Options

Inside a PBS job script, all of the options to the PBS interface are the same as standard `mpirun` except for the following:

-machinefile <file>

The `file` argument contents are ignored and replaced by the contents of `$PBS_NODEFILE`.

-np

If not specified, the number of entries found in the `$PBS_NODEFILE` is used. The maximum number of ranks that can be launched is the number of entries in `$PBS_NODEFILE`.

-pg

The use of the `-pg` option, for having multiple executables on multiple hosts, is allowed but it is up to you to make sure only PBS hosts are specified in the process group file; MPI processes spawned on non-PBS hosts are not guaranteed to be under the control of PBS.

4.2.10.2.ii Examples

Example 4-32: Run a single-executable MPICH-MX job with 64 processes spread out across the PBS-allocated hosts listed in `$PBS_NODEFILE`:

`$PBS_NODEFILE:`

`pbs-host1`

`pbs-host2`

`...`

`pbs-host64`

```
qsub -l select=64:ncpus=1
mpirun -np 64 /path/myprog.x 1200
^D
<job-id>
```

Example 4-33: Run an MPICH-MX job with multiple executables on multiple hosts listed in the process group file `procgrp`:

```
qsub -l select=2:ncpus=1
echo "pbs-host1 1 username /x/y/a.exe arg1 arg2" > procgrp
echo "pbs-host2 1 username /x/x/b.exe arg1 arg2" >> procgrp
mpirun -pg procgrp /path/myprog.x
rm -f procgrp
^D
<job-id>
```

`mpirun` prints the warning message:

```
warning: "-pg" is allowed but it is up to user to make sure only PBS hosts
are specified; MPI processes spawned are not guaranteed to be under
PBS-control
```

The warning is issued because if any of the hosts listed in `procgrp` are not under the control of PBS, then the processes on those hosts will not be under the control of PBS.

4.2.10.3 Restrictions

The maximum number of ranks that can be launched under integrated MPICH-MX is the number of entries in `$PBS_NODEFILE`.

4.2.11 MPICH2 with PBS

PBS provides an interface to MPICH2's `mpirun`. If executed inside a PBS job, this allows for PBS to track all MPICH2 processes so that PBS can perform accounting and have complete job control. If executed outside of a PBS job, it behaves exactly as if standard MPICH2's `mpirun` had been used.

You use the same `mpirun` command as you would use outside of PBS.

When submitting PBS jobs under the PBS interface to MPICH2's `mpirun`, be sure to explicitly specify the actual number of ranks or MPI tasks in the `qsub select` specification. Otherwise, jobs will fail to run with "too few entries in the machinefile".

For instance, the following erroneous specification:

```
#PBS -l select=1:ncpus=1:host=hostA+1:ncpus=2:host=hostB
mpirun -np 3 /tmp/mytask
```

results in this `$PBS_NODEFILE` listing:

```
hostA
hostB
```

which conflicts with the "`-np 3`" specification in `mpirun` as only two MPD daemons are started.

The correct way is to specify either of the following:

```
#PBS -l select=1:ncpus=1:host=hostA+2:ncpus=1:host=hostB
#PBS -l select=1:ncpus=1:host=hostA+1:ncpus=2:host=hostB:mpiprocs=2
```

which causes `$PBS_NODEFILE` to contain:

```
hostA
hostB
hostB
```

and this is consistent with "`mpirun -np 3`".

4.2.11.1 Options

If executed inside a PBS job script, all of the options to the PBS interface are the same as MPICH2's `mpirun` except for the following:

-host, -ghost

For specifying the execution host to run on. Ignored.

-machinefile <file>

The file argument contents are ignored and replaced by the contents of `$PBS_NODEFILE`.

-localonly <x>

For specifying the `<x>` number of processes to run locally. Not supported. You are advised instead to use the equivalent arguments:

`"-np <x> -localonly"`.

-np

If you do not specify a `-np` option, then no default value is provided by the PBS interface to MPICH2. It is up to the standard `mpirun` to decide what the reasonable default value should be, which is usually 1. The maximum number of ranks that can be launched is the number of entries in `$PBS_NODEFILE`.

4.2.11.2 MPD Startup and Shutdown

The interface ensures that the MPD daemons are started on each of the hosts listed in `$PBS_NODEFILE`. It also ensures that the MPD daemons are shut down at the end of MPI job execution.

4.2.11.3 Examples

Example 4-34: Run a single-executable MPICH2 job with six processes spread out across the PBS-allocated hosts listed in `$PBS_NODEFILE`. Only three hosts are available:

`$PBS_NODEFILE:`

```
pbs-host1
pbs-host2
pbs-host3
pbs-host1
pbs-host2
pbs-host3
```

Job.script:

```
# mpirun runs 6 processes, scattered over 3 hosts
# listed in $PBS_NODEFILE
mpirun -np 6 /path/myprog.x 1200
```

Run job script:

```
qsub -l select=6:ncpus=1 -lplace = scatter job.script
<job-id>
```

Example 4-35: Run an MPICH2 job with multiple executables on multiple hosts using `$PBS_NODEFILE` and `mpiexec` arguments in `mpirun`:

`$PBS_NODEFILE:`

```
hostA
hostA
hostB
hostB
hostC
hostC
```

Job script:

```
#PBS -l select=3:ncpus=2:mpiprocs=2
mpirun -np 2 /tmp/mpiTest1 : -np 2 /tmp/mpiTest2 : -np 2 /tmp/mpiTest3
```

Run job:

qsub job.script

Example 4-36: Run an MPICH2 job with multiple executables on multiple hosts using `mpirun -configfile` option and `$PBS_NODEFILE`:

`$PBS_NODEFILE`:

```
hostA
hostA
hostB
hostB
hostC
hostC
```

Job script:

```
#PBS -l select=3:ncpus=2:mpiprocs=2
echo "-np 2 /tmp/mpitest1" > my_config_file
echo "-np 2 /tmp/mpitest2" >> my_config_file
echo "-np 2 /tmp/mpitest3" >> my_config_file
mpirun -configfile my_config_file
rm -f my_config_file
```

Run job:

qsub job.script

4.2.11.4 Restrictions

The maximum number of ranks that can be launched under integrated MPICH2 is the number of entries in `$PBS_NODEFILE`.

4.2.12 MVAPICH with PBS

PBS provides an `mpirun` interface to the MVAPICH `mpirun`. When you use the PBS-supplied `mpirun`, PBS can track all MVAPICH processes, perform accounting, and have complete job control. Your PBS administrator can integrate MVAPICH with PBS so that you can use the PBS-supplied `mpirun` in place of the MVAPICH `mpirun` in your job scripts.

MVAPICH allows your jobs to use InfiniBand.

4.2.12.1 Interface to MVAPICH `mpirun` Command

If executed outside of a PBS job, the PBS-supplied interface to `mpirun` behaves exactly as if standard MVAPICH `mpirun` had been used.

If executed inside a PBS job script, all of the options to the PBS interface are the same as MVAPICH's `mpirun` except for the following:

`-map`

The `map` option is ignored.

`-machinefile <file>`

The `machinefile` option is ignored.

`-exclude`

The `exclude` option is ignored.

`-np`

If you do not specify a `-np` option, then PBS uses the number of entries found in `$PBS_NODEFILE`. The maximum number of ranks that can be launched is the number of entries in `$PBS_NODEFILE`.

4.2.12.2 Examples

Example 4-37: Run a single-executable MVAPICH job with six ranks spread out across the PBS-allocated hosts listed in `$PBS_NODEFILE`:

`$PBS_NODEFILE`:

```
pbs-host1
pbs-host1
pbs-host2
pbs-host2
pbs-host3
pbs-host3
```

Contents of `job.script`:

```
# mpirun runs 6 processes mapped one to each line in $PBS_NODEFILE
mpirun -np 6 /path/myprog
```

Run job script:

```
qsub -l select=3:ncpus=2:mpiprocs=2 job.script  
<job-id>
```

4.2.12.3 Restrictions

The maximum number of ranks that can be launched under integrated MVAPICH is the number of entries in \$PBS_NODEFILE.

4.2.13 MVAPICH2 with PBS

PBS provides an `mpiexec` interface to MVAPICH2's `mpiexec`. When you use the PBS-supplied `mpiexec`, PBS can track all MVAPICH2 processes, perform accounting, and have complete job control. Your PBS administrator can integrate MVAPICH2 with PBS so that you can use the PBS-supplied `mpirun` in place of the MVAPICH2 `mpirun` in your job scripts.

MVAPICH2 allows your jobs to use InfiniBand.

4.2.13.1 Interface to MVAPICH2 `mpiexec` Command

If executed outside of a PBS job, it behaves exactly as if standard MVAPICH2's `mpiexec` had been used.

If executed inside a PBS job script, all of the options to the PBS interface are the same as MVAPICH2's `mpiexec` except for the following:

-host

The `host` option is ignored.

-machinefile <file>

The `file` option is ignored.

-mpdboot

If `mpdboot` is not called before `mpiexec`, it is called automatically before `mpiexec` runs so that an MPD daemon is started on each host assigned by PBS.

4.2.13.2 MPD Startup and Shutdown

The interface ensures that the MPD daemons are started on each of the hosts listed in `$PBS_NODEFILE`. It also ensures that the MPD daemons are shut down at the end of MPI job execution.

4.2.13.3 Examples

Example 4-38: Run a single-executable MVAPICH2 job with six ranks on hosts listed in `$PBS_NODEFILE`:

```
$PBS_NODEFILE:
```

```
pbs-host1
pbs-host1
pbs-host2
pbs-host2
pbs-host3
pbs-host3
```

Job.script:

```
mpiexec -np 6 /path/mpiprogram
```

Run job script:

```
qsub -l select=3:ncpus=2:mpiprocs=2 job.script
<job-id>
```

Example 4-39: Launch an MVAPICH2 MPI job with multiple executables on multiple hosts listed in the default file `"mpd.hosts"`. Here, run executables `prog1` and `prog2` with two ranks of `prog1` on `host1`, two ranks of `prog2` on `host2` and two ranks of `prog2` on `host3`, all specified on the command line:

```
$PBS_NODEFILE:
```

```
pbs-host1
pbs-host1
pbs-host2
pbs-host2
pbs-host3
pbs-host3
```

Job.script:

```
mpiexec -n 2 prog1 : -n 2 prog2 : -n 2 prog2
```

Run job script:

```
qsub -l select=3:ncpus=2:mpiprocs=2 job.script  
<job-id>
```

Example 4-40: Launch an MVAPICH2 MPI job with multiple executables on multiple hosts listed in the default file "mpd.hosts". Run executables prog1 and prog2 with two ranks of prog1 on host1, two ranks of prog2 on host2 and two ranks of prog2 on host3, all specified using the -configfile option:

\$PBS_NODEFILE:

```
pbs-host1  
pbs-host1  
pbs-host2  
pbs-host2  
pbs-host3  
pbs-host3
```

Job.script:

```
echo "-n 2 -host host1 prog1" > /tmp/jobconf  
echo "-n 2 -host host2 prog2" >> /tmp/jobconf  
echo "-n 2 -host host3 prog2" >> /tmp/jobconf  
mpiexec -configfile /tmp/jobconf  
rm /tmp/jobconf
```

Run job script:

```
qsub -l select=3:ncpus=2:mpiprocs=2 job.script  
<job-id>
```

4.2.13.4 Restrictions

The maximum number of ranks that can be launched under MVAPICH2 is the number of entries in \$PBS_NODEFILE.

4.2.14 Open MPI with PBS

Open MPI can be integrated with PBS on UNIX and Linux so that PBS can track resource usage, signal processes, and perform accounting, for all job processes. Your PBS administrator can integrate Open MPI with PBS.

4.2.14.1 Using Open MPI with PBS

You can run jobs under PBS using Open MPI without making any changes to your MPI command line.

4.2.15 Platform MPI with PBS

Platform MPI can be integrated with PBS on UNIX and Linux so that PBS can track resource usage, signal processes, and perform accounting, for all job processes. Your PBS administrator can integrate Platform MPI with PBS.

4.2.15.1 Using Platform MPI with PBS

You can run jobs under PBS using Platform MPI without making any changes to your MPI command line.

4.2.15.2 Setting up Your Environment

In order to override the default `rsh`, set `PBS_RSHCOMMAND` in your job script:

```
export PBS_RSHCOMMAND=<rsh_cmd>
```

4.2.16 SGI MPT with PBS

PBS supplies its own `mpiexec` to use with SGI MPT on the Altix running supported versions of ProPack or Performance Suite. When you use the PBS-supplied `mpiexec`, PBS can track resource usage, signal processes, and perform accounting, for all job processes. The PBS `mpiexec` provides the standard `mpiexec` interface.

See your PBS administrator to find out whether your system is configured for the PBS `mpiexec`.

4.2.16.1 Using SGI MPT with PBS

You can launch an MPI job on a single Altix, or across multiple Altixes. For MPI jobs across multiple Altixes, PBS will manage the multi-host jobs. For example, if you have two Altixes named Alt1 and Alt2, and want to run two applications called `mympi1` and `mympi2` on them, you can put this in your job script:

```
mpiexec -host Alt1 -n 4 mympi1 : -host Alt2 -n 8 mympi2
```

PBS will manage and track the job's processes. When the job is finished, PBS will clean up after it.

You can run MPI jobs in the placement sets chosen by PBS.

4.2.16.2 Prerequisites

In order to use MPI within a PBS job with Performance Suite, you may need to add the following in your job script before you call MPI:

```
module load mpt
```

4.2.16.3 Using Cpusets

PBS will run the MPI tasks in the cpusets it manages.

Jobs will share cpusets if the jobs request sharing and the vnodes' `sharing` attribute is not set to `force_excl`. Jobs can share the memory on a nodeboard if they have a CPU from that nodeboard. To fit as many small jobs as possible onto vnodes that already have shared jobs on them, request sharing in the job resource requests.

The `alt_id` job attribute has the form `cpuset=<name>`, where `<name>` is the name of the cpuset, which is the `$PBS_JOBID`.

To verify how many CPUs are included in a cpuset created by PBS, use:

```
> $ cpuset -d <set name> | egrep cpus
```

This will work either inside or outside a job.

4.2.16.4 Fitting Jobs onto Nodeboards

PBS will try to put a job that fits in a single nodeboard on just one nodeboard. However, if the only CPUs available are on separate nodeboards, and those vnodes are not allocated exclusively to existing jobs, and the job can share a vnode, then the job is run on the separate nodeboards.

4.2.16.5 Checkpointing and Suspending Jobs

Jobs are suspended on the Altix using the PBS suspend feature. If a job is suspended, its processes are moved to the global cpuset. When the job is restarted, they are restored.

Jobs are checkpointed on the Altix using application-level checkpointing. There is no OS-level checkpoint.

Suspended or checkpointed jobs will resume on the original nodeboards.

4.2.16.6 Specifying Array Name

You can specify the name of the array to use via the `PBS_MPI_SGIARRAY` environment variable.

4.2.16.7 Using CSA

PBS support for CSA on SGI systems is no longer available. The CSA functionality for SGI systems has been **removed** from PBS.

4.3 Using PVM with PBS

You use the `pvmexec` command to execute a Parallel Virtual Machine (PVM) program. PVM is not integrated with PBS; PBS is limited to monitoring, controlling, and accounting for job processes only on the primary vnode.

4.3.1 Arguments to `pvmexec` Command

The `pvmexec` command expects a `hostfile` argument for the list of hosts on which to spawn the parallel job.

4.3.2 Using PVM Daemons

To start the PVM daemons on the hosts listed in `$PBS_NODEFILE`:

1. Start the PVM console on the first host in the list
2. Print the hosts to the standard output file named `jobname.o<PBS job ID>`:
`echo conf | pvm $PBS_NODEFILE`

To quit the PVM console but leave the PVM daemons running:

`quit`

To stop the PVM daemons, restart the PVM console, and quit:

```
echo halt | pvm
```

4.3.3 Submitting a PVM Job

To submit a PVM job to PBS, use the following:

```
qsub <job script>
```

4.3.4 Examples

Example 4-41: To submit a PVM job to PBS, use the following:

```
qsub your_pvm_job
```

Here is an example script for your_pvm_job:

```
#PBS -N pvmjob
#PBS -V
cd $PBS_O_WORKDIR
echo conf | pvm $PBS_NODEFILE
echo quit | pvm
./my_pvm_program
echo halt | pvm
```

Example 4-42: Sample PBS script for a PVM job:

```
#PBS -N pvmjob
#
pvmexec a.out -inputfile data_in
```

4.4 Using OpenMP with PBS

PBS Professional supports OpenMP applications by setting the `OMP_NUM_THREADS` variable in the job's environment, based on the resource request of the job. The OpenMP run-time picks up the value of `OMP_NUM_THREADS` and creates threads appropriately.

MoM sets the value of `OMP_NUM_THREADS` based on the first chunk of the `select` statement. If you request `ompthreads` in the first chunk, MoM sets the environment variable to the value of `ompthreads`. If you do not request `ompthreads` in the first chunk, then

OMP_NUM_THREADS is set to the value of the ncpus resource of that chunk. If you do not request either ncpus or ompthreads for the first chunk of the select statement, then OMP_NUM_THREADS is set to 1.

You cannot directly set the value of the OMP_NUM_THREADS environment variable; MoM will override any setting you attempt.

See [“Built-in Resources” on page 299 of the PBS Professional Reference Guide](#) for a definition of the ompthreads resource.

Example 4-43: Submit an OpenMP job as a single chunk, for a two-CPU, two-thread job requiring 10gb of memory:

```
qsub -l select=1:ncpus=2:mem=10gb
```

Example 4-44: Run an MPI application with 64 MPI processes, and one thread per process:

```
#PBS -l select=64:ncpus=1  
mpiexec -n 64 ./a.out
```

Example 4-45: Run an MPI application with 64 MPI processes, and four OpenMP threads per process:

```
#PBS -l select=64:ncpus=4  
mpiexec -n 64 omplace -nt 4 ./a.out  
  
or  
  
#PBS -l select=64:ncpus=4:ompthreads=4  
mpiexec -n 64 omplace -nt 4 ./a.out
```

4.4.1 Running Fewer Threads than CPUs

You might be running an OpenMP application on a host and wish to run fewer threads than the number of CPUs requested. This might be because the threads need exclusive access to shared resources in a multi-core processor system, such as to a cache shared between cores, or to the memory shared between cores.

Example 4-46: You want one chunk, with 16 CPUs and eight threads:

```
qsub -l select=1:ncpus=16:ompthreads=8
```

4.4.2 Running More Threads than CPUs

You might be running an OpenMP application on a host and wish to run more threads than the number of CPUs requested, perhaps because each thread is I/O bound.

Example 4-47: You want one chunk, with eight CPUs and 16 threads:

```
qsub -l select=1:ncpus=8:ompthreads=16
```

4.4.3 Caveats for Using OpenMP with PBS

Make sure that you request the correct number of MPI ranks for your job, so that the PBS node file contains the correct number of entries. See [section 4.1.4, “Specifying Number of MPI Processes Per Chunk”, on page 83](#).

4.5 Hybrid MPI-OpenMP Jobs

For jobs that are both MPI and multi-threaded, the number of threads per chunk, for all chunks, is set to the number of threads requested (explicitly or implicitly) in the first chunk, except for MPIs that have been integrated with the PBS TM API.

For MPIs that are integrated with the PBS TM interface, (LAM MPI and Open MPI), you can specify the number of threads separately for each chunk, by specifying the `ompthreads` resource separately for each chunk.

For most MPIs, the `OMP_NUM_THREADS` and `NCPUS` environment variables default to the number of `ncpus` requested for the first chunk.

Should you have a job that is both MPI and multi-threaded, you can request one chunk for each MPI process, or set `mpiprocs` to the number of MPI processes you want on each chunk. See [section 4.1.4, “Specifying Number of MPI Processes Per Chunk”, on page 83](#).

4.5.1 Examples

Example 4-48: To request four chunks, each with one MPI process, two CPUs and two threads:

```
qsub -l select=4:ncpus=2
```

or

```
qsub -l select=4:ncpus=2:ompthreads=2
```

Example 4-49: To request four chunks, each with two CPUs and four threads:

```
qsub -l select=4:ncpus=2:ompthreads=4
```

Example 4-50: To request 16 MPI processes each with two threads on machines with two processors:

```
qsub -l select=16:ncpus=2
```

Example 4-51: To request two chunks, each with eight CPUs and eight MPI tasks and four threads:

```
qsub -l select=2:ncpus=8:mpiprocs=8:ompthreads=4
```

Example 4-52: For the following:

```
qsub -l select=4:ncpus=2
```

This request is satisfied by four CPUs from VnodeA, two from VnodeB and two from VnodeC, so the following is written to \$PBS_NODEFILE:

VnodeA

VnodeA

VnodeB

VnodeC

The OpenMP environment variables are set, for the four PBS tasks corresponding to the four MPI processes, as follows:

- For PBS task #1 on VnodeA: OMP_NUM_THREADS=2 NCPUS=2
- For PBS task #2 on VnodeA: OMP_NUM_THREADS=2 NCPUS=2
- For PBS task #3 on VnodeB: OMP_NUM_THREADS=2 NCPUS=2

- For PBS task #4 on VnodeC: OMP_NUM_THREADS=2 NCPUS=2

Example 4-53: For the following:

```
qsub -l select=3:ncpus=2:mpiprocs=2:ompthreads=1
```

This is satisfied by two CPUs from each of three vnodes (VnodeA, VnodeB, and VnodeC), so the following is written to \$PBS_NODEFILE:

VnodeA

VnodeA

VnodeB

VnodeB

VnodeC

VnodeC

The OpenMP environment variables are set, for the six PBS tasks corresponding to the six MPI processes, as follows:

- For PBS task #1 on VnodeA: OMP_NUM_THREADS=1 NCPUS=1
- For PBS task #2 on VnodeA: OMP_NUM_THREADS=1 NCPUS=1
- For PBS task #3 on VnodeB: OMP_NUM_THREADS=1 NCPUS=1
- For PBS task #4 on VnodeB: OMP_NUM_THREADS=1 NCPUS=1
- For PBS task #5 on VnodeC: OMP_NUM_THREADS=1 NCPUS=1
- For PBS task #6 on VnodeC: OMP_NUM_THREADS=1 NCPUS=1

Example 4-54: To run two threads on each of *N* chunks, each running a process, all on the same Altix:

```
qsub -l select=N:ncpus=2 -l place=pack
```

This starts *N* processes on a single host, with two OpenMP threads per process, because OMP_NUM_THREADS=2.

Chapter 5

Using the xpbs GUI

The PBS graphical user interface is called **xpbs**, and provides a user-friendly, point and click interface to the PBS commands. **xpbs** utilizes the tcl/tk graphics tool suite, while providing the user with most of the same functionality as the PBS CLI commands. In this chapter we introduce **xpbs**, and show how to create a PBS job using **xpbs**.

5.1 Using the **xpbs** command

5.1.1 Starting **xpbs**

If PBS is installed on your local workstation, or if you are running under Windows, you can launch **xpbs** by double-clicking on the **xpbs** icon on the desktop. You can also start **xpbs** from the command line with the following command.

UNIX:

```
xpbs &
```

Windows:

```
xpbs.exe
```

Doing so will bring up the main **xpbs** window, as shown below.

5.1.2 Running `xpbs` Under UNIX

Before running `xpbs` for the first time under UNIX, you may need to configure your workstation for it. Depending on how PBS is installed at your site, you may need to allow `xpbs` to be displayed on your workstation. However, if the PBS client commands are installed locally on your workstation, you can skip this step. (Ask your PBS administrator if you are unsure.)

The most secure method of running `xpbs` remotely and displaying it on your local XWindows session is to redirect the XWindows traffic through `ssh` (secure shell), via setting the "`X11Forwarding yes`" parameter in the `sshd_config` file. (Your local system administrator can provide details on this process if needed.)

An alternative, but less secure, method is to direct your X-Windows session to permit the `xpbs` client to connect to your local X-server. Do this by running the `xhost` command with the name of the host from which you will be running `xpbs`, as shown in the example below:

```
xhost + server.mydomain.com
```

Next, on the system from which you will be running `xpbs`, set your X-Windows **DISPLAY** variable to your local workstation. For example, if using the C-shell:

```
setenv DISPLAY myWorkstation:0.0
```

However, if you are using the Bourne or Korn shell, type the following:

```
export DISPLAY=myWorkstation:0.0
```

5.2 Using `xpbs`: Definitions of Terms

The various panels, boxes, and regions (collectively called “widgets”) of `xpbs` and how they are manipulated are described in the following sections. A *listbox* can be multi-selectable (a number of entries can be selected/highlighted using a mouse click) or single-selectable (one entry can be highlighted at a time).

For a multi-selectable listbox, the following operations are allowed:

- left-click to select/highlight an entry.
- shift-left-click to contiguously select more than one entry.
- control-left-click to select multiple non-contiguous entries.
- click the *Select All / Deselect All* button to select all entries or deselect all entries at once.
- double clicking an entry usually activates some action that uses the selected entry as a parameter.

An *entry* widget is brought into focus with a left-click. To manipulate this widget, simply type in the text value. Use of arrow keys and mouse selection of text for deletion, overwrite, copying and pasting with sole use of mouse buttons are permitted. This widget has a scrollbar for horizontally scanning a long text entry string.

A *matrix of entry boxes* is usually shown as several rows of entry widgets where a number of entries (called fields) can be found per row. The matrix is accompanied by up/down arrow buttons for paging through the rows of data, and each group of fields gets one scrollbar for horizontally scanning long entry strings. Moving from field to field can be done using the <Tab> (move forward), <Ctrl-f> (move forward), or <Ctrl-b> (move backward) keys.

A *spinbox* is a combination of an entry widget and a horizontal scrollbar. The entry widget will only accept values that fall within a defined list of valid values, and incrementing through the valid values is done by clicking on the up/down arrows.

A *button* is a rectangular region appearing either raised or pressed that invokes an action when clicked with the left mouse button. When the button appears pressed, then hitting the <RETURN> key will automatically select the button.

A *text region* is an editor-like widget. This widget is brought into focus with a left-click. To manipulate this widget, simply type in the text. Use of arrow keys, backspace/delete key, mouse selection of text for deletion or overwrite, and copying and pasting with sole use of mouse buttons are permitted. This widget has a scrollbar for vertically scanning a long entry.

5.3 Introducing the xpbs Main Display

The main window or display of xpbs is comprised of five collapsible subwindows or *panels*. Each panel contains specific information. Top to bottom, these panels are: the Menu Bar, Hosts panel, Queues panel, Jobs panel, and the Info panel.

5.3.1 xpbs Menu Bar

The Menu Bar is composed of a row of command buttons that signal some action with a click of the left mouse button. The buttons are:

Manual Update

forces an update of the information on hosts, queues, and jobs.

Auto Update

sets an automatic update of information every user-specified number of minutes.

Track Job

for periodically checking for returned output files of jobs.

Preferences

for setting parameters such as the list of Server host(s) to query.

Help

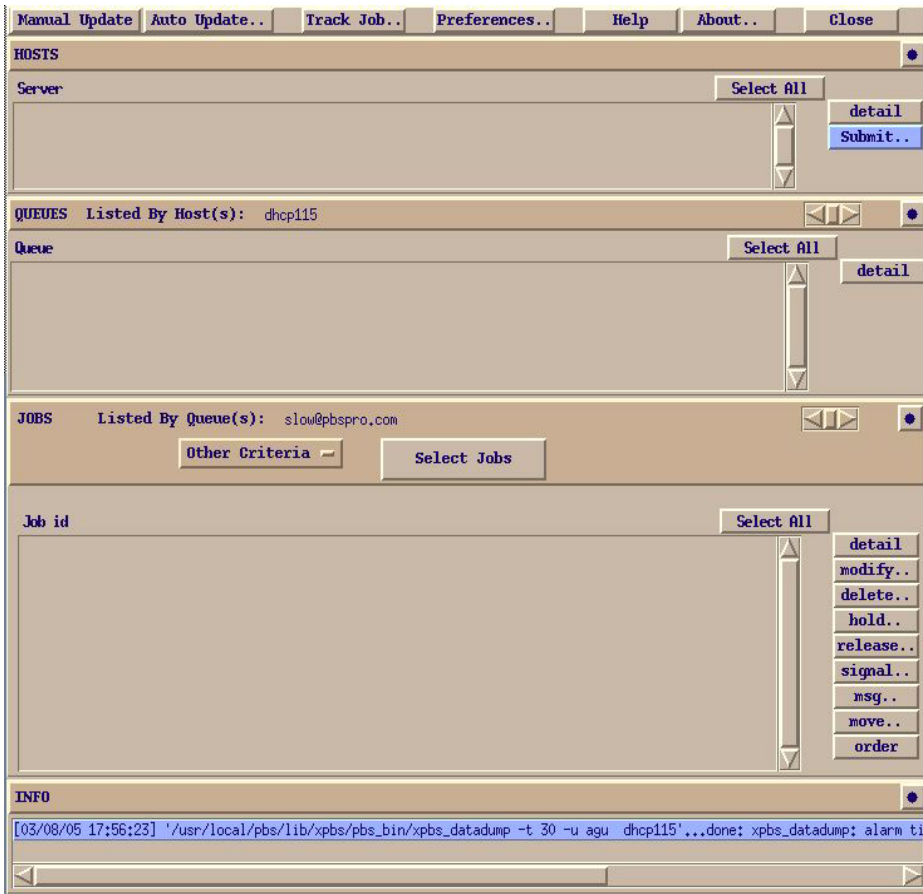
contains some help information.

About

gives general information about the xpbs GUI.

Close

for exiting xpbs plus saving the current setup information.



5.3.2 xpbs Hosts Panel

The Hosts panel is composed of a leading horizontal HOSTS bar, a listbox, and a set of command buttons. The HOSTS bar contains a minimize/maximize button, identified by a dot or a rectangular image, for displaying or iconizing the Hosts region. The listbox displays information about favorite Server host(s), and each entry is meant to be selected via a single left-click, shift-left-click for contiguous selection, or control-left-click for non-contiguous selection.

To the right of the Hosts Panel are buttons that represent actions that can be performed on selected host(s). Use of these buttons will be explained in detail below.

detail

Provides information about selected Server host(s). This functionality can also be achieved by double clicking on an entry in the Hosts listbox.

submit

For submitting a job to any of the queues managed by the selected host(s).

terminate

For terminating (shutting down) PBS Servers on selected host(s). (Visible via the “-admin” option only.)

IMPORTANT:

Note that some buttons are only visible if xpbs is started with the “-admin” option, which requires manager or operator privilege to function.

The middle portion of the Hosts Panel has abbreviated column names indicating the information being displayed, as the following table shows:

Table 5-1: xpbs Server Column Headings

Heading	Meaning
Max	Maximum number of jobs permitted
Tot	Count of jobs currently enqueued in any state
Que	Count of jobs in the Queued state
Run	Count of jobs in the Running state
Hld	Count of jobs in the Held state
Wat	Count of jobs in the Waiting state
Trn	Count of jobs in the Transiting state

Table 5-1: xpbs Server Column Headings

Heading	Meaning
Ext	Count of jobs in the Exiting state
Status	Status of the corresponding Server
PEsInUse	Count of Processing Elements (CPUs, PEs, Vnodes) in Use

5.3.3 xpbs Queues Panel

The Queues panel is composed of a leading horizontal QUEUES bar, a listbox, and a set of command buttons. The QUEUES bar lists the hosts that are consulted when listing queues; the bar also contains a minimize/maximize button for displaying or iconizing the Queues panel. The listbox displays information about queues managed by the Server host(s) selected from the Hosts panel; each listbox entry can be selected as described above for the Hosts panel.

To the right of the Queues Panel area are buttons for actions that can be performed on selected queue(s).

detail

provides information about selected queue(s). This functionality can also be achieved by double clicking on a Queue listbox entry.

stop

for stopping the selected queue(s). (-admin only)

start

for starting the selected queue(s). (-admin only)

disable

for disabling the selected queue(s). (-admin only)

enable

for enabling the selected queue(s). (-admin only)

The middle portion of the Queues Panel has abbreviated column names indicating the information being displayed, as the following table shows:

Table 5-2: xpbs Queue Column Headings

Heading	Meaning
Max	Maximum number of jobs permitted

Table 5-2: xpbs Queue Column Headings

Heading	Meaning
Tot	Count of jobs currently enqueued in any state
Ena	Is queue enabled? yes or no
Str	Is queue started? yes or no
Que	Count of jobs in the Queued state
Run	Count of jobs in the Running state
Hld	Count of jobs in the Held state
Wat	Count of jobs in the Waiting state
Trn	Count of jobs in the Transiting state
Ext	Count of jobs in the Exiting state
Type	Type of queue: execution or route
Server	Name of Server on which queue exists

5.3.4 xpbs Jobs Panel

The Jobs panel is composed of a leading horizontal JOBS bar, a listbox, and a set of command buttons. The JOBS bar lists the queues that are consulted when listing jobs; the bar also contains a minimize/maximize button for displaying or iconizing the Jobs region. The listbox displays information about jobs that are found in the queue(s) selected from the Queues listbox; each listbox entry can be selected as described above for the Hosts panel.

The region just above the Jobs listbox shows a collection of command buttons whose labels describe criteria used for filtering the Jobs listbox contents. The list of jobs can be selected according to the owner of jobs (Owners), job state (Job_States), name of the job (Job_Name), type of hold placed on the job (Hold_Types), the account name associated with the job (Account_Name), checkpoint attribute (Checkpoint), time the job is eligible for queueing/execution (Queue_Time), resources requested by the job (Resources), priority attached to the job (Priority), and whether or not the job is rerunnable (Rerunnable).

The selection criteria can be modified by clicking on any of the appropriate command buttons to bring up a selection box. The criteria command buttons are accompanied by a *Select Jobs* button, which when clicked, will update the contents of the Jobs listbox based on the new selection criteria. Note that only jobs that meet *all* the selected criteria will be displayed.

Finally, to the right of the Jobs panel are the following command buttons, for operating on selected job(s):

detail	provides information about selected job(s). This functionality can also be achieved by double-clicking on a Jobs listbox entry.
modify	for modifying attributes of the selected job(s).
delete	for deleting the selected job(s).
hold	for placing some type of hold on selected job(s).
release	for releasing held job(s).
signal	for sending signals to selected job(s) that are running.
msg	for writing a message into the output streams of selected job(s).
move	for moving selected job(s) into some specified destination.
order	for exchanging order of two selected jobs in a queue.
run	for running selected job(s). (-admin only)
rerun	for requeueing selected job(s) that are running. (-admin only)

The middle portion of the Jobs Panel has abbreviated column names indicating the information being displayed, as the following table shows:

Table 5-3: xpbs Job Column Headings

Heading	Meaning
Job id	Job Identifier

Table 5-3: xpbs Job Column Headings

Heading	Meaning
Name	Name assigned to job, or script name
User	User name under which job is running
PEs	Number of Processing Elements (CPUs) requested
CputUse	Amount of CPU time used
WalltUse	Amount of wall-clock time used
S	State of job
Queue	Queue in which job resides

5.3.5 xpbs Info Panel

The Info panel shows the progress of the commands executed by `xpbs`. Any errors are written to this area. The INFO panel also contains a minimize/maximize button for displaying or iconizing the Info panel.

5.3.6 xpbs Keyboard Tips

There are a number of shortcuts and key sequences that can be used to speed up using `xpbs`. These include:

Tip 1.

All buttons which appear to be depressed in the dialog box/subwindow can be activated by pressing the return/enter key.

Tip 2.

Pressing the tab key will move the blinking cursor from one text field to another.

Tip 3.

To contiguously select more than one entry: left-click then drag the mouse across multiple entries.

Tip 4.

To non-contiguously select more than one entry: hold the control-left-click on the desired entries.

5.4 Setting xpbs Preferences

The “Preferences” button is in the Menu Bar at the top of the main xpbs window. Clicking it will bring up a dialog box that allows you to customize the behavior of xpbs:

1. Define Server hosts to query
2. Select wait timeout in seconds
3. Specify `xterm` command (for interactive jobs, UNIX only)
4. Specify which `rsh/ssh` command to use



5.5 Relationship Between PBS and xpbs

xpbs is built on top of the PBS client commands, such that all the features of the command line interface are available through the GUI. Each “task” that you perform using xpbs is converted into the necessary PBS command and then run.

Table 5-4: xpbs Buttons and PBS Commands

Location	Command Button	PBS Command
Hosts Panel	detail	<code>qstat -B -f selected server_host(s)</code>
Hosts Panel	submit	<code>qsub options selected Server(s)</code>
Hosts Panel	terminate *	<code>qterm selected server_host(s)</code>
Queues Panel	detail	<code>qstat -Q -f selected queue(s)</code>
Queues Panel	stop *	<code>qstop selected queue(s)</code>
Queues Panel	start *	<code>qstart selected queue(s)</code>
Queues Panel	enable *	<code>qenable selected queue(s)</code>
Queues Panel	disable *	<code>qdisable selected queue(s)</code>
Jobs Panel	detail	<code>qstat -f selected job(s)</code>
Jobs Panel	modify	<code>qalter selected job(s)</code>
Jobs Panel	delete	<code>qdel selected job(s)</code>
Jobs Panel	hold	<code>qhold selected job(s)</code>
Jobs Panel	release	<code>qrls selected job(s)</code>
Jobs Panel	run	<code>qrun selected job(s)</code>
Jobs Panel	rerun	<code>qrerun selected job(s)</code>
Jobs Panel	signal	<code>qsig selected job(s)</code>
Jobs Panel	msg	<code>qmsg selected job(s)</code>
Jobs Panel	move	<code>qmove selected job(s)</code>

Table 5-4: xpbs Buttons and PBS Commands

Location	Command Button	PBS Command
Jobs Panel	order	<i>qorder selected job(s)</i>

* Indicates command button is visible only if xpbs is started with the “-admin” option.

5.6 How to Submit a Job Using xpbs

To submit a job using xpbs, perform the following steps:

First, select a host from the HOSTS listbox in the main xpbs display to which you wish to submit the job.

Next, click on the *Submit* button located next to the HOSTS panel. The *Submit* button brings up the Submit Job Dialog box (see below) which is composed of four distinct regions. The Job Script File region is at the upper left. The OPTIONS region containing various widgets for setting job attributes is scattered all over the dialog box. The OTHER OPTIONS is located just below the Job Script file region, and COMMAND BUTTONS region is at the bottom.

The job script region is composed of a header box, the text box, FILE entry box, and two buttons labeled *load* and *save*. If you have a script file containing PBS options and executable lines, then type the name of the file on the FILE entry box, and then click on the *load* button. Alternatively, you may click on the *FILE* button, which will display a File Selection browse window, from which you may point and click to select the file you wish to open. The File Selection Dialog window is shown below. Clicking on the *Select File* button will load the file into xpbs, just as does the *load* button described above.



The various fields in the Submit window will get loaded with values found in the script file. The script file text box will only be loaded with executable lines (non-PBS) found in the script. The job script header box has a *Prefix* entry box that can be modified to specify the PBS directive to look for when parsing a script file for PBS options.

If you don't have an existing script file to load into xpbs, you can start typing the executable lines of the job in the file text box.

Next, review the Destination listbox. This box shows the queues found in the host that you selected. A special entry called "@host" refers to the default queue at the indicated host. Select appropriately the destination queue for the job.

Next, define any required resources in the Resource List subwindow.

The resources specified in the "Resource List" section will be job-wide resources only. In order to specify chunks or job placement, use a script.

To run an array job, use a script. You will not be able to query individual subjobs or the whole job array using xpbs. Type the script into the "File: entry" box. Do not click the "Load" button. Instead, use the "Submit" button.

Finally, review the optional settings to see if any should apply to this job.

For example:

- Use the one of the buttons in the “Output” region to merge output and error files.
- Use “Stdout File Name” to define standard output file and to redirect output
- Use the “Environment Variables to Export” subwindow to have current environment variables exported to the job.
- Use the “Job Name” field in the OPTIONS subwindow to give the job a name.
- Use the “Notify email address” and one of the buttons in the OPTIONS subwindow to have PBS send you mail when the job terminates.

Now that the script is built you have four options of what to do next:

Reset options to default

Save the script to a file

Submit the job as a batch job

Submit the job as an interactive-batch job (UNIX only)

Reset clears all the information from the submit job dialog box, allowing you to create a job from a fresh start.

Use the *FILE* field (in the upper left corner) to define a filename for the script. Then press the *Save* button. This will cause a PBS script file to be generated and written to the named file.

Pressing the *Confirm Submit* button at the bottom of the Submit window will submit the PBS job to the selected destination. *xpbs* will display a small window containing the job identifier returned for this job. Clicking *OK* on this window will cause it and the Submit window to be removed from your screen.

On UNIX systems (not Windows) you can alternatively submit the job as an interactive-batch job, by clicking the *Interactive* button at the bottom of the Submit Job window. Doing so will cause an X-terminal window (*xterm*) to be launched, and within that window a PBS interactive-batch job submitted. The path for the *xterm* command can be set via the preferences, as discussed above in [section 5.4, “Setting xpbs Preferences”, on page 143](#). For further details on usage, and restrictions, see [section 3.13.21, “Interactive-batch Jobs”, on page 77](#).)

5.7 Exiting xpbs

Click on the *Close* button located in the Menu bar to leave *xpbs*. If any settings have been changed, *xpbs* will bring up a dialog box asking for a confirmation in regards to saving state information. The settings will be saved in the *.xpbsrc* configuration file, and will be used the next time you run *xpbs*, as discussed in the following section.

5.8 The xpbs Configuration File

Upon exit, the xpbs state may be written to the `.xpbsrc` file in the user's home directory. (See also [section 2.17.2, “Windows User's HOMEDIR”, on page 15](#).) Information saved includes: the selected host(s), queue(s), and job(s); the different jobs listing criteria; the view states (i.e. minimized/maximized) of the Hosts, Queues, Jobs, and INFO regions; and all settings in the Preferences section. In addition, there is a system-wide xpbs configuration file, maintained by the PBS Administrator, which is used in the absence of a user's personal `.xpbsrc` file.

5.9 xpbs Preferences

The resources that can be set in the xpbs configuration file, `~/.xpbsrc`, are:

***serverHosts**

List of Server hosts (space separated) to query by xpbs. A special keyword **PBS_DEFAULT_SERVER** can be used which will be used as a placeholder for the value obtained from the `/etc/pbs.conf` file (UNIX) or “[PBS Destination Folder]\pbs.conf” file (Windows).

***timeoutSecs**

Specify the number of seconds before timing out waiting for a connection to a PBS host.

***xtermCmd**

The xterm command to run driving an interactive PBS session.

***labelFont**

Font applied to text appearing in labels.

***fixlabelFont**

Font applied to text that label fixed-width widgets such as listbox labels. This must be a fixed-width font.

***textFont**

Font applied to a text widget. Keep this as fixed-width font.

***backgroundColor**

The color applied to background of frames, buttons, entries, scrollbar handles.

***foregroundColor**

The color applied to text in any context.

***activeColor**

The color applied to the background of a selection, a selected command button, or a selected scroll bar handle.

***disabledColor**

Color applied to a disabled widget.

***signalColor**

Color applied to buttons that signal something to the user about a change of state. For example, the

color of the *Track Job* button when returned output files are detected.

***shadingColor**

A color shading applied to some of the frames to emphasize focus as well as decoration.

***selectorColor**

The color applied to the selector box of a radiobutton or checkbutton.

***selectHosts**

List of hosts (space separated) to automatically select/highlight in the HOSTS listbox.

***selectQueues**

List of queues (space separated) to automatically select/highlight in the QUEUES listbox.

***selectJobs**

List of jobs (space separated) to automatically select/highlight in the JOBS listbox.

***selectOwners**

List of owners checked when limiting the jobs appearing on the Jobs listbox in the main xpbs window. Specify value as "Owners: <list_of_owners>". See -u option in `qselect(1B)` for format of <list_of_owners>.

***selectStates**

List of job states to look for (do not space separate) when limiting the jobs appearing on the Jobs listbox in the main xpbs window. Specify value as "Job_States: <states_string>". See -s option in `qselect(1B)` for format of <states_string>.

***selectRes**

List of resource amounts (space separated) to consult when limiting the jobs appearing on the Jobs

listbox in the main xpbs window. Specify value as "Resources: <res_string>". See -l option in `qselect(1B)` for format of <res_string>.

***selectExecTime**

The Execution Time attribute to consult when limiting the list of jobs appearing on the Jobs listbox in the main xpbs window. Specify value as "Queue_Time: <exec_time>". See -a option in `qselect(1B)` for format of <exec_time>.

***selectAcctName**

The name of the account that will be checked when limiting the jobs appearing on the Jobs listbox in the main xpbs window. Specify value as "Account_Name: <account_name>". See -A option in `qselect(1B)` for format of <account_name>.

***selectCheckpoint**

The checkpoint attribute relationship (including the logical operator) to consult when limiting the list of jobs appearing on the Jobs listbox in the main xpbs window. Specify value as "Checkpoint: <checkpoint_arg>". See -c option in `qselect(1B)` for format of <checkpoint_arg>.

***selectHold**

The hold types string to look for in a job when limiting the jobs appearing on the Jobs listbox in the main xpbs window. Specify value as "Hold_Types: <hold_string>". See -h option in `qselect(1B)` for format of <hold_string>.

***selectPriority**

The priority relationship (including the logical operator) to consult when limiting the list of jobs appearing on the Jobs listbox in the main xpbs window. Specify value as "Priority: <priority_value>". See -p option in `qselect(1B)` for format of <priority_value>.

***selectRerun**

The rerunnable attribute to consult when limiting the list of jobs appearing on the Jobs listbox in the main xpbs window. Specify value as "Rerunable: <rerun_val>". See -r option in `qselect(1B)` for format of <rerun_val>.

***selectJobName**

Name of the job that will be checked when limiting the jobs appearing on the Jobs listbox in the main xpbs window. Specify value as "Job_Name: <jobname>". See -N option in `qselect(1B)` for format of <jobname>.

***iconizeHostsView**

A boolean value (true or false) indicating whether or not to iconize the HOSTS region.

***iconizeQueuesView**

A boolean value (true or false) indicating whether or not to iconize the QUEUES region.

***iconizeJobsView**

A boolean value (true or false) indicating whether or not to iconize the JOBS region.

***iconizeInfoView**

A boolean value (true or false) indicating whether or not to iconize the INFO region.

***jobResourceList**

A curly-braced list of resource names as according to architecture known to xpbs. The format is as follows:

{ <arch-type1> resname1 resname2 ... resnameN }

{ <arch-type2> resname1 resname2 ... resnameN }

{ <arch-typeN> resname1 resname2 ... resnameN }

Chapter 6

Working with PBS Jobs

This chapter introduces the reader to various commands useful in working with PBS jobs. Covered topics include: modifying job attributes, holding and releasing jobs, sending messages to jobs, changing order of jobs within a queue, sending signals to jobs, and deleting jobs. In each section below, the command line method for accomplishing a particular task is presented first, followed by the `xpbs` method.

6.1 Modifying Job Attributes

Most attributes can be changed by the owner of the job (or a manager or operator) while the job is still queued. However, once a job begins execution, the only resources that can be modified are `cpus` and `walltime`. These can only be reduced.

When the `qalter -l` option is used to alter the resource list of a queued job, it is important to understand the interactions between altering the `select` directive and job limits.

If the job was submitted with an explicit `-l select=`, then vnode-level resources must be `qalter`d using the `-l select=` form. In this case a vnode level resource `RES` cannot be `qalter`d with the `-l RES` form.

For example:

Submit the job:

```
% qsub -l select=1:ncpus=2:mem=512mb jobscript
```

Job's ID is 230

`qalter` the job using `-l RES` form:

```
% qalter -l ncpus=4 230
```

Error reported by qalter:

```
qalter: Resource must only appear in "select"  
specification when select is used: ncpus 230
```

qalter the job using the "-l select=" form:

```
% qalter -l select=1:ncpus=4:mem=512mb 230
```

No error reported by qalter:

```
%
```

6.1.1 Changing the Selection Directive

If the selection directive is altered, the job limits for any consumable resource in the directive are also modified.

For example, if a job is queued with the following resource list:

```
select=2:ncpus=1:mem=5gb, ncpus=2, mem=10gb
```

and the selection directive is altered to request

```
select=3:ncpus=2:mem=6gb
```

then the job limits are reset to ncpus=6 and mem=18gb

6.1.2 Changing the Job-wide Limit

If the job-wide limit is modified, the corresponding resources in the selection directive are not modified. It would be impossible to determine where to apply the changes in a compound directive.

Reducing a job-wide limit to a new value less than the sum of the resource in the directive is strongly discouraged. This may produce a situation where the job is aborted during execution for exceeding its limits. The actual effect of such a modification is not specified.

A job's walltime may be altered at any time, except when the job is in the *Exiting* state, regardless of the initial value.

If a job is queued, requested modifications must still fit within the queue's and server's job resource limits. If a requested modification to a resource would exceed the queue's or server's job resource limits, the resource request will be rejected.

Resources are modified by using the `-l` option, either in chunks inside of selection statements, or in job-wide modifications using `resource_name=value` pairs. The selection statement is of the form:

```
-l select=[N:]chunk[+[N:]chunk ...]
```

where `N` specifies how many of that chunk, and a chunk is of the form:

```
resource_name=value[:resource_name=value ...]
```

Job-wide `resource_name=value` modifications are of the form:

```
-l resource_name=value[,resource_name=value ...]
```

It is an error to use a boolean resource as a job-wide limit.

Placement of jobs on vnodes is changed using the `place` statement:

```
-l place=modifier[:modifier]
```

where *modifier* is any combination of *group*, *excl*, *exclhost*, and/or one of *free*|*pack*|*scatter*|*vscatter*.

The usage syntax for `qalter` is:

```
qalter job-resources job-list
```

The following examples illustrate how to use the `qalter` command. First we list all the jobs of a particular user. Then we modify two attributes as shown (increasing the wall-clock time from 20 to 25 minutes, and changing the job name from “airfoil” to “engine”):

```
qstat -u barry
```

Job ID	User	Queue	Jobname	Sess	NDS	TSK	Mem	Time	Req'd	Elap
									S	Time
51.south	barry	workq	airfoil	930	--	1	--	0:16	R	0:01
54.south	barry	workq	airfoil	--	--	1	--	0:20	Q	--

```
qalter -l walltime=20:00 -N engine 54
```

```
qstat -a 54
```

Job ID	User	Queue	Jobname	Sess	NDS	TSK	Mem	Time	Req'd	Elap
									S	Time
54.south	barry	workq	engine	--	--	1	--	0:25	Q	--

To alter a job attribute via `xpbs`, first select the job(s) of interest, and then click on *modify* button. Doing so will bring up the *Modify Job Attributes* dialog box. From this window you may set the new values for any attribute you are permitted to change. Then click on the *confirm modify* button at the lower left of the window.

The `qalter` command can be used on job arrays, but not on subjobs or ranges of subjobs. When used with job arrays, any job array identifiers must be enclosed in double quotes, e.g.:

```
qalter -l walltime=25:00 "1234[] .south"
```

You cannot use the `qalter` command (or any other command) to alter a custom resource which has been created to be invisible or unrequestable. See [section 3.5.15, “Resource Permissions”, on page 42](#).

For more information, see the `qalter(1B)` manual page.

6.2 Holding and Releasing Jobs

PBS provides a pair of commands to hold and release jobs. To hold a job is to mark it as ineligible to run until the hold on the job is “released”.

The `qhold` command requests that a Server place one or more holds on a job. A job that has a hold is not eligible for execution. There are three types of holds: *user*, *operator*, and *system*. A user may place a *user* hold upon any job the user owns. An “operator”, who is a user with “operator privilege”, may place either an *user* or an *operator* hold on any job. The PBS Manager may place any hold on any job. The usage syntax of the `qhold` command is:

```
qhold [-h hold_list] job_identifier ...
```

Note that for a job array the `job_identifier` must be enclosed in double quotes.

The `hold_list` defines the type of holds to be placed on the job. The `hold_list` argument is a string consisting of one or more of the letters `u`, `p`, `o`, or `s` in any combination, or the letter `n`. The hold type associated with each letter is:

Table 6-1: Hold Types

Letter	Meaning
n	none - no hold type specified
u	user - the user may set and release this hold type
p	password - set if job fails due to a bad password; can be unset by the user

Table 6-1: Hold Types

Letter	Meaning
o	operator; require operator privilege to unset
s	system - requires manager privilege to unset

If no `-h` option is given, the *user* hold will be applied to the jobs described by the *job_identifier* operand list. If the job identified by *job_identifier* is in the queued, held, or waiting states, then all that occurs is that the hold type is added to the job. The job is then placed into held state if it resides in an execution queue.

If the job is running, then the following additional action is taken to interrupt the execution of the job. If the job is checkpointable, requesting a hold on a running job will cause (1) the job to be checkpointed, (2) the resources assigned to the job to be released, and (3) the job to be placed in the held state in the execution queue. If the job is not checkpointable, `qhold` will only set the requested hold attribute. This will have no effect unless the job is requested with the `qrerun` command. See [section 3.13.14.1, “Checkpointable Jobs”, on page 72](#).

The `qhold` command can be used on job arrays, but not on subjobs or ranges of subjobs. On job arrays, the `qhold` command can be applied only in the ‘Q’, ‘B’ or ‘W’ states. This will put the job array in the ‘H’, held, state. If any subjobs are running, they will run to completion. Job arrays cannot be moved in the ‘H’ state if any subjobs are running.

Checkpointing is not supported for job arrays. Even on systems that support checkpointing, no subjobs will be checkpointed -- they will run to completion.

Similarly, the `qrls` command releases a hold on a job. However, the user executing the `qrls` command must have the necessary privilege to release a given hold. The same rules apply for releasing a hold as exist for setting a hold.

The `qrls` command can only be used with job array objects, not with subjobs or ranges. The job array will be returned to its pre-hold state, which can be either ‘Q’, ‘B’, or ‘W’.

The usage syntax of the `qrls` command is:

```
qrls [ -h hold_list ] job_identifier ...
```

For job arrays, the *job_identifier* must be enclosed in double quotes.

The following examples illustrate how to use both the `qhold` and `qrls` commands. Notice that the state (“S”) column shows how the state of the job changes with the use of these two commands.

```
qstat -a 54
```

Job ID	User	Queue	Jobname	Sess	NDS	TSK	Mem	Time	Req'd	Elap
									S	Time
54.south	barry	workq	engine	--	--	1	--	0:20	Q	--

qhold 54**qstat -a 54**

Job ID	User	Queue	Jobname	Sess	NDS	TSK	Mem	Time	Req'd	Elap
									S	Time
54.south	barry	workq	engine	--	--	1	--	0:20	H	--

qrls -h u 54**qstat -a 54**

Job ID	User	Queue	Jobname	Sess	NDS	TSK	Mem	Time	Req'd	Elap
									S	Time
54.south	barry	workq	engine	--	--	1	--	0:20	Q	--

If you attempted to release a hold on a job which is not on hold, the request will be ignored. If you use the `qrls` command to release a hold on a job that had been previously running, and subsequently checkpointed, the hold will be released, and the job will return to the queued (Q) state (and be eligible to be scheduled to run when resources come available).

To hold (or release) a job using `xpbs`, first select the job(s) of interest, then click the *hold* (or *release*) button.

The `qrls` command does not run the job; it simply releases the hold and makes the job eligible to be run the next time the scheduler selects it.

6.3 Deleting Jobs

PBS provides the `qdel` command for deleting jobs. The `qdel` command deletes jobs in the order in which their job identifiers are presented to the command. A batch job may be deleted by its owner, a PBS operator, or a PBS administrator.

Example:

```
qdel 51
qdel 1234[ ] .server
```

Job array identifiers must be enclosed in double quotes.

Mail is sent for each job deleted unless you specify otherwise. Use the following option to `qdel` to prevent more email than you want from being sent:

```
-Wsuppress_email=<N>
```

`N` must be a non-negative integer. Make `N` the largest number of emails you wish to receive per `qdel` command. PBS will send one email for each deleted job, up to `N`. Note that a job array is one job, so deleting a job array results in one email being sent.

To delete a job using `xpbs`, first select the job(s) of interest, then click the *delete* button.

6.3.1 Deleting Finished and Moved Jobs

6.3.1.1 Deleting Finished Jobs

The `qdel` command does not affect finished jobs, whether this job finished at the local server or at the destination server. If you try to delete a finished job, you will get the following error:

```
qdel: Job <jobid> has finished
```

6.3.1.2 Deleting Moved Jobs

A job that has been moved to another server is either finished or still active, i.e. queued or running. If the moved job is active at the destination server, the `qdel` command deletes the job.

6.4 Sending Messages to Jobs

To send a message to a job is to write a message string into one or more output files of the job. Typically this is done to leave an informative message in the output of the job. Such messages can be written using the `qmsg` command.

IMPORTANT:

A message can only be sent to running jobs.

The usage syntax of the `qmsg` command is:

```
qmsg [-E ][-O ] message_string job_identifier
```

Example:

```
qmsg -O "output file message" 54
```

```
qmsg -O "output file message" "1234[.server]"
```

Job array identifiers must be enclosed in double quotes.

The `-E` option writes the message into the error file of the specified job(s). The `-O` option writes the message into the output file of the specified job(s). If neither option is specified, the message will be written to the error file of the job.

The first operand, *message_string*, is the message to be written. If the string contains blanks, the string must be quoted. If the final character of the string is not a newline, a newline character will be added when written to the job's file. All remaining operands are *job_identifiers* which specify the jobs to receive the message string. For example:

```
qmsg -E "hello to my error (.e) file" 55
```

```
qmsg -O "hello to my output (.o) file" 55
```

```
qmsg "this too will go to my error (.e) file" 55
```

To send a message to a job using `xpbs`, first select the job(s) of interest, then click the *msg* button. Doing so will launch the *Send Message to Job* dialog box. From this window, you may enter the message you wish to send and indicate whether it should be written to the standard output or the standard error file of the job. Click the *Send Message* button to complete the process.

6.5 Sending Signals to Jobs

The `qsig` command requests that a signal be sent to executing PBS jobs. The signal is sent to the session leader of the job. Usage syntax of the `qsig` command is:

```
qsig [-s signal ] job_identifier
```

Job array *job_identifiers* must be enclosed in double quotes.

If the `-s` option is not specified, `SIGTERM` is sent. If the `-s` option is specified, it declares which *signal* is sent to the job. The *signal* argument is either a signal name, e.g. `SIGKILL`, the signal name without the `SIG` prefix, e.g. `KILL`, or an unsigned signal number, e.g. `9`. The signal name `SIGNULL` is allowed; the Server will send the signal `0` to the job

which will have no effect. Not all signal names will be recognized by `qsig`. If it doesn't recognize the signal name, try issuing the signal number instead. The request to signal a batch job will be rejected if:

- The user is not authorized to signal the job.
- The job is not in the running state.
- The requested signal is not supported by the execution host.
- The job is exiting.

Two special signal names, “suspend” and “resume”, (note, all lower case), are used to suspend and resume jobs. When suspended, a job continues to occupy system resources but is not executing and is not charged for walltime. Manager or operator privilege is required to suspend or resume a job.

The three examples below all send a signal 9 (SIGKILL) to job 34:

```
qsig -s SIGKILL 34
```

```
qsig -s KILL 34
```

IMPORTANT:

On most UNIX systems the command “kill -l” (that's ‘minus ell’) will list all the available signals.

To send a signal to a job using `xpbs`, first select the job(s) of interest, then click the *signal* button. Doing so will launch the *Signal Running Job* dialog box.

From this window, you may click on any of the common signals, or you may enter the signal number or signal name you wish to send to the job. Click the *Signal* button to complete the process.

6.6 Changing Order of Jobs

PBS provides the **qorder** command to change the order of two jobs, within or across queues. To order two jobs is to exchange the jobs' positions in the queue or queues in which the jobs reside. If job1 is at position 3 in queue A and job2 is at position 4 in queue B, qordering them will result in job1 being in position 4 in queue B and job2 being in position 3 in queue A.

No attribute of the job (such as priority) is changed. The impact of changing the order within the queue(s) is dependent on local job scheduling policy; contact your systems administrator for details.

Usage of the **qorder** command is:

```
qorder job_identifier1 job_identifier2
```

Job array identifiers must be enclosed in double quotes.

Both operands are *job_identifiers* which specify the jobs to be exchanged.

```
qstat -u bob
```

Job ID	User	Queue	Jobname	Sess	NDS	TSK	Mem	Time	Req'd	Elap
									S	Time
54.south	bob	workq	twinkie	--	--	1	--	0:20	Q	--
63[.south	bob	workq	airfoil	--	--	1	--	0:13	Q	--

```
qorder 54 "63["
```

```
qstat -u bob
```

Job ID	User	Queue	Jobname	Sess	NDS	TSK	Mem	Time	Req'd	Elap
									S	Time
63[.south	bob	workq	airfoil	--	--	1	--	0:13	Q	--
54.south	bob	workq	twinkie	--	--	1	--	0:20	Q	--

To change the order of two jobs using `xpbs`, select the two jobs, and then click the *order* button.

6.6.1 Restrictions

- The two jobs must be located at the same Server, and both jobs must be owned by the user.
- A job in the running state cannot be reordered.
- The `qorder` command can be used with entire job arrays, but not on subjobs or ranges. Reordering a job array changes the queue order of the job array in relation to other jobs or job arrays in the queue.

6.7 Moving Jobs Between Queues

PBS provides the **qmove** command to move jobs between different queues (even queues on different Servers). To move a job is to remove the job from the queue in which it resides and instantiate the job in another queue.

IMPORTANT:

A job in the running state cannot be moved.

The usage syntax of the **qmove** command is:

qmove destination job_identifier(s)

Job array **job_identifiers** must be enclosed in double quotes.

The first operand is the new destination for

queue
@server
queue@server

If the *destination* operand describes only a queue, then **qmove** will move jobs into the queue of the specified name at the job's current Server. If the *destination* operand describes only a Server, then **qmove** will move jobs into the default queue at that Server. If the *destination* operand describes both a queue and a Server, then **qmove** will move the jobs into the specified queue at the specified Server. All following operands are *job_identifiers* which specify the jobs to be moved to the new *destination*.

To move jobs between queues or between Servers using **xpbs**, select the job(s) of interest, and then click the move button. Doing so will launch the Move Job dialog box from which you can select the queue and/or Server to which you want the job(s) moved.

The **qmove** command can only be used with job array objects, not with subjobs or ranges. Job arrays can only be moved from one server to another if they are in the 'Q', 'H', or 'W' states, and only if there are no running subjobs. The state of the job array object is preserved in the move. The job array will run to completion on the new server.

As with jobs, a **qstat** on the server from which the job array was moved will not show the job array. A **qstat** on the job array object will be redirected to the new server.

Note: The subjob accounting records will be split between the two servers.

6.8 Converting a Job into a Reservation Job

The `pbs_rsub` command can be used to convert a normal job into a reservation job that will run as soon as possible. PBS creates a reservation queue and a reservation, and moves the job into the queue. Other jobs can also be moved into that queue via `qmove(1B)` or submitted to that queue via `qsub(1B)`. The reservation is called an ASAP reservation.

The format for converting a normal job into a reservation job is:

```
pbs_rsub [-l walltime=time] -W qmove=job_identifier
```

Example:

```
pbs_rsub -W qmove=54  
pbs_rsub -W qmove="1234[.]server"
```

The `-R` and `-E` options to `pbs_rsub` are disabled when using the `-W qmove` option.

For more information, see ["Advance and Standing Reservation of Resources" on page 209](#), and the `pbs_rsub(1B)`, `qsub(1B)` and `qmove(1B)` manual pages.

A job's default walltime is 5 years. Therefore an ASAP reservation's start time can be in 5 years, if all the jobs in the system have the default walltime.

You cannot use the `pbs_rsub` command (or any other command) to request a custom resource which has been created to be invisible or unrequestable. See [section 3.5.15, "Resource Permissions", on page 42](#).

6.9 Using Job History Information

6.9.1 Introduction

PBS Professional can provide job history information, including what the submission parameters were, whether the job started execution, whether execution succeeded, whether staging out of results succeeded, and which resources were used.

PBS can keep job history for jobs which have finished execution, were deleted, or were moved to another server.

6.9.2 Definitions

Moved jobs

Jobs which were moved to another server

Finished jobs

Jobs whose execution is done, for any reason:

- Jobs which finished execution successfully and exited
- Jobs terminated by PBS while running
- Jobs whose execution failed because of system or network failure
- Jobs which were deleted before they could start execution

6.9.3 Job History Information

PBS can keep all job attribute information, including the following:

- Submission parameters
- Whether the job started execution
- Whether execution succeeded
- Whether staging out of results succeeded
- Which resources were used

PBS keeps job history for the following jobs:

- Jobs that have finished execution
- Jobs that were deleted
- Jobs that were moved to another server

The job history for finished and moved jobs is preserved and available for the specified duration. After the duration has expired, PBS deletes the job history information and it is no longer available. The state of a finished job is *F*, and the state of a moved job is *M*. See [“Job States” on page 411 of the PBS Professional Reference Guide](#).

Subjobs are not considered finished jobs until the parent array job is finished, which happens when all of its subjobs have terminated execution.

6.9.4 Working With Finished and Moved Jobs

6.9.4.1 Working With Moved Jobs

You can use the following commands with moved jobs. They will function as they do with normal jobs.

```
qalter
qhold
qmove
qmsg
qorder
qrerun
qrls
qrun
qsig
```

6.9.4.2 PBS Commands and Finished Jobs

The commands listed above cannot be used with finished jobs, whether they finished at the local server or a remote server. These jobs are no longer running; PBS is storing their information, and this information cannot be altered. Trying to use one of the above commands with a finished job results in the following error message:

```
<command name>: Job <jobid> has finished
```

6.9.5 Viewing Information for Finished and Moved Jobs

You can view information for finished and moved jobs in the same way as for queued and running jobs, as long as the job history is still being preserved.

The `-x` option to the `qstat` command allows you to see information for all jobs, whether they are running, queued, finished or moved. This information is presented in standard format. The `-H` option to the `qstat` command allows you to see alternate-format information for finished or moved jobs only. See [section 7.1.20, “Viewing Job History”, on page 183](#).

6.9.5.1 UNIX/Linux:

```
qstat -fx `qselect -x -s "MF"``
```

6.9.5.2 Windows:

```
for /F "usebackq" %%j in (`"\Program Files\ PBSPro\ exec\ bin\qselect" -x
-s MF`)
do ("\"Program Files\PBS Pro\exec\bin\qstat" -fx %%j)
```

6.9.6 Listing Job Identifiers of Finished and Moved Jobs

You can list identifiers of finished and moved jobs in the same way as for queued and running jobs, as long as the job history is still being preserved.

The `-x` option to the `qselect` command allows you to list job identifiers for all jobs, whether they are running, queued, finished or moved. The `-H` option to the `qselect` command allows you to list job identifiers for finished or moved jobs only. See [section 7.3, “The qselect Command”, on page 187](#).

6.9.6.1 Listing Jobs by Time Attributes

You can use the `qselect` command to list queued, running, finished and moved jobs, job arrays, and subjobs according to their time attributes. The `-t` option to the `qselect` command allows you to specify how you want to select based on time attributes. You can also use the `-t` option twice to bracket a time period. See [section 7.3, “The qselect Command”, on page 187](#).

Example 6-1: Select jobs with end time between noon and 3PM.

```
qselect -te.gt.09251200 -te.lt.09251500
```

Example 6-2: Select finished and moved jobs with start time between noon and 3PM.

```
qselect -x -s "MF" -ts.gt.09251200 -ts.lt.09251500
```

Example 6-3: Select all jobs with creation time between noon and 3PM

```
qselect -x -tc.gt.09251200 -tc.lt.09251500
```

Example 6-4: Select all jobs including finished and moved jobs with qtime of 2.30PM (default relation is `.eq.`)

```
qselect -x -tq09251430
```


6.9.6.2 Deleting Moved and Finished Jobs

You can use the `qdel -x` option to delete job histories. This option also deletes any specified jobs that are queued, running, held, suspended, finished, or moved. When you use this, you are deleting the job and its history in one step. If you use the `qdel` command without the `-x` option, you delete the job, but not the job history, and you cannot delete a moved or finished job.

Unless you are an administrator or an operator, you can delete only your own jobs.

See [“qdel” on page 143 of the PBS Professional Reference Guide](#).

Chapter 7

Checking Job / System Status

This chapter introduces several PBS commands useful for checking status of jobs, queues, and PBS Servers. Examples for use are included, as are instructions on how to accomplish the same task using the `xpbs` graphical interface.

7.1 The **qstat** Command

The **qstat** command is used to request the status of jobs, queues, and the PBS Server. The requested status is written to standard output stream (usually the user's terminal). When requesting job status, any jobs for which the user does not have view privilege are not displayed. For detailed usage information, see [“qstat” on page 194 of the PBS Professional Reference Guide](#).

Usage:

```
qstat [-J] [-p] [-t] [-x] [[ job_identifier | destination ] ...]
qstat -f [-J] [-p] [-t] [-x] [[ job_identifier | destination ] ...]
qstat [-a [-w] | -H | -i | -r ] [-G|-M] [-J] [-n [-l][-w]] [-s [-l][-w]] [-t]
      [-T [-w]] [-u user] [[job_id | destination] ...]
qstat -Q [-f] [ destination... ]
qstat -q [-G|-M] [ destination... ]
qstat -B [-f] [ server_name... ]
qstat --version
```

7.1.1 Checking Job Status

Executing the `qstat` command without any options displays job information in the default format. (An alternative display format is also provided, and is discussed below.) The default display includes the following information:

- The job identifier assigned by PBS
- The job name given by the submitter
- The job owner
- The CPU time used
- The job state
- The queue in which the job resides

See [“Job States” on page 411 of the PBS Professional Reference Guide](#).

The following example illustrates the default display of `qstat`.

```
qstat
Job id      Name      User      Time Use S Queue
-----
16.south    aims14    user1      0 H workq
18.south    aims14    user1      0 W workq
26.south    airfoil   barry     00:21:03 R workq
27.south    airfoil   barry     21:09:12 R workq
28.south    myjob     user1      0 Q workq
29.south    tns3d     susan      0 Q workq
30.south    airfoil   barry      0 Q workq
31.south    seq_35_3  donald     0 Q workq
```

An alternative display (accessed via the “-a” option) is also provided that includes extra information about jobs, including the following additional fields:

```
Session ID
Number of vnodes requested
Number of parallel tasks (or CPUs)
Requested amount of memory
Requested amount of wall clock time
Walltime or CPU time, whichever submitter specified, if job is running.
```

```
qstat -a
```

Job ID	User	Queue	Jobname	Ses	NDS	TSK	Mem	Time	S	Time
16.south	user1	workq	aims14	--	--	1	--	0:01	H	--
18.south	user1	workq	aims14	--	--	1	--	0:01	W	--
51.south	barry	workq	airfoil	930	--	1	--	0:13	R	0:01
52.south	user1	workq	myjob	--	--	1	--	0:10	Q	--
53.south	susan	workq	tns3d	--	--	1	--	0:20	Q	--
54.south	barry	workq	airfoil	--	--	1	--	0:13	Q	--
55.south	donald	workq	seq_35_	--	--	1	--	2:00	Q	--

Other options which utilize the alternative display are discussed in subsequent sections of this chapter.

7.1.2 Viewing Specific Information

When requesting queue or Server status `qstat` will output information about each destination. The various options to `qstat` take as an operand either a job identifier or a destination. If the operand is a job identifier, it must be in the following form:

sequence_number[.server_name][@server]

where *sequence_number*.*server_name* is the job identifier assigned at submittal time, see `qsub`. If the *.server_name* is omitted, the name of the default Server will be used. If *@server* is supplied, the request will be for the job identifier currently at that Server.

If the operand is a destination identifier, it takes one of the following three forms:

queue

@server

queue@server

If *queue* is specified, the request is for status of all jobs in that queue at the default Server. If the *@server* form is given, the request is for status of all jobs at that Server. If a full destination identifier, *queue@server*, is given, the request is for status of all jobs in the named *queue* at the named *server*.

IMPORTANT:

If a PBS Server is not specified on the `qstat` command line, the default Server will be used. (See discussion of **PBS_DEFAULT** in [section 2.18, “Environment Variables”](#), on page 17.)

7.1.3 Checking Server Status

The “-B” option to `qstat` displays the status of the specified PBS Batch Server. One line of output is generated for each Server queried. The three letter abbreviations correspond to various job limits and counts as follows: Maximum, Total, Queued, Running, Held, Waiting, Transiting, and Exiting. The last column gives the status of the Server itself: active, idle, or scheduling.

```
qstat -B
```

Server	Max	Tot	Que	Run	Hld	Wat	Trn	Ext	Status
fast.domain	0	14	13	1	0	0	0	0	Active

When querying jobs, Servers, or queues, you can add the “-f” option to `qstat` to change the display to the *full* or *long* display. For example, the Server status shown above would be expanded using “-f” as shown below:

```
qstat -Bf
Server: fast.mydomain.com
  server_state = Active
  scheduling = True
  total_jobs = 14
  state_count = Transit:0 Queued:13 Held:0 Waiting:0
                Running:1 Exiting:0
  managers = user1@fast.mydomain.com
  default_queue = workq
  log_events = 511
  mail_from = adm
  query_other_jobs = True
  resources_available.mem = 64mb
  resources_available.ncpus = 2
  resources_default.ncpus = 1
  resources_assigned.ncpus = 1
  resources_assigned.nodect = 1
  scheduler_iteration = 600
  pbs_version = PBSPro_12.41640
```

7.1.4 Checking Queue Status

The “-Q” option to `qstat` displays the status of all (or any specified) queues at the (optionally specified) PBS Server. One line of output is generated for each queue queried. The three letter abbreviations correspond to limits, queue states, and job counts as follows: Maximum, Total, Enabled Status, Started Status, Queued, Running, Held, Waiting, Transiting, and Exiting. The last column gives the type of the queue: *routing* or *execution*.

```
qstat -Q
Queue Max Tot Ena Str Que Run Hld Wat Trn Ext Type
-----
workq  0 10 yes yes  7  1  1  1  0  0 Execution
```

The full display for a queue provides additional information:

```
qstat -Qf
Queue: workq
    queue_type = Execution
    total_jobs = 10
    state_count = Transit:0 Queued:7 Held:1 Waiting:1
                  Running:1 Exiting:0
    resources_assigned.ncpus = 1
    hasnodes = False
    enabled = True
    started = True
```

7.1.5 Viewing Job Information

We saw above that the “-f” option could be used to display full or long information for queues and Servers. The same applies to jobs. By specifying the “-f” option and a job identifier, PBS will print all information known about the job (e.g. resources requested, resource

limits, owner, source, destination, queue, etc.) as shown in the following example. (See [“Job Attributes” on page 375 of the PBS Professional Reference Guide](#) for a description of attributes.)

```
qstat -f 13
Job Id: 13.host1
  Job_Name = STDIN
  Job_Owner = user1@host2
  resources_used.cputpercent = 0
  resources_used.cput = 00:00:00
  resources_used.mem = 2408kb
  resources_used.ncpus = 1
  resources_used.vmem = 12392kb
  resources_used.walltime = 00:01:31
  job_state = R
  queue = workq
  server = host1
  Checkpoint = u
  ctime = Thu Apr  2 12:07:05 2010
  Error_Path = host2:/home/user1/STDIN.e13
  exec_host = host2/0
  exec_vnode = (host3:ncpus=1)
  Hold_Types = n
  Join_Path = n
  Keep_Files = n
  Mail_Points = a
  mtime = Thu Apr  2 12:07:07 2010
  Output_Path = host2:/home/user1/STDIN.o13
  Priority = 0
  qtime = Thu Apr  2 12:07:05 2010
  Rerunable = True
  Resource_List.ncpus = 1
  Resource_List.nodect = 1
  Resource_List.place = free
  Resource_List.select = host=host3
  stime = Thu Apr  2 12:07:08 2010
  session_id = 32704
```

```

jobdir = /home/user1
substate = 42
Variable_List = PBS_O_HOME=/home/user1,PBS_O_LANG=en_US.UTF-8,
                PBS_O_LOGNAME=user1,
                PBS_O_PATH=/opt/gnome/sbin:/root/bin:/usr/local/bin:/usr/bin:/usr/
X11R
                6/bin:/bin:/usr/games:/opt/gnome/bin:/opt/kde3/bin:/usr/lib/mit/
bin:/us
                r/lib/mit/sbin,PBS_O_MAIL=/var/mail/root,PBS_O_SHELL=/bin/bash,
                PBS_O_HOST=host2,PBS_O_WORKDIR=/home/user1,PBS_O_SYSTEM=Linux,
                PBS_O_QUEUE=workq
comment = Job run at Thu Apr 02 at 12:07 on (host3:ncpus=1)
alt_id = <dom0:JobID xmlns:dom0="http://schemas.microsoft.com/
HPCS2008/hpcb
        p">149</dom0:JobID>
etime = Thu Apr  2 12:07:05 2010
Submit_arguments = -lselect=host=host3 -- ping -n 100 127.0.0.1
executable = <jsdl-hpcpa:Executable>ping</jsdl-hpcpa:Executable>
argument_list = <jsdl-hpcpa:Argument>-n</jsdl-hpcpa:Argument><jsdl-
hpcpa:Ar
                gument>100</jsdl-hpcpa:Argument><jsdl-hpcpa:Argument>127.0.0.1</
jsdl-hp
                cpa:Argument>

```

7.1.6 List User-Specific Jobs

The “-u” option to `qstat` displays jobs owned by any of a list of user names specified. The syntax of the list of users is:

```
user_name[@host][,user_name[@host],...]
```


Host names are not required, and may be “wild carded” on the left end, e.g. “*.mydo-main.com”. *user_name* without a “@host” is equivalent to “*user_name*@*”, that is at any host.

qstat -u user1

Job ID	User	Queue	Jobname	Sess	NDS	TSK	Mem	Req'd Time	Elap S	Time
16.south	user1	workq	aims14	--	--	1	--	0:01	H	--
18.south	user1	workq	aims14	--	--	1	--	0:01	W	--
52.south	user1	workq	my_job	--	--	1	--	0:10	Q	--

qstat -u user1,barry

51.south	barry	workq	airfoil	930	--	1	--	0:13	R	0:01
52.south	user1	workq	my_job	--	--	1	--	0:10	Q	--
54.south	barry	workq	airfoil	--	--	1	--	0:13	Q	--

7.1.7 List Running Jobs

The “-r” option to **qstat** displays the status of all running jobs at the (optionally specified) PBS Server. Running jobs include those that are running and suspended. One line of output is generated for each job reported, and the information is presented in the alternative display. For example:

qstat -r

host1:

Job ID	User	Queue	Jobname	Sess	NDS	TSK	Mem	Req'd Time	Req'd S	Elap Time
43.host1	user1	workq	STDIN	4693	1	1	--	--	R	00:00

7.1.8 List Non-Running Jobs

The “-i” option to `qstat` displays the status of all non-running jobs at the (optionally specified) PBS Server. Non-running jobs include those that are queued, held, and waiting. One line of output is generated for each job reported, and the information is presented in the alternative display (see description above). For example:

qstat -i

host1:

Job ID	User	Queue	Jobname	Sess	NDS	TSK	Req'd Mem	Req'd Time	Elap S	Time
44[].host1	user1	workq	STDIN	--	1	1	--	--	Q	--

7.1.9 Display Size in Gigabytes

The “-G” option to `qstat` displays all jobs at the requested (or default) Server using the alternative display, showing all size information in gigabytes (GB) rather than the default of smallest displayable units. Note that if the size specified is less than 1 GB, then the amount is rounded up to 1 GB. For example:

qstat -G

host1:

Job ID	User	Queue	Jobname	Sess	NDS	TSK	Req'd Mem	Req'd Time	Elap S	Time
43.host1	user1	workq	STDIN	4693	1	1	--	--	R	00:05
44[].host1	user1	workq	STDIN	--	1	1	--	--	Q	--
45.host1	user1	workq	STDIN	--	1	1	1gb	--	Q	--

7.1.10 Display Size in Megawords

The “-M” option to `qstat` displays all jobs at the requested (or default) Server using the alternative display, showing all size information in megawords (MW) rather than the default of smallest displayable units. A word is considered to be 8 bytes. For example:

qstat -M

 host1:

							Req'd	Req'd	Elap	
Job ID	User	Queue	Jobname	Sess	NDS	TSK	Mem	Time	S	Time
43.host1	user1	workq	STDIN	4693	1	1	--	--	R	00:05
44[.].host1	user1	workq	STDIN	--	1	1	--	--	Q	--
45.host1	user1	workq	STDIN	--	1	1	25mw	--	Q	--

7.1.11 List Hosts Assigned to Jobs

The “-n” option to `qstat` displays the hosts allocated to any running job at the (optionally specified) PBS Server, in addition to the other information presented in the alternative display. The host information is printed immediately below the job (see job 51 in the example below), and includes the host name and number of virtual processors assigned to the job (i.e. “south/0”, where “south” is the host name, followed by the virtual processor(s) assigned.). A text string of “--” is printed for non-running jobs. Notice the differences between the queued and running jobs in the example below:

qstat -n

							Req'd	Elap		
Job ID	User	Queue	Jobname	Sess	NDS	TSK	Mem	Time	S	Time
16.south	user1	workq	aims14	--	--	1	--	0:01	H	--
--										
18.south	user1	workq	aims14	--	--	1	--	0:01	W	--
--										
51.south	barry	workq	airfoil	930	--	1	--	0:13	R	
								0:01	south/0	
52.south	user1	workq	my_job	--	--	1	--	0:10	Q	--
--										

7.1.12 Display Job Comments

The “-s” option to `qstat` displays the job comments, in addition to the other information presented in the alternative display. The job comment is printed immediately below the job. By default the job comment is updated by the Scheduler with the reason why a given job is not running, or when the job began executing. A text string of “--” is printed for jobs whose comment has not yet been set. The example below illustrates the different type of messages that may be displayed:

qstat -s

Job ID	User	Queue	Jobname	Sess	NDS	TSK	Mem	Time	S	Time	Req'd	Elap
16.south	user1	workq	aims14	--	--	1	--	0:01	H	--		
Job held by user1 on Wed Aug 22 13:06:11 2004												
18.south	user1	workq	aims14	--	--	1	--	0:01	W	--		
Waiting on user requested start time												
51.south	barry	workq	airfoil	930	--	1	--	0:13	R	0:01		
Job run on host south - started Thu Aug 23 at 10:56												
52.south	user1	workq	my_job	--	--	1	--	0:10	Q	--		
Not Running: No available resources on nodes												
57.south	susan	workq	solver	--	--	2	--	0:20	Q	--		
--												

7.1.13 Display Queue Limits

The “-q” option to `qstat` displays any limits set on the requested (or default) queues. Since PBS is shipped with no queue limits set, any visible limits will be site-specific. The limits are listed in the format shown below.

```
qstat -q
```

```
server: south
```

Queue	Memory	CPU	Time	Walltime	Node	Run	Que	Lim	State
-----	-----	-----	-----	-----	-----	-----	-----	-----	-----
workq	--	--	--	--	--	1	8	--	E R

7.1.14 Show State of Job, Job Array or Subjob

The “-t” option to `qstat` will show the state of a job, a job array object, and all non-X sub-jobs.

The “-J” option to `qstat` will show only the state of job arrays.

The combination of “-J” and “-t” options to `qstat` will show only the state of subjobs.

For example:

```
qstat -t
```

Job ID	Name	User	Time Use	S	Queue
-----	-----	-----	-----	-	-----
44[] .host1	STDIN	user1	0	B	workq
44[1] .host1	STDIN	user1	00:00:00	R	workq
44[2] .host1	STDIN	user1	0	Q	workq
44[3] .host1	STDIN	user1	0	Q	workq

```
qstat -J
```

Job ID	Name	User	Time Use	S	Queue
-----	-----	-----	-----	-	-----
44[] .host1	STDIN	user1	0	B	workq

```
$ qstat -Jt
```

Job ID	Name	User	Time Use	S	Queue
44[1].host1	STDIN	user1	00:00:00	R	workq
44[2].host1	STDIN	user1	0	Q	workq
44[3].host1	STDIN	user1	0	Q	workq

7.1.15 Viewing Job Start Time

There are two ways you can find the job's start time. If the job is still running, you can do a `qstat -f` and look for the `stime` attribute. If the job has finished, you look in the accounting log for the `S` record for the job. For an array job, only the `S` record is available.

7.1.16 Viewing Job Status in Wide Format

The `-w qstat` option displays job status in wide format. The total width of the display is extended from 80 characters to 120 characters. The Job ID column can be up to 30 characters wide, while the Username, Queue, and Jobname column can be up to 15 characters wide. The SessID column can be up to eight characters wide, and the NDS column can be up to four characters wide.

Note: You can use this option only with the `-a`, `-n`, or `-s qstat` options.

7.1.17 Print Job Array Percentage Completed

The `-p` option to `qstat` prints the default display, with a column for Percentage Completed. For a job array, this is the number of subjobs completed and deleted, divided by the total number of subjobs. For example:

```
qstat -p
```

Job ID	Name	User	% done	S	Queue
44[].host1	STDIN	user1	40	B	workq

7.1.18 Getting Information on Jobs Moved to Another Server

If your job is running at another server, you can use the `qstat` command to see its status. If your site is using peer scheduling, your job may be moved to a server that is not your default server. When that happens, you will need to give the job ID as an argument to `qstat`. If you use only “`qstat`”, your job will not appear to exist. For example: you submit a job to ServerA, and it returns the jobid as “123.ServerA”. Then 123.ServerA is moved to ServerB. In this case, use

```
qstat 123
```

or

```
qstat 123.ServerA
```

to get information about your job. ServerA will query ServerB for the information. To list all jobs at ServerB, you can use:

```
qstat @ServerB
```

If you use “`qstat`” without the job ID, the job will not appear to exist.

7.1.19 Viewing Resources Allocated to a Job

The `exec_vnode` attribute displayed via `qstat` shows the resources allocated from each vnode for the job.

The `exec_vnode` line looks like:

```
exec_vnode = (<vnode name>:ncpus=W:mem=X)+(<vnode name>:ncpus=Y:mem=Z)
```

For example, a job requesting

```
-l select=2:ncpus=1:mem=1gb+1:ncpus=4:mem=2gb
```

gets an `exec_vnode` of

```
exec_vnode = (VNA:ncpus=1:mem=1gb)+(VNB:ncpus=1:mem=1gb)  
+(VNC:ncpus=4:mem=2gb)
```

Note that the vnodes and resources required to satisfy a chunk are grouped by parentheses. In the example above, if two vnodes on a single host were required to satisfy the last chunk, the `exec_vnode` might be:

```
exec_vnode = (VNA:ncpus=1:mem=1gb)+(VNB:ncpus=1:mem=1gb)  
+(VNC1:ncpus=2:mem=1gb+VNC2:ncpus=2:mem=1gb)
```

Note also that if a vnode is allocated to a job because the job requests an arrangement of *exclhost*, only the vnode name appears in the chunk. For example, if a job requesting

```
-l select 2:ncpus=4 -l place = exclhost
```

is placed on a host with 4 vnodes, each with 4 CPUs, the **exec_vnode** attribute looks like this:

```
exec_vnode = (VN0:ncpus=4)+(VN1:ncpus=4)+(VN2)+(VN3)
```

You cannot use the **qstat** command to view any custom resource which has been created to be invisible or unrequestable, whether this resource is on a queue, the server, or is a job attribute. See [section 3.5.15, “Resource Permissions”, on page 42](#).

7.1.19.1 Resources for Requeued Jobs

When a job is requeued due to an error in the prologue or initialization, the job’s **exec_host** and **exec_vnode** attributes are cleared. The only exception is when the job is checkpointed and must be rerun on the exact same system. In this case, the **exec_host** and **exec_vnode** attributes are preserved.

7.1.20 Viewing Job History

You can view information for jobs that have finished or were moved, as long as that information is still being stored by PBS. See [section 6.9, “Using Job History Information”, on page 164](#).

You can view the same attribute information regardless of whether the job is queued, running, finished, or moved, as long as job history information is being preserved.

7.1.20.1 Job History In Standard Format

You can use the **-x** option to the **qstat** command to see information for finished, moved, queued, and running jobs, in standard format.

Usage:

```
qstat -x
```

Displays information for queued, running, finished, and moved jobs, in standard format.

```
qstat -x <job ID>
```

Displays information for a job, regardless of its state, in standard format.

Example 7-1: Showing finished and moved jobs with queued and running jobs:

```
qstat -x
```

Job id	Name	User	Time Use	S	Queue
-----	-----	-----	-----	---	-----
101.server1	STDIN	user1	00:00:00	F	workq
102.server1	STDIN	user1	00:00:00	M	destq@server2
103.server1	STDIN	user1	00:00:00	R	workq
104.server1	STDIN	user1	00:00:00	Q	workq

To see status for jobs, job arrays and subjobs that are queued, running, finished, and moved, use `qstat -xt`.

To see status for job arrays that are queued, running, finished, or moved, use `qstat -xJ`.

When information for a moved job is displayed, the destination queue and server are shown as `<queue>@<server>`.

Example 7-2: `qstat -x` output for moved job: destination queue is `destq`, and destination server is `server2`.

Job id	Name	User	Time Use	S	Queue
-----	-----	-----	-----	---	-----
101.sequoia	STDIN	user1	00:00:00	F	workq
102.sequoia	STDIN	user1	00:00:00	M	destq@server2
103.sequoia	STDIN	user1	00:00:00	R	workq

Example 7-3: Viewing moved job:

- There are three servers with hostnames ServerA, ServerB, and ServerC
- User1 submits job 123 to ServerA.
- After some time, User1 moves the job to ServerB.
- After more time, the administrator moves the job to QueueC at ServerC.

This means:

- The `qstat` command will show `QueueC@ServerC` for job 123.

7.1.20.2 Job History In Alternate Format

You can use the `-H` option to the `qstat` command to see job history for finished or moved jobs in alternate format.

Usage:

qstat -H

Displays information for finished or moved jobs, in alternate format

qstat -H job identifier

Displays information for that job in alternate format, whether or not it is finished or moved

qstat -H destination

Displays information for finished or moved jobs at that destination

Example 7-4: Job history in alternate format:

qstat -H

							Req'd	Req'd	Elap
Job ID	Username	Queue	Jobname	SessID	NDS	TSK	Memory	Time	S Time
-----	-----	----	-----	-----	---	---	-----	----	--
101.S1	user1	workq	STDIN	5168	1	1	--	--	F 00:00
102.S1	user1	Q1@S2	STDIN	--	1	2	--	--	M --

To see alternate-format status for jobs, job arrays and subjobs that are finished and moved, use `qstat -Ht`.

To see alternate-format status for job arrays that are finished or moved, use `qstat -HJ`.

The `-H` option is incompatible with the `-a`, `-i` and `-r` options.

7.1.21 Viewing Estimated Start Times For Jobs

You can view the estimated start times and vnodes of jobs using the `qstat` command. If you use the `-T` option to `qstat` when viewing job information, the *Elap Time* field is replaced with the *Est Start* field. Running jobs are shown above queued jobs.

See [“qstat” on page 194 of the PBS Professional Reference Guide](#).

If the estimated start time or vnode information is invisible to unprivileged users, no estimated start time or vnode information is available via `qstat`.

Example output:

qstat -T

Job ID	Username	Queue	Jobname	SessID	NDS	TSK	Memory	Req'd Time	Req'd S	Est Start
5.host1	user1	workq	foojob	12345	1	1	128mb	00:10	R	--
9.host1	user1	workq	foojob	--	1	1	128mb	00:10	Q	11:30
10.host1	user1	workq	foojob	--	1	1	128mb	00:10	Q	Tu 15
7.host1	user1	workq	foojob	--	1	1	128mb	00:10	Q	Jul
8.host1	user1	workq	foojob	--	1	1	128mb	00:10	Q	2010
11.host1	user1	workq	foojob	--	1	1	128mb	00:10	Q	>5yrs
13.host1	user1	workq	foojob	--	1	1	128mb	00:10	Q	--

If the start time for a job cannot be estimated, the start time behaves as if it is unset. For `qstat -T`, the start time appears as a question mark ("?"). for `qstat -f`, the start time appears as a time in the past.

7.1.22 Caveats

- MOM periodically polls jobs for usage by the jobs running on her host, collects the results, and reports this to the server. When a job exits, she polls again to get the final tally of usage for that job.

For example, MOM polls the running jobs at times T1, T2, T4, T8, T16, T24, and so on.

The output shown by a `qstat` during the window of time between T8 and T16 shows the resource usage up to T8.

If the `qstat` is done at T17, the output shows usage up through T16. If the job ends at T20, the accounting log (and the final log message, and the email to the user if "`qsub -me`" was used in job submission) contains usage through T20.

- The final report does not include the epilogue. The time required for the epilogue is treated as system overhead.
- The order in which jobs are displayed is undefined.

7.2 Viewing Job / System Status with `xpbs`

The main display of `xpbs` shows a brief listing of all selected Servers, all queues on those Servers, and any jobs in those queues that match the *selection criteria* (discussed below). Servers are listed in the HOST panel near the top of the display.

To view detailed information about a given Server (i.e. similar to that produced by “`qstat -fB`”) select the Server in question, then click the “Detail” button. Likewise, for details on a given queue (i.e. similar to that produced by “`qstat -fQ`”) select the queue in question, then click its corresponding “Detail” button. The same applies for jobs as well (i.e. “`qstat -f`”). You can view detailed information on any displayed job by selecting it, and then clicking on the “Detail” button. Note that the list of jobs displayed will be dependent upon the Selection Criteria currently selected. This is discussed in the `xpbs` portion of the next section.

7.3 The `qselect` Command

The `qselect` command provides a method to list the job identifier of those jobs, job arrays or subjobs which meet a list of selection criteria. Jobs are selected from those owned by a single Server. When `qselect` successfully completes, it will have written to standard output a list of zero or more job identifiers which meet the criteria specified by the options. Each option acts as a filter restricting the number of jobs which might be listed. With no options, the `qselect` command will list all jobs at the Server which the user is authorized to list (query status of). The `-u` option may be used to limit the selection to jobs owned by this user or other specified users.

For a description of the `qselect` command, see [“qselect” on page 183 of the PBS Professional Reference Guide](#).

For example, say you want to list all jobs owned by user “barry” that requested more than 16 CPUs. You could use the following `qselect` command syntax:

```
qselect -u barry -l ncpus.gt.16  
121.south  
133.south  
154.south
```

Notice that what is returned is the job identifiers of jobs that match the selection criteria. This may or may not be enough information for your purposes. Many users will use shell syntax to pass the list of job identifiers directly into `qstat` for viewing purposes, as shown in the next example (necessarily different between UNIX and Windows).

UNIX:

```
qstat -a 'qselect -u barry -l ncpus.gt.16'
```

Job ID	User	Queue	Jobname	Sess	NDS	TSK	Mem	Req'd Time	Req'd S	Elap Time
121.south	barry	workq	airfoil	--	--	32	--	0:01 H	--	--
133.south	barry	workq	trialx	--	--	20	--	0:01 W	--	--
154.south	barry	workq	airfoil	930	--	32	--	1:30 R	0:32	--

Windows (type the following at the cmd prompt, all on one line):

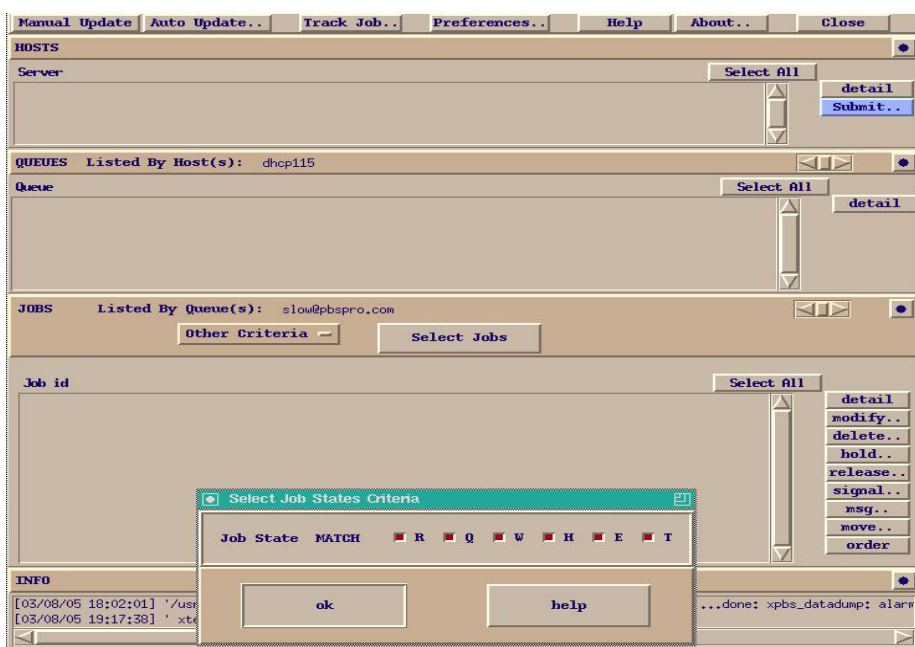
```
for /F "usebackq" %j in (`qselect -u barry -l ncpus.gt.16`) do
( qstat -a %j )
121.south
133.south
154.south
```

Note: This technique of using the output of the `qselect` command as input to `qstat` can also be used to supply input to other PBS commands as well.

7.4 Selecting Jobs Using `xpbs`

The `xpbs` command provides a graphical means of specifying job selection criteria, offering the flexibility of the `qselect` command in a point and click interface. Above the JOBS panel in the main `xpbs` display is the *Other Criteria* button. Clicking it will bring up a menu that lets you choose and select any job selection criteria you wish.

The example below shows a user clicking on the *Other Criteria* button, then selecting *Job States*, to reveal that all job states are currently selected. Clicking on any of these job states would remove that state from the selection criteria.



You may specify as many or as few selection criteria as you wish. When you have completed your selection, click on the *Select Jobs* button above the HOSTS panel to have *xpbs* refresh the display with the jobs that match your selection criteria. The selected criteria will remain in effect until you change them again. If you exit *xpbs*, you will be prompted if you wish to save your configuration information; this includes the job selection criteria.

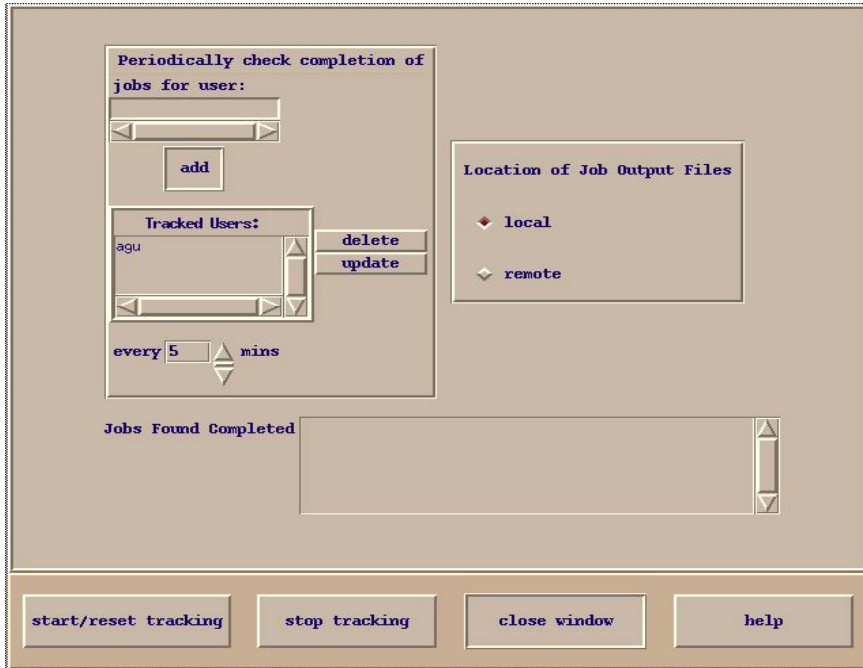
7.5 Using *xpbs* TrackJob Feature

The *xpbs* command includes a feature that allows you to track the progress of your jobs. When you enable the *Track Job* feature, *xpbs* will monitor your jobs, looking for the output files that signal completion of the job. The *Track Job* button will flash red on the *xpbs* main display, and if you then click it, *xpbs* will display a list of all completed jobs (that you were previously tracking). Selecting one of those jobs will launch a window containing the standard output and standard error files associated with the job.

IMPORTANT:

The Track Job feature is not currently available on Windows.

To enable xpbs job tracking, click on the *Track Job* button at the top center of the main xpbs display. Doing so will bring up the Track Job dialog box shown below.



From this window you can name the users whose jobs you wish to monitor. You also need to specify where you expect the output files to be: either local or remote (e.g. will the files be retained on the Server host, or did you request them to be delivered to another host?). Next, click the *start/reset tracking* button and then the *close window* button. Note that you can disable job tracking at any time by clicking the *Track Job* button on the main xpbs display, and then clicking the *stop tracking* button.

7.6 Job Comments for Problem Jobs

PBS can detect when a job cannot run with the current unused resources and when a job will never be able to run with all of the configured resources. PBS can set the job's comment attribute to reflect why the job is not running.

If the job's comment starts with "Can never run", the job will never be able to run with the resources that are currently configured. This can happen when:

- A job requests more of a consumable resource than is available on the entire complex
- A job requests a non-consumable resource that is not available on the complex

For example, if there are 128 total CPUs in the complex, and the job requests 256 CPUs, the job's comment will start with this message.

If the job's comment starts with "Not running", the job cannot run with the resources that are currently available. For example, if a job requests 8 CPUs and the complex has 16 CPUs but 12 are in use, the job's comment will start with this message.

You may see the following comments:

```
"Not enough free nodes available"
"Not enough total nodes available"
"Job will never run with the resources currently configured in the complex"
"Insufficient amount of server resource <resource> (Requested | Available |
  Total | <requested value> !=<available values for requested resource>)"
"Insufficient amount of queue resource <resource> (Requested | Available |
  Total | <requested value> !=<available values for requested resource>)"
"Error in calculation of start time of top job"
"Can't find start time estimate"
```

The "Can Never Run" prefix may be seen with the following messages:

```
"Insufficient amount of resource <resource> (Requested | Available | Total
  | <requested value> !=<available values for requested resource>)"
"Insufficient amount of Server resource <resource> (Requested | Available |
  Total | <requested value> !=<available values for requested resource>)"
"Insufficient amount of Queue resource <resource> (Requested | Available |
  Total | <requested value> !=<available values for requested resource>)"
"Not enough total nodes available"
"can't fit in the largest placement set, and can't span psets"
```


Chapter 8

Advanced PBS Features

This chapter covers the less commonly used commands and more complex topics which will add substantial functionality to your use of PBS. The reader is advised to read chapters 5 - 7 of this manual first.

8.1 UNIX Job Exit Status

On UNIX systems, the exit status of a job is normally the exit status of the shell executing the job script. If a user is using `csh` and has a `.logout` file in the home directory, the exit status of `csh` becomes the exit status of the last command in `.logout`. This may impact the use of job dependencies which depend on the job's exit status. To preserve the job's exit status, the user may either remove `.logout` or edit it as shown in this example:

```
set EXITVAL = $status
[ .logout's original content ]
```

Doing so will ensure that the exit status of the job persists across the invocation of the `.logout` file.

For a description of what job exit codes mean, see section 12.10, "Job Exit Codes", on page 829 of the PBS Professional Administrator's Guide.

The exit status of a job array is determined by the status of each of the completed subjobs. It is only available when all valid subjobs have completed. The individual exit status of a completed subjob is passed to the epilogue, and is available in the 'E' accounting log record of that subjob. See ["Job Array Exit Status" on page 253](#).

8.2 Changing UNIX Job umask

The “-W umask=*nnn*” option to `qsub` allows you to specify, on UNIX systems, what umask PBS should use when creating and/or copying your `stdout` and `stderr` files, and any other files you direct PBS to transfer on your behalf.

IMPORTANT:

This feature does not apply to Windows.

The following example illustrates how to set your umask to 022 (i.e. to have files created with write permission for owner only: `-rw-r--r--`).

```
qsub -W umask=022 my_job
#PBS -W umask=022
```

8.3 Requesting qsub Wait for Job Completion

The “-W block=true” option to `qsub` allows you to specify that you want `qsub` to wait for the job to complete (i.e. “block”) and report the exit value of the job. If job submission fails, no special processing will take place. If the job is successfully submitted, `qsub` will block until the job terminates or an error occurs.

If `qsub` receives one of the signals: `SIGHUP`, `SIGINT`, or `SIGTERM`, it will print a message and then exit with the exit status 2. If the job is deleted before running to completion, or an internal PBS error occurs, an error message describing the situation will be printed to this error stream and `qsub` will exit with an exit status of 3. Signals `SIGQUIT` and `SIGKILL` are not trapped and thus will immediately terminate the `qsub` process, leaving the associated job either running or queued. If the job runs to completion, `qsub` will exit with the exit status of the job. (See also [section 8.1, “UNIX Job Exit Status”, on page 193](#) for further discussion of the job exit status.)

For job arrays, blocking `qsub` waits until the entire job array is complete, then returns the exit status of the job array.

8.4 Specifying Job Dependencies

PBS allows you to specify dependencies between two or more jobs. Dependencies are useful for a variety of tasks, such as:

1. Specifying the order in which jobs in a set should execute
2. Requesting a job run only if an error occurs in another job
3. Holding jobs until a particular job starts or completes execution

The “-w depend=dependency_list” option to `qsub` defines the dependency between multiple jobs. The *dependency_list* has the format:

type:arg_list[,type:arg_list ...]

where except for the `on` type, the *arg_list* is one or more PBS job IDs in the form:

jobid[:jobid ...]

There are several types:

after:arg_list

This job may be scheduled for execution at any point after all jobs in *arg_list* have started execution.

afterok:arg_list

This job may be scheduled for execution only after all jobs in *arg_list* have terminated with no errors. See "Warning about exit status with `csh`" in EXIT STATUS.

afternotok:arg_list

This job may be scheduled for execution only after all jobs in *arg_list* have terminated with errors. See "Warning about exit status with `csh`" in EXIT STATUS.

afterany:arg_list

This job may be scheduled for execution after all jobs in *arg_list* have finished execution, with or without errors.

before:arg_list

Jobs in *arg_list* may begin execution once this job has begun execution.

beforeok:arg_list

Jobs in *arg_list* may begin execution once this job terminates without errors. See "Warning about exit status with `csh`" in EXIT STATUS.

beforenotok:arg_list

If this job terminates execution with errors, the jobs in *arg_list* may begin. See "Warning about exit status with `csh`" in EXIT STATUS.

beforeany:arg_list

Jobs in `arg_list` may begin execution once this job terminates execution, with or without errors.

on:count

This job may be scheduled for execution after `count` dependencies on other jobs have been satisfied. This type is used in conjunction with one of the `before` types listed. `count` is an integer greater than 0.

Job IDs in the `arg_list` of `before` types must have been submitted with a type of `on`.

To use the `before` types, the user must have the authority to alter the jobs in `arg_list`. Otherwise, the dependency is rejected and the new job aborted.

Error processing of the existence, state, or condition of the job on which the newly submitted job is a deferred service, i.e. the check is performed after the job is queued. If an error is detected, the new job will be deleted by the server. Mail will be sent to the job submitter stating the error.

Suppose you have three jobs (job1, job2, and job3) and you want job3 to start *after* job1 and job2 have *ended*. The first example below illustrates the options you would use on the `qsub` command line to specify these job dependencies.

```
qsub job1
16394.jupiter
qsub job2
16395.jupiter
qsub -W depend=afterany:16394:16395 job3
16396.jupiter
```

As another example, suppose instead you want job2 to start *only if* job1 ends with no errors (i.e. it exits with a no error status):

```
qsub job1
16397.jupiter
qsub -W depend=afterok:16397 job2
16396.jupiter
```

Similarly, you can use `before` dependencies, as the following example exhibits. Note that unlike `after` dependencies, `before` dependencies require the use of the `on` dependency.

```
qsub -W depend=on:2 job1
16397.jupiter
qsub -W depend=beforeany:16397 job2
16398.jupiter
qsub -W depend=beforeany:16397 job3
16399.jupiter
```

You can use `xpbs` to specify job dependencies as well. On the *Submit Job* window, in the other options section (far left, center of window) click on one of the three dependency buttons: “after depend”, “before depend”, or “concurrency”. These will launch a “Dependency” window in which you will be able to set up the dependencies you wish.

8.4.1 Job Array Dependencies

Job dependencies are supported:

- Between jobs and jobs
- Between job arrays and job arrays
- Between job arrays and jobs
- Between jobs and job arrays

Note: Job dependencies are not supported for subjobs or ranges of subjobs.

8.4.2 Caveats and Advice for Job Dependencies

8.4.2.1 Warning About Exit Status

The exit status of the job can affect how the job’s dependencies work. If your `.logout` runs a command that sets the job’s exit status, you may need to change it. See [section 8.1, “UNIX Job Exit Status”, on page 193](#).

8.4.2.2 Warning About Job History

Enabling job history changes the behavior of dependent jobs. If a job `j1` depends on a finished job `j2` for which PBS is maintaining history than `j1` will go into the held state. If job `j1` depends on a finished job `j3` that has been purged from the historical records than `j1` will be rejected just as in previous versions of PBS where the job was no longer in the system.

8.5 Delivery of Output Files

To transfer output files or to transfer staged-in or staged-out files to/from a remote destination, PBS uses either `rcp` or `scp` depending on the configuration options. The version of `rcp` used by PBS always exits with a non-zero exit status for any error. Thus MOM knows if the file was delivered or not. The secure copy program, `scp`, is also based on this version of `rcp` and exits with the proper exit status.

If using `rcp`, the copy of output or staged files can fail for (at least) two reasons.

- The user lacks authorization to access the specified system. (See discussion in ["User's PBS Environment" on page 12.](#))
- Under UNIX, if the user's `.cshrc` outputs any characters to standard output, e.g. contains an echo command, the copy will fail.

If using *Secure Copy* (`scp`), then PBS will first try to deliver output or stagein/out files using `scp`. If `scp` fails, PBS will try again using `rcp` (assuming that `scp` might not exist on the remote host). If `rcp` also fails, the above cycle will be repeated after a delay, in case the problem is caused by a temporary network problem. All failures are logged in MOM's log, and an email containing the errors is sent to the job owner.

For delivery of output files on the local host, PBS uses the `cp` command (UNIX) or the `xcopy` command (Windows XP) or the `robocopy` command (Windows Vista). Local and remote delivery of output may fail for the following additional reasons:

- A directory in the specified destination path does not exist.
- A directory in the specified destination path is not searchable by the user.
- The target directory is not writable by the user.

8.6 Input/Output File Staging

File staging is a way to specify which files should be copied onto the execution host before the job starts, and which should be copied off the execution host when it finishes.

8.6.1 Staging and Execution Directory: User's Home vs. Job-specific

The job's staging and execution directory is the directory to which files are copied before the job runs, and from which output files are copied after the job has finished. This directory is either your home directory or a job-specific directory created by PBS just for this job. If you

use job-specific staging and execution directories, you don't need to have a home directory on each execution host, as long as those hosts are configured properly. In addition, each job gets its own staging and execution directory, so you can more easily avoid filename collisions.

This table lists the differences between using your home directory for staging and execution and using a job-specific staging and execution directory created by PBS.

Table 8-1: Differences Between User's Home and Job-specific Directory for Staging and Execution

Question Regarding Action, Requirement, or Setting	User's Home Directory	Job-specific Directory
Does PBS create a job-specific staging and execution directory?	No	Yes
User's home directory must exist on execution host(s)?	Yes	No
Standard out and standard error automatically deleted when qsub -k option is used?	No	Yes
When are staged-out files are deleted?	Successfully staged-out files are deleted; others go to "undelivered"	Only after all are successfully staged out
Staging and execution directory deleted after job finishes?	No	Yes
How is job's sandbox attribute set?	HOME or not set	PRIVATE

8.6.2 Using Job-specific Staging and Execution Directories

8.6.2.1 Setting the Job's Staging and Execution Directory

The job's `sandbox` attribute controls whether PBS creates a unique job-specific staging and execution directory for this job. If the job's `sandbox` attribute is set to `PRIVATE`, PBS creates a unique staging and execution directory for the job. If `sandbox` is unset, or is set to `HOME`, PBS uses the user's home directory as the job's staging and execution directory. By default, the `sandbox` attribute is not set.

The user can set the `sandbox` attribute via `qsub`, or through a PBS directive. For example:

```
qsub -Wsandbox=PRIVATE
```

The job's `sandbox` attribute cannot be altered while the job is executing.

Table 8-2: Effect of Job's `sandbox` Attribute on Location of Staging and Execution Directory

Job's <code>sandbox</code> attribute	Effect
not set	Job's staging and execution directory is the user's home directory
HOME	Job's staging and execution directory is the user's home directory
PRIVATE	Job's staging and execution directory is a job-specific directory created by PBS. If the <code>qsub -k</code> option is used, output and error files are retained on the primary execution host in the staging and execution directory. This directory is removed, along with all of its contents, when the job finishes.

8.6.2.2 The Job's `jobdir` Attribute and the `PBS_JOBDIR` Environment Variable

The job's `jobdir` attribute is a read-only attribute, set to the pathname of the job's staging and execution directory on the primary host. The user can view this attribute by using `qstat -f`, only while the job is executing. The value of `jobdir` is not retained if a job is rerun; it is undefined whether `jobdir` is visible or not when the job is not executing.

The environment variable `PBS_JOBDIR` is set to the pathname of the staging and execution directory on the primary execution host. `PBS_JOBDIR` is added to the job script process, any job tasks, and the prologue and epilogue.

8.6.3 Attributes and Environment Variables Affecting Staging

The following attributes and environment variables affect staging and execution.

Table 8-3: Attributes and Environment Variables Affecting Staging

Job's Attribute or Environment Variable	Effect
<code>sandbox</code> attribute	Determines whether PBS uses user's home directory or creates job-specific directory for staging and execution. User-settable per job via <code>qsub -W</code> or through a PBS directive.
<code>stagein</code> attribute	Sets list of files or directories to be staged in. User-settable per job via <code>qsub -W</code> or through a PBS directive.
<code>stageout</code> attribute	Sets list of files or directories to be staged out. User-settable per job via <code>qsub -W</code> or through a PBS directive.
<code>Keep_Files</code> attribute	Determines whether output and/or error files remain on execution host. User-settable per job via <code>qsub -k</code> or through a PBS directive. If the <code>Keep_Files</code> attribute is set to <code>o</code> and/or <code>e</code> (output and/or error files remain in the staging and execution directory) and the job's <code>sandbox</code> attribute is set to <code>PRIVATE</code> , standard out and/or error files are removed, when the staging directory is removed at job end along with its contents.
<code>jobdir</code> attribute	Set to pathname of staging and execution directory on primary execution host. Read-only; viewable via <code>qstat -f</code> .
<code>PBS_JOBDIR</code> environment variable	Set to pathname of staging and execution directory on primary execution host. Added to environments of job script process, job tasks, and prologue and epilogue.
<code>TMPDIR</code> environment variable	Location of job-specific scratch directory.

8.6.4 Specifying Files To Be Staged In or Staged Out

You can specify files to be staged in before the job runs and staged out after the job runs by using `-W stagein=file_list` and `-W stageout=file_list`. You can use these as options to `qsub`, or as directives in the job script.

The *file_list* takes the form:

```
execution_path@hostname:storage_path[,...]
```

for both stagein and stageout.

The name *execution_path* is the name of the file in the job's staging and execution directory (on the execution host). The *execution_path* can be relative to the job's staging and execution directory, or it can be an absolute path.

The '@' character separates the execution specification from the storage specification.

The name *storage_path* is the file name on the host specified by *hostname*. For stagein, this is the location where the input files come from. For stageout, this is where the output files end up when the job is done. You must specify a hostname. The name can be absolute, or it can be relative to the user's home directory on the machine named *hostname*.

IMPORTANT:

It is advisable to use an absolute pathname for the *storage_path*. Remember that the path to your home directory may be different on each machine, and that when using `sandbox = PRIVATE`, you may or may not have a home directory on all execution machines.

For stagein, the direction of travel is **from** *storage_path* **to** *execution_path*.

For stageout, the direction of travel is **from** *execution_path* **to** *storage_path*.

The following example shows how to use a directive to stagein a file named `grid.dat` located in the directory `/u/user1` on the host called `serverA`. The staged-in file is copied to the staging and execution directory and given the name `dat1`. Since *execution_path* is evaluated relative to the staging and execution directory, it is not necessary to specify a full pathname for `dat1`. Always use a relative pathname for *execution_path* when the job's staging and execution directory is created by PBS.

```
#PBS -W stagein=dat1@serverA:/u/user1/grid.dat ...
```

To use the `qsub` option to stage in the file residing on `myhost`, in `/Users/myhome/mydata/data1`, calling it `input_data1` in the staging and execution directory:

```
qsub -W stagein=input_data1@myhost:/Users/myhome/mydata/data1
```

To stage more than one file or directory, use a comma-separated list of paths, and enclose the list in double quotes. For example, to stage two files `data1` and `data2` in:

```
qsub -W stagein="input1@hostA:/myhome/data1, \input2@hostA:/myhome/data1"
```

- Under Windows, special characters such as spaces, backslashes (`\`), colons (`:`), and drive letter specifications are valid pathnames. For example, the following will stage in the `grid.dat` file on drive D at `hostB` to a local file (`"dat1"`) on drive C.:

```
qsub -W stagein="dat1@hostB:D:\Documents and Settings\grid.dat"
```

8.6.4.1 Copying Directories Into and Out Of the Staging and Execution Directory

You can stage directories into and out of the staging and execution directory the same way you stage files. The `storage_path` and `execution_path` for both `stagein` and `stageout` can be a directory. If you `stagein` or `stageout` a directory, PBS copies that directory along with all of its files and subdirectories. At the end of the job, the directory, including all files and subdirectories, is deleted. This can create a problem if multiple jobs are using the same directory.

8.6.4.2 Wildcards In File Staging

You can use wildcards when staging files and directories, according to the following rules.

- The asterisk `"*"` matches one or more characters.
- The question mark `"?"` matches a single character.
- All other characters match only themselves.
- Wildcards inside of quote marks are expanded.
- Wildcards cannot be used to match UNIX files that begin with period `"."` or Windows files that have the `"SYSTEM"` or `"HIDDEN"` attributes.
- When using the `qsub` command line on UNIX, you must prevent the shell from expanding wildcards. For some shells, you can enclose the pathnames in double quotes. For some shells, you can use a backspace before the wildcard.
- Wildcards can only be used in the source side of a staging specification. This means they can be used in the `storage_path` specification for `stagein`, and in the `execution_path` specification for `stageout`.
- When staging using wildcards, the destination must be a directory. If the destination is not a directory, the result is undefined. So for example, when staging out all `.out` files, you must specify a directory for `storage_path`.
- Wildcards can only be used in the final path component, i.e. the basename.

- When wildcards are used during stagein, PBS will not automatically delete staged files at job end. Note that if PBS created the staging and execution directory, that directory and all its contents are deleted at job end.

8.6.4.3 Examples of File Staging

Example 8-1: Stage out all files from the execution directory to a specific directory:

UNIX

```
-W stageout=*@myworkstation:/user/project1/case1
```

Windows

```
-W stageout=*@mypc:E:\project1\case1
```

Example 8-2: Stage out specific types of result files and disregard the scratch and other temporary files after the job terminates. The result files that are interesting for this example end in '.dat':

UNIX

```
-W stageout=*.dat@myworkstation:project3/data
```

Windows

```
-W stageout=*.dat@mypc:C:\project\data
```

Example 8-3: Stage in all files from an application data directory to a subdirectory:

UNIX

```
-W stagein=jobarea@myworkstation:crashtest1/*
```

Windows

```
-W stagein=jobarea@mypc:E:\crashtest1\*
```

Example 8-4: Stage in data from files and directories matching “wing*”:

UNIX

```
-W stagein=.*@myworkstation:848/wing*
```

Windows

```
-W stagein=.*@mypc:E:\flowcalc\wing*
```

Example 8-5: Stage in .bat and .dat files to jobarea:

UNIX:

```
-W stagein=jobarea@myworkstation:/users/me/crash1.?at
```

Windows:

```
-W stagein=jobarea@myworkstation:C:\me\crash1.?at
```

8.6.4.4 Caveats

When using a job-specific staging and execution directory, do not use an absolute path in `execution_path`.

8.6.4.5 Output Filenames

The name of the job defaults to the script name, if no name is given via `qsub -N`, via a PBS directive, or via `stdin`. For example, if the sequence number is 1234,

```
#PBS -N fixgamma
```

gives `stdout` the name `fixgamma.o1234` and `stderr` the name `fixgamma.e1234`.

For information on submitting jobs, see [section 3.4, “Submitting a PBS Job”, on page 29](#).

8.6.5 Example of Using Job-specific Staging and Execution Directories

In this example, you want the file “`jay.fem`” to be delivered to the job-specific staging and execution directory given in `PBS_JOBDIR`, by being copied from the host “`submithost`”. The job script is executed in `PBS_JOBDIR` and “`jay.out`” is staged out from `PBS_JOBDIR` to your home directory on the submittal host (i.e., “`hostname`”):

```
qsub -Wsandbox=PRIVATE -Wstagein=jay.fem@submit- host:jay.fem -Wstage-  
out=jay.out@submithost:jay.out
```

8.6.6 Summary of the Job’s Lifecycle

This is a summary of the steps performed by PBS. The steps are not necessarily performed in this order.

- On each execution host, if specified, PBS creates a job-specific staging and execution directory.
- PBS sets `PBS_JOBDIR` and the job’s `jobdir` attribute to the path of the job’s staging and execution directory.
- On each execution host allocated to the job, PBS creates a job-specific temporary directory.
- PBS sets the `TMPDIR` environment variable to the pathname of the temporary directory.
- If any errors occur during directory creation or the setting of variables, the job is

requeued.

- PBS stages in any files or directories.
- The prologue is run on the primary execution host, with its current working directory set to `PBS_HOME/mom_priv`, and with `PBS_JOBDIR` and `TMPDIR` set in its environment.
- The job is run as the user on the primary execution host.
- The job's associated tasks are run as the user on the execution host(s).
- The epilogue is run on the primary execution host, with its current working directory set to the path of the job's staging and execution directory, and with `PBS_JOBDIR` and `TMPDIR` set in its environment.
- PBS stages out any files or directories.
- PBS removes any staged files or directories.
- PBS removes any job-specific staging and execution directories and their contents, and all `TMPDIRs` and their contents.
- PBS writes the final job accounting record and purges any job information from the Server's database.

8.6.7 Detailed Description of Job's Lifecycle

8.6.7.1 Creation of `TMPDIR`

For each host allocated to the job, PBS creates a job-specific temporary scratch directory for the job. If the temporary scratch directory cannot be created, the job is aborted.

8.6.7.2 Choice of Staging and Execution Directories

If the job's `sandbox` attribute is set to `PRIVATE`, PBS creates job-specific staging and execution directories for the job. If the job's `sandbox` attribute is set to `HOME`, or is unset, PBS uses the user's home directory for staging and execution.

8.6.7.2.i Job-specific Staging and Execution Directories

If the staging and execution directory cannot be created the job is aborted. If PBS fails to create a staging and execution directory, see the system administrator.

You should not depend on any particular naming scheme for the new directories that PBS creates for staging and execution.

8.6.7.2.ii User's Home Directory as Staging and Execution Directory

The user must have a home directory on each execution host. The absence of the user's home directory is an error and causes the job to be aborted.

8.6.7.3 Setting Environment Variables and Attributes

PBS sets `PBS_JOBDIR` and the job's `jobdir` attribute to the pathname of the staging and execution directory. The `TMPDIR` environment variable is set to the pathname of the job-specific temporary scratch directory.

8.6.7.4 Staging Files Into Staging and Execution Directories

PBS evaluates `execution_path` and `storage_path` relative to the staging and execution directory given in `PBS_JOBDIR`, whether this directory is the user's home directory or a job-specific directory created by PBS. PBS copies the specified files and/or directories to the job's staging and execution directory.

8.6.7.5 Running the Prologue

The MOM's prologue is run on the primary host as root, with the current working directory set to `PBS_HOME/mom_priv`, and with `PBS_JOBDIR` and `TMPDIR` set in its environment.

8.6.7.6 Job Execution

PBS runs the job script on the primary host as the user. PBS also runs any tasks created by the job as the user. The job script and tasks are executed with their current working directory set to the job's staging and execution directory, and with `PBS_JOBDIR` and `TMPDIR` set in their environment.

8.6.7.7 Standard Out and Standard Error

The job's `stdout` and `stderr` files are created directly in the job's staging and execution directory on the primary execution host.

8.6.7.7.i Job-specific Staging and Execution Directories

If the `qsub -k` option is used, the `stdout` and `stderr` files will **not** be automatically copied out of the staging and execution directory at job end - they will be deleted when the directory is automatically removed.

8.6.7.7.ii User's Home Directory as Staging and Execution Directory

If the `-k` option to `qsub` is used, standard out and/or standard error files are retained on the primary execution host instead of being returned to the submission host, and are not deleted after job end.

8.6.7.8 Running the Epilogue

PBS runs the epilogue on the primary host as root. The epilogue is executed with its current working directory set to the job's staging and execution directory, and with `PBS_JOBDIR` and `TMPDIR` set in its environment.

8.6.7.9 Staging Files Out and Removing Execution Directory

When PBS stages files out, it evaluates `execution_path` and `storage_path` relative to `PBS_JOBDIR`. Files that cannot be staged out are saved in `PBS_HOME/undelivered`. See [section 14.6.6, "Non-delivery of Output" on page 885 in the PBS Professional Administrator's Guide](#).

8.6.7.9.i Job-specific Staging and Execution Directories

If PBS created job-specific staging and execution directories for the job, it cleans up at the end of the job. The staging and execution directory and all of its contents are removed, on all execution hosts.

8.6.7.10 Removing TMPDIRs

PBS removes all TMPDIRs, along with their contents.

8.6.8 Staging with Job Arrays

File staging is supported for job arrays. See ["File Staging" on page 239](#).

8.6.9 Using xpbs for File Staging

Using `xpbs` to set up file staging directives may be easier than using the command line. On the *Submit Job* window, in the miscellany options section (far left, center of window) click on the *file staging* button. This will launch the *File Staging* dialog box (shown below) in which you will be able to set up the file staging you desire.

The *File Selection Box* will be initialized with your current working directory. If you wish to select a different directory, double-click on its name, and `xpbs` will list the contents of the new directory in the *File Selection Box*. When the correct directory is displayed, simply click on the name of the file you wish to stage (in or out). Its name will be written in the *File Selected* area.

Next, click either of the *Add file selected...* buttons to add the named file to the stagein or stageout list. Doing so will write the file name into the corresponding area on the lower half of the *File Staging* window. Now you need to provide location information. For stagein, type in the path and filename where you want the named file placed. For stageout, specify the host-name and pathname where you want the named file delivered. You may repeat this process for as many files as you need to stage.

When you are done selecting files, click the *OK* button.

8.6.10 Stagein and Stageout Failure

When stagein fails, the job is placed in a 30-minute wait to allow the user time to fix the problem. Typically this is a missing file or a network outage. Email is sent to the job owner when the problem is detected. Once the problem has been resolved, the job owner or the Operator may remove the wait by resetting the time after which the job is eligible to be run via the `-a` option to `qalter`. The server will update the job's comment with information about why the job was put in the wait state. When the job is eligible to run, it may run on different vnodes.

When stageout encounters an error, there are three retries. PBS waits 1 second and tries again, then waits 11 seconds and tries a third time, then finally waits another 21 seconds and tries a fourth time. Email is sent to the job owner if all attempts fail. Files that cannot be staged out are saved in `PBS_HOME/undelivered`. See [section 14.6.6, "Non-delivery of Output" on page 885 in the PBS Professional Administrator's Guide](#).

8.7 Advance and Standing Reservation of Resources

8.7.1 Definitions

Advance reservation

A reservation for a set of resources for a specified time. The reservation is only available to a specific user or group of users.

Standing reservation

An advance reservation which recurs at specified times. For example, the user can reserve 8 CPUs and 10GB every Wednesday and Thursday from 5pm to 8pm, for the next three months.

Occurrence of a standing reservation

An instance of the standing reservation.

An occurrence of a standing reservation behaves like an advance reservation, with the following exceptions:

- while a job can be submitted to a specific advance reservation, it can only be submitted to the standing reservation as a whole, not to a specific occurrence. You can only specify *when* the job is eligible to run. See the `qsub(1B)` man page.
- when an advance reservation ends, it and all of its jobs, running or queued, are deleted, but when an occurrence ends, only its running jobs are deleted.

Each occurrence of a standing reservation has reserved resources which satisfy the resource request, but each occurrence may have its resources drawn from a different source. A query for the resources assigned to a standing reservation will return the resources assigned to the soonest occurrence, shown in the `resv_nodes` attribute reported by `pbs_rstat`.

Soonest occurrence of a standing reservation

The occurrence which is currently active, or if none is active, then it is the next occurrence.

Degraded reservation

An advance reservation for which one or more associated vnodes are unavailable.

A standing reservation for which one or more vnodes associated with any occurrence are unavailable.

8.7.2 Introduction to Creating and Using Reservations

The user creates both advance and standing reservations using the `pbs_rsub` command. PBS either confirms that the reservation can be made, or rejects the request. Once the reservation is confirmed, PBS creates a queue for the reservation's jobs. Jobs are then submitted to this queue.

When a reservation is confirmed, it means that the reservation will not conflict with currently running jobs, other confirmed reservations, or dedicated time, and that the requested resources are available for the reservation. A reservation request that fails these tests is rejected. All occurrences of a standing reservation must be acceptable in order for the standing reservation to be confirmed.

The `pbs_rsub` command returns a *reservation ID*, which is the reservation name. For an advance reservation, this reservation ID has the format:

R<unique integer>.<server name>

For a standing reservation, this reservation ID refers to the entire series, and has the format:

S<unique integer>.<server name>

The user specifies the resources for a reservation using the same syntax as for a job. Jobs in reservations are placed the same way non-reservation jobs are placed in placement sets.

The `xpbs` GUI cannot be used for creation, querying, or deletion of reservations.

The time for which a reservation is requested is in the time zone at the submission host.

8.7.3 Creating Advance Reservations

You create an advance reservation using the `pbs_rsub` command. PBS must be able to calculate the start and end times of the reservation, so you must specify two of the following three options:

D	Duration
E	End time
R	Start time

8.7.3.1 Examples of Creating Advance Reservations

The following example shows the creation of an advance reservation asking for 1 vnode, 30 minutes of wall-clock time, and a start time of 11:30. Since an end time is not specified, PBS will calculate the end time based on the reservation start time and duration.

```
pbs_rsub -R 1130 -D 00:30:00
```

PBS returns the reservation ID:

```
R226.south UNCONFIRMED
```

The following example shows an advance reservation for 2 CPUs from 8:00 p.m. to 10:00 p.m.:

```
pbs_rsub -R 2000.00 -E 2200.00 -l select=1:ncpus=2
```

PBS returns the reservation ID:

```
R332.south UNCONFIRMED
```

8.7.4 Creating Standing Reservations

You create standing reservations using the `pbs_rsub` command. You **must** specify a start and end date when creating a standing reservation. The recurring nature of the reservation is specified using the `-r` option to `pbs_rsub`. The `-r` option takes the `recurrence_rule` argument, which specifies the standing reservation's occurrences. The recurrence rule uses iCalendar syntax, and uses a subset of the parameters described in RFC 2445.

The recurrence rule can take two forms:

```
"FREQ= freq_spec; COUNT= count_spec; interval_spec"
```

In this form, you specify how often there will be occurrences, how many there will be, and which days and/or hours apply.

```
"FREQ= freq_spec; UNTIL= until_spec; interval_spec"
```

In this form, the user specifies how often there will be occurrences, when the occurrences will end, and which days and/or hours apply.

freq_spec

This is the frequency with which the reservation repeats. Valid values are WEEKLY | DAILY | HOURLY

When using a `freq_spec` of WEEKLY, you may use an `interval_spec` of BYDAY and/or BYHOUR. When using a `freq_spec` of DAILY, you may use an `interval_spec` of BYHOUR. When using a `freq_spec` of HOURLY, do not use an `interval_spec`.

count_spec

The exact number of occurrences. Number up to 4 digits in length. Format: integer.

interval_spec

Specifies the interval at which there will be occurrences. Can be one or both of BYDAY=<days> or BYHOUR=<hours>. Valid values are BYDAY = MO | TU | WE | TH | FR | SA | SU and BYHOUR = 0 | 1 | 2 | . . . | 23.

When using both, separate them with a semicolon. Separate days or hours with a comma.

For example, to specify that there will be recurrences on Tuesdays and Wednesdays, at 9 a.m. and 11 a.m., use BYDAY=TU,WE;BYHOUR=9,11

BYDAY should be used with FREQ=WEEKLY. BYHOUR should be used with FREQ=DAILY or FREQ=WEEKLY.

until_spec

Occurrences will start up to but not after this date and time. This means that if occurrences last for an hour, and normally start at 9 a.m., then a time of 9:05 a.m on the day specified in the `until_spec` means that an occurrence will start on that day.

Format: `YYYYMMDD[THHMMSS]`

Note that the year-month-day section is separated from the hour-minute-second section by a capital `T`.

Default: 3 years from time of reservation creation.

8.7.4.1 Setting Reservation Start Time and Duration

In a standing reservation, the arguments to the `-R` and `-E` options to `pbs_rsub` can provide more information than they do in an advance reservation. In an advance reservation, they provide the start and end time of the reservation. In a standing reservation, they can provide the start and end time, but they can also be used to compute the duration and the offset from the interval start.

The difference between the values of the arguments for `-R` and `-E` is the duration of the reservation. For example, if you specify

```
-R 0930 -E 1145
```

the duration of your reservation will be two hours and fifteen minutes. If you specify

```
-R 150800 -E 170830
```

the duration of your reservation will be two days plus 30 minutes.

The `interval_spec` can be used to specify the day or the hour at which the interval starts. If you specify

```
-R 0915 -E 0945 ... BYHOUR=9,10
```

the duration is 30 minutes, and the offset is 15 minutes from the start of the interval. The interval start is at 9 and again at 10. Your reservation will run from 9:15 to 9:45, and again at 10:15 and 10:45. Similarly, if you specify

```
-R 0800 -E -1000 ... BYDAY=WE,TH
```

the duration is two hours and the offset is 8 hours from the start of the interval. Your reservation will run Wednesday from 8 to 10, and again on Thursday from 8 to 10.

Elements specified in the recurrence rule override those specified in the arguments to the `-R` and `-E` options. Therefore if you specify

```
-R 0730 -E 0830 ... BYHOUR=9
```

the duration is one hour, but the hour element (9:00) in the recurrence rule has overridden the hour element specified in the argument to `-R` (7:00). The offset is still 30 minutes after the interval start. Your reservation will run from 9:30 to 10:30. Similarly, if the 16th is a Monday, and you specify

```
-R 160800 -E 170900 ... BYDAY=TU;BYHOUR=11
```

the duration 25 hours, but both the day and the hour elements have been overridden. Your reservation will run on Tuesday at 11, for 25 hours, ending Wednesday at 12. However, if you specify

```
-R 160810 -E 170910 ... BYDAY=TU;BYHOUR=11
```

the duration is 25 hours, and the offset from the interval start is 10 minutes. Your reservation will run on Tuesday at 11:10, for 25 hours, ending Wednesday at 12:10. The minutes in the offset weren't overridden by anything in the recurrence rule.

The values specified for the arguments to the `-R` and `-E` options can be used to set the start and end times in a standing reservation, just as they are in an advance reservation. To do this, don't override their elements inside the recurrence rule. If you specify

```
-R 0930 -E 1030 ... BYDAY=MO,TU
```

you haven't overridden the hour or minute elements. Your reservation will run Monday and Tuesday, from 9:30 to 10:30.

8.7.4.2 Requirements for Creating Standing Reservations

- The user must specify a start and end date. See the `-R` and `-E` options to the `pbs_rsub` command in [section 8.7.5, “The pbs_rsub Command”, on page 215](#).
- The user must set the submission host's `PBS_TZID` environment variable. The format for `PBS_TZID` is a timezone location. Example: `America/Los_Angeles`, `America/Detroit`, `Europe/Berlin`, `Asia/Calcutta`. See [section 8.7.9.1, “Setting the Submission Host's Time Zone”, on page 223](#).
- The recurrence rule must be one unbroken line. See the `-r` option to `pbs_rsub` in [section 8.7.5, “The pbs_rsub Command”, on page 215](#).
- The recurrence rule must be enclosed in double quotes.
- Vnodes that have been configured to accept jobs only from a specific queue (vnode-queue restrictions) cannot be used for advance or standing reservations. See your PBS administrator to determine whether some vnodes have been configured to accept jobs only from specific queues.

8.7.4.3 Examples of Creating Standing Reservations

For a reservation that runs every day from 8am to 10am, for a total of 10 occurrences:

```
pbs_rsub -R 0800 -E 1000 -r "FREQ=DAILY;COUNT=10"
```

Every weekday from 6am to 6pm until December 10, 2008:

```
pbs_rsub -R 0600 -E 1800 -r "FREQ=WEEKLY; BYDAY=MO,TU,WE,TH,FR;  
UNTIL=20081210"
```

Every week from 3pm to 5pm on Monday, Wednesday, and Friday, for 9 occurrences, i.e., for three weeks:

```
pbs_rsub -R 1500 -E 1700 -r "FREQ=WEEKLY;BYDAY=MO,WE,FR; COUNT=9"
```

8.7.5 The **pbs_rsub** Command

The **pbs_rsub** command returns a reservation ID string, and the current status of the reservation.

For the options to the **pbs_rsub** command, see [“pbs_rsub” on page 79 of the PBS Professional Reference Guide](#).

8.7.5.1 Getting Confirmation of a Reservation

By default the **pbs_rsub** command does not immediately notify you whether the reservation is confirmed or denied. Instead you receive email with this information. You can specify that the **pbs_rsub** command should wait for confirmation by using the **-I <block_time>** option. The **pbs_rsub** command will wait up to **<block_time>** seconds for the reservation to be confirmed or denied and then notify you of the outcome. If **block_time** is negative and the reservation is not confirmed in that time, the reservation is automatically deleted.

To find out whether the reservation has been confirmed, use the **pbs_rstat** command. It will display the state of the reservation. **CO** and **RESV_CONFIRMED** indicate that it is confirmed. If the reservation does not appear in the output from **pbs_rstat**, that means that the reservation was denied.

To ensure that you receive mail about your reservations, set the reservation's **Mail_Users** attribute via the **-M <email address>** option to **pbs_rsub**. By default, you will get email when the reservation is terminated or confirmed. If you want to receive email about events other than those, set the reservation's **Mail_Points** attribute via the **-m <mail events>** option. For more information, see the **pbs_rsub(1B)** and **pbs_resv_attributes(7B)** man pages.

8.7.6 Viewing the Status of a Reservation

The following table shows the list of possible states for a reservation. The states that you will usually see are CO, UN, BD, and RN, although a reservation usually remains unconfirmed for too short a time to see that state. See [“Reservation States” on page 419 of the PBS Professional Reference Guide](#).

To view the status of a reservation, use the `pbs_rstat` command. It will display the status of all reservations at the PBS server. For a standing reservation, the `pbs_rstat` command will display the status of the soonest occurrence. Duration is shown in seconds. The `pbs_rstat` command will not display a custom resource which has been created to be invisible. See [section 3.5.15, “Resource Permissions”, on page 42](#). This command has three options:

Table 8-4: Options to `pbs_rstat` Command

Option	Meaning	Description
B	Brief	Lists only the names of the reservations
S	Short	Lists in table format the name, queue name, owner, state, and start, duration and end times of each reservation
F	Full	Lists the name and all non-default-value attributes for each reservation.
<none>	Default	Default is S option

The full listing for a standing reservation is identical to the listing for an advance reservation, with the following additions:

- A line that specifies the recurrence rule:
`reserve_rrule = FREQ=WEEKLY;BYDAY=MO;COUNT=5`
- An entry for the vnodes reserved for the soonest occurrence of the standing reservation. This entry also appears for an advance reservation, but will be different for each occur-

rence:

resv_nodes=(vnode_name:...)

- A line that specifies the total number of occurrences of the standing reservation:
reserve_count = 5
- The index of the soonest occurrence:
reserve_index = 1
- The timezone at the site of submission of the reservation is appended to the reservation Variable list. For example, in California:
Variable_List=<other variables>PBS_TZID=America/Los_Angeles

To get the status of a reservation at a server other than the default server, set the PBS_SERVER environment variable to the name of the server you wish to query, then use the pbs_rstat command. Your PBS commands will treat the new server as the default server, so you may wish to unset this environment variable when you are finished.

You can also get information about the reservation's queue by using the qstat command. See [“qstat” on page 194 of the PBS Professional Reference Guide](#) and the qstat (1B) man page.

8.7.6.1 Examples of Viewing Reservation Status Using pbs_rstat

In our example, we have one advance reservation and one standing reservation. The advance reservation is for today, for two hours, starting at noon. The standing reservation is for every Thursday, for one hour, starting at 3:00 p.m. Today is Monday, April 28th, and the time is 1:00, so the advance reservation is running, and the soonest occurrence of the standing reservation is Thursday, May 1, at 3:00 p.m.

Example brief output:

```
pbs_rstat -B
```

```
Name: R302.south
```

```
Name: S304.south
```

Example short output:

pbs_rstat -S

Name	Queue	User	State	Start	/	Duration	/	End
------	-------	------	-------	-------	---	----------	---	-----

R302.south	R302	user1	RN	Today 12:00	/	7200/		Today 14:00
------------	------	-------	----	-------------	---	-------	--	-------------

S304.south	S304	user1	CO	May 1 2008 15:00	/	3600/		May 1 2008 16:00
------------	------	-------	----	------------------	---	-------	--	------------------

Example full output:

pbs_rstat -F

```
Name: R302.south
Reserve_Name = NULL
Reserve_Owner = user1@south.mydomain.com
reserve_state = RESV_RUNNING
reserve_substate = 5
reserve_start = Mon Apr 28 12:00:00 2008
reserve_end = Mon Apr 28 14:00:00 2008
reserve_duration = 7200
queue = R302
Resource_List.ncpus = 2
Resource_List.nodect = 1
Resource_List.walltime = 02:00:00
Resource_List.select = 1:ncpus=2
Resource_List.place = free
resv_nodes = (south:ncpus=2)
Authorized_Users = user1@south.mydomain.com
server = south
ctime = Mon Apr 28 11:00:00 2008
Mail_Users = user1@mydomain.com
mtime = Mon Apr 28 11:00:00 2008
Variable_List = PBS_O_LOGNAME=user1,PBS_O_HOST=south.mydomain.com
```

```
Name: S304.south
Reserve_Name = NULL
Reserve_Owner = user1@south.mydomain.com
reserve_state = RESV_CONFIRMED
reserve_substate = 2
reserve_start = Thu May 1 15:00:00 2008
reserve_end = Thu May 1 16:00:00 2008
reserve_duration = 3600
queue = S304
Resource_List.ncpus = 2
Resource_List.nodect = 1
Resource_List.walltime = 01:00:00
Resource_List.select = 1:ncpus=2
```

```
Resource_List.place = free
resv_nodes = (south:ncpus=2)
reserve_rrule = FREQ=WEEKLY;BYDAY=MO;COUNT=5
reserve_count = 5
reserve_index = 2
Authorized_Users = user1@south.mydomain.com
server = south
ctime = Mon Apr 28 11:01:00 2008
Mail_Users = user1@mydomain.com
mtime = Mon Apr 28 11:01:00 2008
Variable_List =
    PBS_O_LOGNAME=user1,PBS_O_HOST=south.mydomain.com,PBS_TZID=America/
    Los_Angeles
```

8.7.7 Deleting Reservations

You can delete an advance or standing reservation by using the `pbs_rdel` command. For a standing reservation, you can only delete the entire reservation, including all occurrences. When you delete a reservation, all of the jobs that have been submitted to the reservation are also deleted. A reservation can be deleted by its owner or by a PBS Operator or Manager. For example, to delete `S304.south`:

```
pbs_rdel S304.south
```

or

```
pbs_rdel S304
```

8.7.8 Submitting a Job to a Reservation

Jobs can be submitted to the queue associated with a reservation, or they can be moved from another queue into the reservation queue. You submit a job to a reservation by using the `-q <queue>` option to the `qsub` command to specify the reservation queue. For example, to submit a job to the soonest occurrence of a standing reservation named `S123.south`, submit to its queue `S123`:

```
qsub -q S123 <script>
```

You move a job into a reservation queue by using the `qmove` command. For more information, see the `qsub(1B)` and `qmove(1B)` man pages. For example, to `qmove` job `22.myhost` from `workq` to `S123`, the queue for the reservation named `S123.south`:

```
qmove S123 22.myhost
```

or

qmove S123 22

A job submitted to a standing reservation without a restriction on when it can run will be run, if possible, during the soonest occurrence. In order to submit a job to a specific occurrence, use the `-a <start time>` option to the `qsub` command, setting the start time to the time of the occurrence that you want. You can also use a `cron` job to submit a job at a specific time. See the `qsub(1B)` and `cron(8)` man pages.

8.7.8.1 Running Jobs in a Reservation

A confirmed reservation will accept jobs into its queue at any time. Jobs are only scheduled to run from the reservation once the reservation period arrives.

The jobs in a reservation are not allowed to use, in aggregate, more resources than the reservation requested. A reservation job is started only if its requested walltime will fit within the reservation period. So for example if the reservation runs from 10:00 to 11:00, and the job's walltime is 4 hours, the job will not be started.

When an advance reservation ends, any running or queued jobs in that reservation are deleted.

When an occurrence of a standing reservation ends, any running jobs in that reservation are killed. Any jobs still queued for that reservation are kept in the queued state. They are allowed to run in future occurrences. When the last occurrence of a standing reservation ends, all jobs remaining in the reservation are deleted, whether queued or running.

A job in a reservation cannot be preempted.

8.7.8.1.i Reservation Fault Tolerance

If one or more vnodes allocated to an advance reservation or to the soonest occurrence of a standing reservation become unavailable, the reservation's state becomes *DG* or *RESV_DEGRADED*. A degraded reservation does not have all the reserved resources to run its jobs.

PBS attempts to reconfirm degraded reservations. This means that it looks for alternate available vnodes on which to run the reservation. The reservation's `retry_time` attribute lists the next time when PBS will try to reconfirm the reservation.

If PBS is able to reconfirm a degraded reservation, the reservation's state becomes *CO*, or *RESV_CONFIRMED*, and the reservation's `resv_nodes` attribute shows the new vnodes.

8.7.8.2 Access to Reservations

By default, the reservation accepts jobs only from the user who created the reservation, and accepts jobs submitted from any group or host. You can specify a list of users and groups whose jobs will and will not be accepted by the reservation by setting the reservation's `Authorized_Users` and `Authorized_Groups` attributes using the `-U auth_user_list` and `-G auth_group_list` options to `pbs_rsub`. You can specify the hosts from which jobs can and cannot be submitted by setting the reservation's `Authorized_Hosts` attribute using the `-H auth_host_list` option to `pbs_rsub`.

The administrator can also specify which users and groups can and cannot submit jobs to a reservation, and the list of hosts from which jobs can and cannot be submitted.

For more information, see the `pbs_rsub(1B)` and `pbs_resv_attributes(7B)` man pages.

8.7.8.3 Viewing Status of a Job Submitted to a Reservation

You can view the status of a job that has been submitted to a reservation or to an occurrence of a standing reservation by using the `qstat` command. See [“qstat” on page 194 of the PBS Professional Reference Guide](#) and the `qstat(1B)` man page.

For example, if a job named `MyJob` has been submitted to the soonest occurrence of the standing reservation named `S304.south`, it is listed under `S304`, the name of the queue:

qstat

Job id	Name	User	Time Use	S	Queue
-----	-----	-----	-----	--	-----
139.south	MyJob	user1	0	Q	S304

8.7.9 Reservation Caveats and Errors

8.7.9.1 Setting the Submission Host's Time Zone

The environment variable `PBS_TZID` must be set at the submission host. The time for which a reservation is requested is the time defined at the submission host. The format for `PBS_TZID` is a timezone location, rather than a timezone POSIX abbreviation. Examples of values for `PBS_TZID` are:

`America/Los_Angeles`

`America/Detroit`

`Europe/Berlin`

`Asia/Calcutta`

8.7.9.2 Reservation Errors

The following table describes the error messages that apply to reservations:

Table 8-5: Reservation Errors

Description of Error	Server Log Error Code	Error Message
Invalid syntax when specifying a standing reservation	15133	“pbs_rsub error: Undefined iCalendar syntax”
Recurrence rule has both a COUNT and an UNTIL parameter	15134	“pbs_rsub error: Undefined iCalendar syntax. COUNT or UNTIL is required”
Recurrence rule missing valid COUNT or UNTIL parameter	15134	“pbs_rsub error: Undefined iCalendar syntax. A valid COUNT or UNTIL is required”

Table 8-5: Reservation Errors

Description of Error	Server Log Error Code	Error Message
Problem with the start and/or end time of the reservation, such as: Given start time is earlier than current date and time Missing start time or end time End time is earlier than start time	15086	“pbs_rsub: Bad time specification(s)”
Reservation duration exceeds 24 hours and the recurrence frequency, <code>FREQ</code> , is set to <code>DAILY</code>	15129	“pbs_rsub error: DAILY recurrence duration cannot exceed 24 hours”
Reservation duration exceeds 7 days and the frequency <code>FREQ</code> is set to <code>WEEKLY</code>	15128	“pbs_rsub error: WEEKLY recurrence duration cannot exceed 1 week”
Reservation duration exceeds 1 hour and the frequency <code>FREQ</code> is set to <code>HOURLY</code> or the BY-rule is set to <code>BYHOUR</code> and occurs every hour, such as <code>BYHOUR=9, 10</code>	15130	“pbs_rsub error: HOURLY recurrence duration cannot exceed 1 hour”
The <code>PBS_TZID</code> environment variable is not set correctly at the submission host; rejection at submission host	None	“pbs_rsub error: a valid <code>PBS_TZID</code> timezone environment variable is required”
The <code>PBS_TZID</code> environment variable is not set correctly at the submission host; rejection at Server	15135	“Unrecognized <code>PBS_TZID</code> environment variable”

8.7.9.3 Time Required Between Reservations

Leave enough time between reservations for the reservations and jobs in them to clean up. A job consumes resources even while it is in the E or exiting state. This can take longer when large files are being staged. If the job is still running when the reservation ends, it may take

up to two minutes to be cleaned up. The reservation itself cannot finish cleaning up until its jobs are cleaned up. This will delay the start time of jobs in the next reservation unless there is enough time between the reservations for cleanup.

8.7.10 Reservation Information in the Accounting Log

The PBS Server writes an accounting record for each reservation in the job accounting file. The accounting record for a reservation is similar to that for a job. The accounting record for any job belonging to a reservation will include the reservation ID. See [“Accounting Log” on page 423 of the PBS Professional Reference Guide](#).

8.7.11 Cannot Mix Reservations and mpp*

Do not request any mpp* resources in a reservation. PBS mpp* resources are loosely coupled to Cray resources, and those Cray resources are not completely controlled by PBS. A reservation requesting mppnodes, for example, does not prevent ALPS from running another job on those nodes. If this happens, the PBS job in the reservation is prevented from running, even though those resources are reserved. Mixing reservations and mpp* resources would lead to disappointment.

8.8 Dedicated Time

Dedicated time is one or more specific time periods defined by the administrator. These are not repeating time periods. Each one is individually defined.

During dedicated time, the only jobs PBS starts are those in special dedicated time queues. PBS schedules non-dedicated jobs so that they will not run over into dedicated time. Jobs in dedicated time queues are also scheduled so that they will not run over into non-dedicated time. PBS will attempt to backfill around the dedicated-non-dedicated time borders.

PBS uses walltime to schedule within and around dedicated time. If a job is submitted without a walltime to a non-dedicated-time queue, it will not be started until all dedicated time periods are over. If a job is submitted to a dedicated-time queue without a walltime, it will never run.

To submit a job to be run during dedicated time, use the -q <queue name> option to qsub and give the name of the dedicated-time queue you wish to use as the queue name. Queues are created by the administrator; see your administrator for queue name(s).

8.9 Using Comprehensive System Accounting

PBS support for CSA on SGI systems is no longer available. The CSA functionality for SGI systems has been **removed** from PBS. You can use CSA on Cray systems.

CSA provides accounting information about user jobs, called user job accounting.

CSA works the same with and without PBS. To run user job accounting, either you must specify the file to which raw accounting information will be written, or an environment variable must be set. The environment variable is “ACCT_TMPDIR”. This is the directory where a temporary file of raw accounting data is written.

To run user job accounting, you issue the CSA command “**ja <filename>**” or, if the environment variable “ACCT_TMPDIR” is set, “**ja**”. In order to have an accounting report produced, you issue the command “**ja -<options>**” where the options specify that a report will be written and what kind. To end user job accounting, you issue the command “**ja -t**”; the **-t** option can be included in the previous set of options. See the manpage on **ja** for details.

The starting and ending **ja** commands must be used before and after any other commands you wish to monitor. Here are examples of command line and a script:

On the command line:

```
qsub -N myjobname -l ncpus=1
ja myrawfile
sleep 50
ja -c > myreport
ja -t myrawfile
ctrl-D
```

Accounting data for your job (sleep 50) is written to myreport.

If you create a file foo with these commands:

```
#PBS -N myjobname
#PBS -l ncpus=1
ja myrawfile
sleep 50
ja -c > myreport
ja -t myrawfile
```

Then you could run this script via **qsub**:

```
qsub foo
```

This does the same thing, via the script “foo”.

8.10 Running PBS in a UNIX DCE Environment

PBS Professional includes optional support for UNIX-based DCE. (By optional, we mean that the customer may acquire a copy of PBS Professional with the standard security and authentication module replaced with the DCE module.)

There are two `-W` options available with `qsub` which will enable a `dcelogin` context to be set up for the job when it eventually executes. The user may specify either an encrypted password or a forwardable/renewable Kerberos V5 TGT.

Specify the “`-W cred=dce`” option to `qsub` if a forwardable, renewable, Kerberos V5, TGT (ticket granting ticket) with the user as the listed principal is what is to be sent with the job. If the user has an established credentials cache and a non-expired, forwardable, renewable, TGT is in the cache, that information is used.

The other choice, “`-W cred=dce:pass`”, causes the `qsub` command to interact with the user to generate a DES encryption of the user's password. This encrypted password is sent to the PBS Server and MOM processes, where it is placed in a job-specific file for later use by `pbs_mom` in acquiring a DCE login context for the job. The information is destroyed if the job terminates, is deleted, or aborts.

IMPORTANT:

The “`-W pwd= ' '`” option to `qsub` has been superseded by the above two options, and therefore should no longer be used.

Any acquired login contexts and accompanying DCE credential caches established for the job get removed on job termination or deletion.

`qsub -Wcred=dce <other qsub options> job-script`

IMPORTANT:

The “`-W cred`” option to `qsub` is not available under Windows.

8.11 Running PBS in a UNIX Kerberos Environment

PBS Professional includes optional support for Kerberos-only (i.e. no DCE) environment. (By optional, we mean that the customer may acquire a copy of PBS Professional with the standard security and authentication module replaced with the KRB5 module.) This is not supported under Windows.

To use a forwardable/renewable Kerberos V5 TGT specify the “-w cred=krb5” option to `qsub`. This will cause `qsub` to check the user's credential cache for a valid forwardable/renewable TGT which it will send to the Server and then eventually to the execution MOM. While it's at the Server and the MOM, this TGT will be periodically refreshed until either the job finishes or the maximum refresh time on the TGT is exceeded, whichever comes first. If the maximum refresh time on the TGT is exceeded, no KRB5 services will be available to the job, even though it will continue to run.

8.12 Support for Large Page Mode on AIX

A process running as part of a job can use large pages. The memory reported in `resources_used.mem` may be larger with large page sizes.

You can set an environment variable to request large memory pages:

```
LDR_CNTRL="LARGE_PAGE_DATA=M"
LDR_CNTRL="LARGE_PAGE_DATA=Y"
```

For more information see the man page for `setpcred`. This can be viewed with the command "man setpcred" on an AIX machine.

You can run a job that requests large page memory in "mandatory mode":

```
% qsub
export LDR_CNTRL="LARGE_PAGE_DATA=M"
/path/to/exe/bigprog
^D
```

You can run a job that requests large page memory in "advisory mode":

```
% qsub
export LDR_CNTRL="LARGE_PAGE_DATA=Y"
/path/to/exe/bigprog
^D
```

8.13 Checking License Availability

You can check to see where licenses are available. You can do either of the following:

- Display license information for the current host:
- Display resources available (including licenses) on all hosts:

```
qstat -Bf
```

```
qmgr
```

```
Qmgr: print node @default
```

When looking at the server's `license_count` attribute, use the sum of the *Avail_Global* and *Avail_Local* values.

8.14 Adjusting Job Running Time

8.14.1 Shrink-to-fit Jobs

PBS allows you to submit a job whose running time can be adjusted to fit into an available scheduling slot. The job's minimum and maximum running time are specified in the `min_walltime` and `max_walltime` resources. PBS chooses the actual `walltime`. Any job that requests `min_walltime` is a **shrink-to-fit** job.

8.14.1.1 Requirements for a Shrink-to-fit Job

A job must have a value for `min_walltime` to be a shrink-to-fit job. Shrink-to-fit jobs are not required to request `max_walltime`, but it is an error to request `max_walltime` and not `min_walltime`.

Jobs that do not have values for `min_walltime` are not shrink-to-fit jobs, and you can specify their `walltime`.

8.14.1.2 Comparison Between Shrink-to-fit and Non-shrink-to-fit Jobs

The only difference between a shrink-to-fit and a non-shrink-to-fit job is how the job's `walltime` is treated. PBS sets the `walltime` when it runs the job. Any `walltime` value that exists before the job runs is ignored.

8.14.2 Using Shrink-to-fit Jobs

If you have jobs that can run for less than the expected time needed and still make useful progress, you can make them shrink-to-fit jobs in order to maximize utilization.

You can use shrink-to-fit jobs for the following:

- Jobs that are internally checkpointed. This includes jobs which are part of a larger effort, where a job does as much work as it can before it is killed, and the next job in that effort takes up where the previous job left off.
- Jobs using periodic PBS checkpointing
- Jobs whose real running time might be much less than the expected time
- When you have dedicated time for system maintenance, and you want to take advantage of time slots right up until shutdown, you can run speculative shrink-to-fit jobs if you can risk having a job killed before it finishes. Similarly, speculative jobs can take advantage of the time just before a reservation starts
- Any job where you do not mind running the job as a speculative attempt to finish some work

8.14.3 Running Time of a Shrink-to-fit Job

8.14.3.1 Setting Running Time Range for Shrink-to-fit Jobs

It is only required that the job request `min_walltime` to be a shrink-to-fit job. Requesting `max_walltime` without requesting `min_walltime` is an error.

You can set the job's running time range by requesting `min_walltime` and `max_walltime`, for example:

```
qsub -l min_walltime=<min walltime>, max_walltime=<max walltime> <job script>
```

8.14.3.2 Setting walltime for Shrink-to-fit Jobs

For a shrink-to-fit job, PBS sets the `walltime` resource based on the values of `min_walltime` and `max_walltime`, regardless of whether `walltime` is specified for the job.

PBS examines each shrink-to-fit job when it gets to it, and looks for a time slot whose length is between the job's `min_walltime` and `max_walltime`. If the job can fit somewhere, PBS sets the job's `walltime` to a duration that fits the time slot, and runs the job. The chosen value for `walltime` is visible in the job's `resource_list.walltime` attribute. Any existing `walltime` value, regardless of where it comes from, e.g. previous execution, is reset to the new calculated running time.

If a shrink-to-fit job is run more than once, PBS recalculates the job's running time to fit an available time slot that is between `min_walltime` and `max_walltime`, and resets the job's `walltime`, each time the job is run.

8.14.3.3 How PBS Places Shrink-to-fit Jobs

The PBS scheduler treats shrink-to-fit jobs the same way as it treats non-shrink-to-fit jobs when it schedules them to run. The scheduler looks at each job in order of priority, and tries to run it on available resources. If a shrink-to-fit job can be shrunk to fit in an available slot, the scheduler runs it in its turn. The scheduler chooses a time slot that is at least as long as the job's `min_walltime` value. A shrink-to-fit job may be placed in a time slot that is shorter than its `max_walltime` value, even if a longer time slot is available.

For a multi-vnode job, PBS chooses a `walltime` that works for all of the chunks required by the job, and places job chunks according to the placement specification.

8.14.4 Shrink-to-fit Jobs and Time Boundaries

The time boundaries that constrain job running time are the following:

- Reservations
- Dedicated time
- Primetime
- Start time for a top job

Time boundaries are not affected by shrink-to-fit jobs.

A shrink-to-fit job can shrink to avoid time boundaries, as long as the available time slot before the time boundary is greater than `min_walltime`.

If any job is already running, whether or not it is shrink-to-fit, and you introduce a new period of dedicated time that would impinge on the job's running time, PBS does not kill or otherwise take any action to prevent the job from hitting the new boundary.

8.14.4.1 Shrink-to-fit Jobs and Prime Time

If you have enabled prime time by setting `backfill_prime` to `True`, shrink-to-fit jobs will honor the boundary between primetime and non-primetime. If `prime_spill` is `True`, shrink-to-fit jobs are scheduled so that they cross the prime-nonprime boundary by up to `prime_spill` duration only. If `prime_exempt_anytime_queues` is set to `True`, a job submitted in an anytime queue is not affected by primetime boundaries.

8.14.5 Modifying Shrink-to-fit and Non-shrink-to-fit Jobs

8.14.5.1 Modifying min_walltime and max_walltime

You can change min_walltime and/or max_walltime for a shrink-to-fit job by using the `qalter` command. Any changes take effect after the current scheduling cycle. Changes affect only queued jobs; running jobs are unaffected unless they are rerun.

8.14.6 Viewing Running Time for a Job

8.14.6.1 Viewing min_walltime and max_walltime

You can use `qstat -f` to view the values of the min_walltime and max_walltime. For example:

```
% qsub -lmin_walltime=01:00:15, max_walltime=03:30:00 job.sh
<job-id>
% qstat -f <job-id>
...
resource_list.min_walltime=01:00:15
resource_list.max_walltime=03:30:00
```

You can use `tracejob` to display max_walltime and min_walltime as part of the job's resource list. For example:

```
12/16/2011 14:28:55 A user=pbsadmin group=Users
project=_pbs_project_default
...
Resource_List.max_walltime=10:00:00
Resource_List.min_walltime=00:00:10
```

8.14.6.2 Viewing walltime for a Shrink-to-fit Job

PBS sets a job's walltime only when the job runs. While the job is running, you can see its walltime via `qstat -f`. While the job is not running, you cannot see its real walltime; it may have a value set for walltime, but this value is ignored.

You can see the walltime value for a finished shrink-to-fit job if you are preserving job history. See [section 12.16, “Managing Job History”, on page 843](#).

8.14.7 Lifecycle of a Shrink-to-fit Job

8.14.7.1 Execution of Shrink-to-fit Jobs

Shrink-to-fit jobs are started just like non-shrink-to-fit jobs.

8.14.7.2 Termination of Shrink-to-fit Jobs

When a shrink-to-fit job exceeds the `walltime` PBS has set for it, it is killed by PBS exactly as a non-shrink-to-fit job is killed when it exceeds its `walltime`.

8.14.8 The `min_walltime` and `max_walltime` Resources

`max_walltime`

Maximum `walltime` allowed for a shrink-to-fit job. Job's actual `walltime` is between `max_walltime` and `min_walltime`. PBS sets `walltime` for a shrink-to-fit job. If this resource is specified, `min_walltime` must also be specified. Must be greater than or equal to `min_walltime`. Cannot be used for `resources_min` or `resources_max`. Cannot be set on job arrays or reservations. If not specified, PBS uses 5 years as the maximum time slot. Can be requested only outside of a select statement. Non-consumable. Default: None. Type: duration. Python type: `pbs.duration`

`min_walltime`

Minimum `walltime` allowed for a shrink-to-fit job. When this resource is specified, job is a shrink-to-fit job. If this attribute is set, PBS sets the job's `walltime`. Job's actual `walltime` is between `max_walltime` and `min_walltime`. Must be less than or equal to `max_walltime`. Cannot be used for `resources_min` or `resources_max`. Cannot be set on job arrays or reservations. Can be requested only outside of a select statement. Non-consumable. Default: None. Type: duration. Python type: `pbs.duration`

8.14.9 Caveats and Restrictions for Shrink-to-fit Jobs

It is erroneous to specify `max_walltime` for a job without specifying `min_walltime`. If attempted via `qsub` or `qalter`, the following error is printed:

```
'Can not have "max_walltime" without "min_walltime"'
```

It is erroneous to specify a `min_walltime` that is greater than `max_walltime`. If attempted via `qsub` or `qalter`, the following error is printed:

```
'"min_walltime" can not be greater than "max_walltime"'
```

Job arrays cannot be shrink-to-fit. You cannot have a shrink-to-fit job array. It is erroneous to specify a `min_walltime` or `max_walltime` for a job array. If attempted via `qsub` or `qalter`, the following error is printed:

```
"min_walltime" and "max_walltime" are not valid resources for a job array'
```

Reservations cannot be shrink-to-fit. You cannot have a shrink-to-fit reservation. It is erroneous to set `min_walltime` or `max_walltime` for a reservation. If attempted via `pbs_rsub`, the following error is printed:

```
"min_walltime" and "max_walltime" are not valid resources for  
reservation.'
```

It is erroneous to set `resources_max` or `resources_min` for `min_walltime` and `max_walltime`. If attempted, the following error message is displayed, whichever is appropriate:

```
"Resource limits can not be set for min_walltime"
```

```
"Resource limits can not be set for max_walltime"
```

Chapter 9

Job Arrays

This chapter describes job arrays and their use. A job array represents a collection of jobs which only differ by a single index parameter. The purpose of a job array is twofold. It offers the user a mechanism for grouping related work, making it possible to submit, query, modify and display the set as a single unit. Second, it offers a way to possibly improve performance, because the batch system can use certain known aspects of the collection for speedup.

9.1 Definitions

Subjob

Individual entity within a job array (e.g. **1234[7]**, where **1234[]** is the job array itself, and **7** is the index) which has many properties of a job as well as additional semantics (defined below.)

Sequence_number

The numeric part of a job or job array identifier, e.g. **1234**.

Subjob index

The unique index which differentiates one subjob from another. This must be a non-negative integer.

Job array identifier

The identifier returned upon success when submitting a job array. The format is **sequence_number[]** or `sequence_number[].server.domain.com`.

Job array range

A set of subjobs within a job array. When specifying a range, indices used must be valid members of the job array's indices.

9.1.1 Description

A job array is a compact representation of one or more jobs, called subjobs when part of a Job array, which have the same job script, and have the same values for all attributes and resources, with the following exceptions:

- each subjob has a unique index
- Job Identifiers of subjobs only differ by their indices
- the state of subjobs can differ

All subjobs within a job array have the same scheduling priority.

A job array is submitted through a single command which returns, on success, a “job array identifier” with a server-unique sequence number. Subjob indices are specified at submission time. These can be:

- a contiguous range, e.g. 1 through 100
- a range with a stepping factor, e.g. every second entry in 1 through 100 (1, 3, 5, ... 99)

A job array identifier can be used

- by itself to represent the set of all subjobs of the job array
- with a single index (a “job array identifier”) to represent a single subjob
- with a range (a “job array range”) to represent the subjobs designated by the range

9.1.2 Identifier Syntax

Job arrays have three identifier syntaxes:

- The job array object itself : 1234[.server or 1234[]
- A single subjob of a job array with index M: 1234[M].server or 1234[M]
- A range of subjobs of a job array: 1234[X-Y:Z].server or 1234[X-Y:Z]

Examples:

1234[.server.domain.com Full job array identifier

1234[] Short job array identifier

1234[73] Subjob identifier of the 73rd index of job array 1234[]

1234 Error, if 1234[] is a job array

1234.server.domain.com Error, if 1234[.server.domain.com is a job array

The sequence number (1234 in 1234[.server) is unique, so that jobs and job arrays cannot share a sequence number.

Note: Since some shells, for example `csh` and `tcsh`, read “[“ and “]” as shell metacharacters, job array names and subjob names will need to be enclosed in double quotes for all PBS commands.

Example:

```
qdel "1234.myhost[5]"
qdel "1234.myhost[]"
```

Single quotes will work, except where you are using shell variable substitution.

9.2 qsub: Submitting a Job Array

To submit a job array, `qsub` is used with the option **-J range**, where **range** is of the form **X-Y[:Z]**. **X** is the starting index, **Y** is the ending index, and **Z** is the optional **stepping factor**. **X** and **Y** must be whole numbers, and **Z** must be a positive integer. **Y** must be greater than **X**. If **Y** is not a multiple of the stepping factor above **X**, (i.e. it won't be used as an index value) the highest index used will be the next below **Y**. For example, 1-100:2 gives 1, 3, 5, ... 99.

Blocking `qsub` waits until the entire job array is complete, then returns the exit status of the job array.

Interactive submission of job arrays is not allowed.

Examples:

Example 9-1: To submit a job array of 10,000 subjobs, with indices 1, 2, 3, ... 10000:

```
$ qsub -J 1-10000 job.scr
1234[] .server.domain.com
```

Example 9-2: To submit a job array of 500 subjobs, with indices 500, 501, 502, ... 1000:

```
$ qsub -J 500-1000 job.scr
1235[] .server.domain.com
```

Example 9-3: To submit a job array with indices 1, 3, 5 ... 999:

```
$ qsub -J 1-1000:2 job.scr
1236[] .server.domain.com
```

9.2.1 Interactive Job Submission

Job arrays do not support interactive submission.

9.3 Job Array Attributes

Job arrays and subjobs have all of the attributes of a job. In addition, they have the following when appropriate. These attributes are read-only.

Table 9-1: Job Array Attributes

Name	Type	Applies To	Value
array	boolean	job array	True if item is job array
array_id	string	subjob	Subjob's job array identifier
array_index	string	subjob	Subjob's index number
array_state_count	string	job array	Similar to state_count attribute for server and queue objects. Lists number of subjobs in each state.
array_indices_remaining	string	job array	List of indices of subjobs still queued. Range or list of ranges, e.g. 500, 552, 596-1000
array_indices_submitted	string	job array	Complete list of indices of subjobs given at submission time. Given as range, e.g. 1-100

9.4 Job Array States

See [“Job Array States” on page 414 of the PBS Professional Reference Guide](#) and [“Subjob States” on page 415 of the PBS Professional Reference Guide](#).

9.5 PBS Environmental Variables

Table 9-2: PBS Environmental Variables

Environment Variable Name	Used For	Description
\$PBS_ARRAY_INDEX	subjobs	Subjob index in job array, e.g. 7
\$PBS_ARRAY_ID	subjobs	Identifier for a job array. Sequence number of job array, e.g. 1234[.server]
\$PBS_JOBID	Jobs, sub-jobs	Identifier for a job or a subjob. For subjob, sequence number and subjob index in brackets, e.g. 1234[7].server

9.6 File Staging

File staging for job arrays is like that for jobs, with an added variable to specify the subjob index. This variable is `^array_index^`. This is the name of the variable that will be used for the actual array index. The stdout and stderr files follow the naming convention for jobs, but include the identifier of the job array, which includes the subscripted index. As with jobs, the stagein and stageout keywords require the `-W` option to qsub.

9.6.1 Specifying Files To Be Staged In or Staged Out

You can specify files to be staged in before the job runs and staged out after the job runs by using `-W stagein=file_list` and `-W stageout=file_list`. You can use these as options to qsub, or as directives in the job script.

The *file_list* takes the form:

```
execution_path@storage_hostname:storage_path[,...]
```

for both stagein and stageout.

The name *execution_path* is the name of the file in the job's staging and execution directory (on the execution host). The *execution_path* can be relative to the job's staging and execution directory, or it can be an absolute path.

The '@' character separates the execution specification from the storage specification.

The name *storage_path* is the file name on the host specified by *storage_hostname*. For stagein, this is the location where the input files come from. For stageout, this is where the output files end up when the job is done. You must specify a *storage_hostname*. The name can be absolute, or it can be relative to the user's home directory on the remote machine.

IMPORTANT:

It is advisable to use an absolute pathname for the *storage_path*. Remember that the path to your home directory may be different on each machine, and that when using `sandbox = PRIVATE`, you may or may not have a home directory on all execution machines.

For stagein, the direction of travel is **from** *storage_path* **to** *execution_path*.

For stageout, the direction of travel is **from** *execution_path* **to** *storage_path*.

When staging more than one filename, separate the filenames with a comma and enclose the entire list in double quotes.

Examples:

storage_path: store:/film

Data files used as input: frame1, frame2, frame3

execution_path: pix

Executable: a.out

For this example, *a.out* produces *frame2.out* from *frame2*.

```
#PBS -W stagein=pix/in/frame^array_index^@store:/film/frame^array_index^
#PBS- W stageout=pix/out/frame^array_index^.out @store:/film/
      frame^array_index^.out
#PBS -J 1-3 a.out frame$PBS_ARRAY_INDEX ./in ./out
```

Note that the stageout statement is all one line, broken here for readability.

The result will be that the user's directory named "film" contains the original files *frame1*, *frame2*, *frame3*, plus the new files *frame1.out*, *frame2.out* and *frame3.out*.

9.6.1.1 Scripts

Example 9-4: In this example, we have a script named *ArrayScript* which calls *scriptlet1* and *scriptlet2*.

All three scripts are located in `/homedir/testdir`.

```
#!/bin/sh
#PBS -N ArrayExample
#PBS -J 1-2
echo "Main script: index " $PBS_ARRAY_INDEX
/homedir/testdir/scriptlet$PBS_ARRAY_INDEX
```

In our example, `scriptlet1` and `scriptlet2` simply echo their names. We run `ArrayScript` using the `qsub` command:

qsub ArrayScript

Example 9-5: In this example, we have a script called `StageScript`. It takes two input files, `dataX` and `extraX`, and makes an output file, `newdataX`, as well as echoing which iteration it is on. The `dataX` and `extraX` files will be staged from `inputs` to `work`, then `newdataX` will be staged from `work` to `outputs`.

```
#!/bin/sh
#PBS -N StagingExample
#PBS -J 1-2
#PBS -W stagein="/homedir/work/data^array_index^
    @host1:/homedir/inputs/data^array_index^, \
    /homedir/work/extra^array_index^ \
    @host1:/homedir/inputs/extra^array_index^"
#PBS -W stageout=/homedir/work/newdata^array_index^
    @host1:/homedir/outputs/newdata^array_index^
echo "Main script: index " $PBS_ARRAY_INDEX
cd /homedir/work
cat data$PBS_ARRAY_INDEX extra$PBS_ARRAY_INDEX \
    >> newdata$PBS_ARRAY_INDEX
```

Local path (execution directory):

```
/homedir/work
```

Remote host (data storage host):

```
host1
```

Remote path for inputs (original data files `dataX` and `extraX`):

```
/homedir/inputs
```

Remote path for results (output of computation `newdataX`):

```
/homedir/outputs
```

StageScript resides in `/homedir/testdir`. In that directory, we can run it by typing:

```
qsub StageScript
```

It will run in `/homedir`, our home directory, which is why the line

```
"cd /homedir/work"
```

is in the script.

Example 9-6: In this example, we have the same script as before, but we will run it in a staging and execution directory created by PBS. StageScript takes two input files, `dataX` and `extraX`, and makes an output file, `newdataX`, as well as echoing which iteration it is on. The `dataX` and `extraX` files will be staged from `inputs` to the staging and execution directory, then `newdataX` will be staged from the staging and execution directory to `outputs`.

```
#!/bin/sh
#PBS -N StagingExample
#PBS -J 1-2
#PBS -W stagein="data^array_index^\
    @host1:/homedir/inputs/data^array_index^, \
    extra^array_index^ \
    @host1:/homedir/inputs/extra^array_index^"
#PBS -W stageout=newdata^array_index^\
    @host1:/homedir/outputs/newdata^array_index^
echo "Main script: index " $PBS_ARRAY_INDEX
cat data$PBS_ARRAY_INDEX extra$PBS_ARRAY_INDEX \
    >> newdata$PBS_ARRAY_INDEX
```

Local path (execution directory):

created by PBS; we don't know the name

Remote host (data storage host):

```
host1
```

Remote path for inputs (original data files `dataX` and `extraX`):

```
/homedir/inputs
```

Remote path for results (output of computation `newdataX`):

```
/homedir/outputs
```

StageScript resides in /homedir/testdir. In that directory, we can run it by typing:

```
qsub StageScript
```

It will run in the staging and execution directory created by PBS. See [section 8.6, “Input/Output File Staging”](#), on page 198.

9.6.1.2 Output Filenames

The name of the job array will default to the script name if no name is given via qsub -N.

For example, if the sequence number were 1234,

```
#PBS -N fixgamma
```

would give stdout for index number 7 the name fixgamma.o1234.7 and stderr the name fixgamma.e1234.7. The name of the job array can also be given through stdin.

9.6.2 Job Array Staging Syntax on Windows

In Windows the stagein and stageout string must be contained in double quotes when using ^array_index^.

Example of a stagein:

```
qsub -W stagein="foo.^array_index^@host-1:C:\WINNT\Temp\foo.^array_index^"
-J 1-5 stage_script
```

Example of a stageout:

```
qsub -W stageout="C:\WINNT\Temp\foo.^array_index^@host-
1:Q:\my_username\foo.^array_index^_out" -J 1-5 stage_script
```

9.7 PBS Commands

9.7.1 PBS Commands Taking Job Arrays as Arguments

Note: Some shells such as csh and tesh use the square bracket (“[“, “]”) as a metacharacter. When using one of these shells, and a PBS command taking subjobs, job arrays or job array ranges as arguments, the subjob, job array or job array range must be enclosed in double quotes.

The following table shows PBS commands that take job arrays, subjobs or ranges as arguments. The cells in the table indicate which objects are acted upon. In the table,

Array[] = the job array object

Array[Range] =	the set of subjobs of the job array with indices in range given
Array[Index] =	the individual subjob of the job array with the index given
Array[RUNNING] =	the set of subjobs of the job array which are currently running
Array[QUEUED] =	the set of subjobs of the job array which are currently queued
Array[REMAINING] =	the set of subjobs of the job array which are queued or running
Array[DONE]=	the set of subjobs of the job array which have finished running

Table 9-3: PBS Commands Taking Job Arrays as Arguments

Command	Argument to Command		
	Array[]	Array[Range]	Array[Index]
qstat	Array[]	Array[Range]	Array[Index]
qdel	Array[] & Array[REMAINING]	Array[Range] where Array[REMAINING]	Array[Index]
qalter	Array[]	erroneous	erroneous
qorder	Array[]	erroneous	erroneous
qmove	Array[] & Array[QUEUED]	erroneous	erroneous
qhold	Array[] & Array[QUEUED]	erroneous	erroneous
qrls	Array[] & Array[QUEUED]	erroneous	erroneous
qrerun	Array[RUNNING] & Array[DONE]	Array[Range] where Array[RUNNING]	Array[Index]
qrun	erroneous	Array[Range] where Array[QUEUED]	Array[Index]
tracejob	erroneous	erroneous	Array[Index]
qsig	Array[RUNNING]	Array[Range] where Array[RUNNING]	Array[Index]

Table 9-3: PBS Commands Taking Job Arrays as Arguments

	Argument to Command		
Command	Array[]	Array[Range]	Array[Index]
qmsg	erroneous	erroneous	erroneous

9.7.2 qstat: Status of a Job Array

The `qstat` command is used to query the status of a Job Array. The default output is to list the Job Array in a single line, showing the Job Array Identifier. Options can be combined. To show the state of all running subjobs, use `-t -r`. To show the state only of subjobs, not job arrays, use `-t -J`.

Table 9-4: Job Array and Subjob Options to qstat

Option	Result
-t	Shows state of job array object and subjobs. Will also show state of jobs.
-J	Shows state only of job arrays.
-p	Prints the default display, with column for Percentage Completed. For a job array, this is the number of subjobs completed or deleted divided by the total number of subjobs. For a job, it is time used divided by time requested.

Examples:

We run an example job and an example job array, on a machine with 2 processors:
demoscript:

```
#!/bin/sh
#PBS -N JobExample
sleep 60
```

arrayscript:

```
#!/bin/sh
#PBS -N ArrayExample
#PBS -J 1-5
sleep 60
```

We run these scripts using qsub.

```
qsub arrayscript
1235[ ].host
qsub demoscrypt
1236.host
```

Then:

qstat

Job id	Name	User	Time Use	S	Queue
1235[].host	ArrayExample	user1		0 B	workq
1236.host	JobExample	user1		0 Q	workq

qstat -J

Job id	Name	User	Time Use	S	Queue
1235[].host	ArrayExample	user1		0 B	workq

qstat -p

Job id	Name	User	% done	S	Queue
1235[].host	ArrayExample	user1	0	B	workq
1236.host	JobExample	user1	--	Q	workq

qstat -t

Job id	Name	User	Time Use	S	Queue
-----	-----	-----	-----	-	-----
1235[].host	ArrayExample	user1		0 B	workq
1235[1].host	ArrayExample	user1	00:00:00	R	workq
1235[2].host	ArrayExample	user1	00:00:00	R	workq
1235[3].host	ArrayExample	user1		0 Q	workq
1235[4].host	ArrayExample	user1		0 Q	workq
1235[5].host	ArrayExample	user1		0 Q	workq
1236.host	JobExample	user1		0 Q	workq

qstat -Jt

Job id	Name	User	Time Use	S	Queue
-----	-----	-----	-----	-	-----
1235[1].host	ArrayExample	user1	00:00:00	R	workq
1235[2].host	ArrayExample	user1	00:00:00	R	workq
1235[3].host	ArrayExample	user1		0 Q	workq
1235[4].host	ArrayExample	user1		0 Q	workq
1235[5].host	ArrayExample	user1		0 Q	workq

After the first two subjobs finish:

qstat -Jtp

Job id	Name	User	% done	S	Queue
-----	-----	-----	-----	-	-----
1235[1].host	ArrayExample	user1	100 X		workq
1235[2].host	ArrayExample	user1	100 X		workq
1235[3].host	ArrayExample	user1	--	R	workq
1235[4].host	ArrayExample	user1	--	R	workq

```
1235[5].host ArrayExample user1      -- Q workq
```

```
qstat -pt
```

Job id	Name	User	% done	S	Queue
1235[5].host	ArrayExample	user1	40	B	workq
1235[1].host	ArrayExample	user1	100	X	workq
1235[2].host	ArrayExample	user1	100	X	workq
1235[3].host	ArrayExample	user1	--	R	workq
1235[4].host	ArrayExample	user1	--	R	workq
1235[5].host	ArrayExample	user1	--	Q	workq
1236.host	JobExample	user1	--	Q	workq

Now if we wait until only the last subjob is still running:

```
qstat -rt
```

Job ID	Username	Queue	Jobname	SessID	NDS	TSK	Memory	Req'd Time	Req'd S	Elap Time
1235[5].host	user1	workq	ArrayExamp	3048	--	1	--	--	--	R 00:00
1236.host	user1	workq	JobExample	3042	--	1	--	--	--	R 00:00

```
qstat -Jrt
```

Job ID	Username	Queue	Jobname	SessID	NDS	TSK	Memory	Req'd Time	Req'd S	Elap Time
1235[5].host	user1	workq	ArrayExamp	048	--	1	--	--	--	R 00:01

9.7.3 **qdel: Deleting a Job Array**

The `qdel` command will take a job array identifier, subjob identifier or job array range. The indicated object(s) are deleted, including any currently running subjobs. Running subjobs are treated like running jobs. Subjobs not running will be deleted and never run. Only one email is sent per deleted job array, so deleting a job array of 5000 subjobs results in one email being sent.

9.7.4 **qalter: Altering a Job Array**

The `qalter` command can only be used on a job array object, not on subjobs or ranges. Job array attributes are the same as for jobs.

9.7.5 **qorder: Ordering Job Arrays in the Queue**

The `qorder` command can only be used with job array objects, not on subjobs or ranges. This will change the queue order of the job array in association with other jobs or job arrays in the queue.

9.7.6 **qmove: Moving a Job Array**

The `qmove` command can only be used with job array objects, not with subjobs or ranges. Job arrays can only be moved from one server to another if they are in the 'Q', 'H', or 'W' states, and only if there are no running subjobs. The state of the job array object is preserved in the move. The job array will run to completion on the new server.

As with jobs, a `qstat` on the server from which the job array was moved will not show the job array. A `qstat` on the job array object will be redirected to the new server.

Note: The subjob accounting records will be split between the two servers.

9.7.7 **qhold: Holding a Job Array**

The `qhold` command can only be used with job array objects, not with subjobs or ranges. A hold can be applied to a job array only from the 'Q', 'B' or 'W' states. This will put the job array in the 'H', held, state. If any subjobs are running, they will run to completion. No queued subjobs will be started while in the 'H' state.

9.7.8 **qrls: Releasing a Job Array**

The `qrls` command can only be used with job array objects, not with subjobs or ranges. If the job array was in the ‘Q’ or ‘B’ state, it will be returned to that state. If it was in the ‘W’ state, it will be returned to that state unless its waiting time was reached, it will go to the ‘Q’ state.

9.7.9 **qrerun: Requeueing a Job Array**

The `qrerun` command will take a job array identifier, subjob identifier or job array range. If a job array identifier is given as an argument, it is returned to its initial state at submission time, or to its altered state if it has been altered. All of that job array’s subjobs are requeued, which includes those that are currently running, and completed and deleted. If a subjob or range is given, those subjobs are requeued as jobs would be.

9.7.10 **qrun: Running a Job Array**

The `qrun` command takes a subjob or a range of subjobs, not a job array object. If a single subjob is given as the argument, it is run as a job would be. If a range of subjobs is given as the argument, the non-running subjobs within that range will be run.

9.7.11 **tracejob on Job Arrays**

The `tracejob` command can be run on job arrays and individual subjobs. When `tracejob` is run on a job array or a subjob, the same information is displayed as for a job, with additional information for a job array. Note that subjobs do not exist until they are running, so `tracejob` will not show any information until they are. When `tracejob` is run on a job array, the information displayed is only that for the job array object, not the subjobs. Job arrays themselves do not produce any MOM log information. Running `tracejob` on a job array will give information about why a subjob did not start.

9.7.12 **qsig: Signaling a Job Array**

If a job array object, subjob or job array range is given to `qsig`, all currently running subjobs within the specified set will be sent the signal.

9.7.13 **qmsg: Sending Messages**

The `qmsg` command is not supported for job arrays.

9.8 Other PBS Commands Supported for Job Arrays

9.8.1 qselect: Selection of Job Arrays

The default behavior of `qselect` is to return the job array identifier, without returning subjob identifiers.

Note: `qselect` will not return any job arrays when the state selection (`-s`) option restricts the set to 'R', 'S', 'T' or 'U', because a job array will never be in any of these states. However, `qselect` can be used to return a list of subjobs by using the `-t` option.

Options to `qselect` can be combined. For example, to restrict the selection to subjobs, use both the `-J` and the `-T` options. To select only running subjobs, use `-J -T -sR`.

Table 9-5: Options to `qselect` for Job Arrays

Option	Selects	Result
(none)	jobs, job arrays	Shows job and job array identifiers
<code>-J</code>	job arrays	Shows only job array identifiers
<code>-T</code>	jobs, subjobs	Shows job and subjob identifiers

9.9 Job Arrays and `xpbs`

`xpbs` does not support job arrays.

9.10 More on Job Arrays

9.10.1 Job Array Run Limits

Jobs and subjobs are treated the same way by job run limits. For example, if `max_user_run` is set to 5, a user can have a maximum of 5 subjobs and/or jobs running.

9.10.2 Starving

A job array's starving status is based on the queued portion of the array. This means that if there is a queued subjob which is starving, the job array is starving. A running subjob retains its starving status when it was started.

9.10.3 Job Array Dependencies

Job dependencies are supported:

- between job arrays and job arrays
- between job arrays and jobs
- between jobs and job arrays

Note: Job dependencies are not supported for subjobs or ranges of subjobs.

9.10.4 The “Rerunnable” Flag and Job Arrays

Job arrays are required to be rerunnable. PBS will not accept a job array that is not marked as rerunnable. You can submit a job array without specifying whether it is rerunnable, and PBS will automatically mark it as rerunnable.

9.10.5 Accounting

Job accounting records for job arrays and subjobs are the same as for jobs. When a job array has been moved from one server to another, the subjob accounting records are split between the two servers, except that there will be no ‘Q’ records for subjobs.

9.10.6 Checkpointing

Checkpointing is not supported for job arrays. On systems that support checkpointing, subjobs are not checkpointed, instead they run to completion.

9.10.7 Prologues and Epilogues

If defined, prologues and epilogues will run at the beginning and end of each subjob, but not for job arrays.

9.10.8 Job Array Exit Status

The exit status of a job array is determined by the status of each of the completed subjobs. It is only available when all valid subjobs have completed. The individual exit status of a completed subjob is passed to the epilogue, and is available in the ‘E’ accounting log record of that subjob.

Table 9-6: Job Array Exit Status

Exit Status	Meaning
0	All subjobs of the job array returned an exit status of 0. No PBS error occurred. Deleted subjobs are not considered
1	At least 1 subjob returned a non-zero exit status. No PBS error occurred.
2	A PBS error occurred.

9.10.9 Scheduling Job Arrays

All subjobs within a job array have the same scheduling priority.

9.10.9.1 Preemption

Individual subjobs may be preempted by higher priority work.

9.10.9.2 Peer Scheduling

Peer scheduling does not support job arrays.

9.10.9.3 Fairshare

Subjobs are treated like jobs with respect to fairshare ordering, fairshare accounting and fairshare limits. If running enough subjobs of a job array causes the priority of the owning entity to change, additional subjobs from that job array may not be the next to start.

9.10.9.4 Placement Sets and Node Grouping

All nodes associated with a single subjob should belong to the same placement set or node group. Different subjobs can be put on different placement sets or node groups.

9.11 Job Array Caveats

9.11.1 Job Arrays Are Rerunnable

Job arrays are required to be rerunnable, and are rerunnable by default.

9.11.2 Mail for Job Arrays

The `-m` and `-M qsub` options are ignored for job arrays. No status notifications are mailed for job arrays.

When stagein or stageout fails for a job array, PBS sends mail to the job owner.

Chapter 10

HPC Basic Profile Jobs

Support for HPCBP jobs is **deprecated**.

PBS Professional can schedule and manage jobs on one or more HPC Basic Profile compliant servers using the Grid Forum OGSA HPC Basic Profile web services standard. You can submit a generic job to PBS, so that PBS can run it on an HPC Basic Profile Server. This chapter describes how to use PBS for HPC Basic Profile jobs.

10.1 Definitions

HPC Basic Profile (HPCBP)

Proposed standard web services specification for basic job execution capabilities defined by the OGSA High Performance Computing Profile Working Group

HPC Basic Profile Server

Service that executes jobs from any HPC Basic Profile compliant client

HPCBP MOM

MOM that sends jobs for execution to an HPC Basic Profile Server. This MOM is a client-side implementation of the HPC Basic Profile Specification, and acts as a proxy for and interface to an HPC Basic Profile compliant server.

HPC Basic Profile Job, HPCBP Job

Generic job that can run either on vnodes managed by PBS or on nodes managed by HPC Basic Profile Server.

Job Submission Description Language (JSDL)

Language for describing the resource requirements of jobs

10.2 How HPC Basic Profile Jobs Work

10.2.1 Introduction

PBS automatically schedules jobs on vnodes managed by PBS Professional or on nodes managed by an HPC Basic Profile Server, without the need for you to specify destination-specific parameters. Whether the jobs run on PBS Professional or on an HPC Basic Profile Server is based only on site policies and resource availability.

You can use the `qstat` command for status reporting and the `qdel` command to cancel a job, regardless of where the job runs.

Jobs eligible to run on the HPCBP Server must specify only a single executable and its arguments, and must do so via the `qsub` command line. The job specification must be valid for both PBS and the HPCBP Server. A job that is eligible to run on the HPCBP Server is called an *HPCBP job* in this document.

10.2.2 Assigning Nodes and Resources to Jobs

The HPCBP MOM does not control the resources assigned from each node for a job. The HPC Basic Profile Server assigns resources to the job according to its scheduling policy.

If you specify HPCBP hosts as part of the job's select statement, the list of HPCBP hosts is passed to the HPCBP Server.

10.3 Environmental Requirements for HPCBP

10.3.1 User Account at HPCBP Server

You must be able to run commands at the HPCBP Server. You must have an account in the Domain Controller at the HPCBP Server.

10.3.2 HPCBP Submission Client Architecture

You can submit HPCBP jobs only from submission hosts that have the correct architecture. These are all supported Linux platforms on x86 and x86_64.

10.3.3 Password Requirement For Job Submission

The HPC Basic Profile Server requires a password and a username to perform operations such as job submission, status, termination etc. The PBS Server must pass credential information to the HPCBP MOM at the time of job submission.

Before submitting an HPCBP job, you must run the `pbs_password` command to store your password at the PBS server. When you submit an HPCBP job, you must supply a password. This is done in one of two ways:

- The administrator sets the `single_signon_password_enable` server attribute to True
- You use the `'-Wpwd'` option to the `qsub` command to pass credential information to the PBS Server

10.3.4 Location of Executable

The executable that your job runs must be available at the HPC Server. The following table lists how the path to the executable can be specified:

Table 10-1: Executable Path Specification

Path Specification	Location of Executable
You can specify an absolute path to the executable	Anywhere available to the HPCBP Server
You can specify a path relative to your home directory on the HPC Server	A path relative to your home directory on the HPC Server
You can specify just the name of the executable	The executable is in your PATH or in your default working directory

10.4 Submitting HPC Basic Profile Jobs

As with PBS jobs, you do not need to specify destination-specific parameters.

10.4.1 Restrictions on Submitting Jobs for Execution at HPCBP Server

10.4.1.1 Specifying Executable for Job

The job must specify exactly one executable and its arguments. This must be done on the `qsub` command line.

10.4.1.2 HPCBP Jobs Run on One HPCBP Server

The job must not be split across more than one HPCBP Server:

- It cannot be split across two or more HPCBP Servers
- It cannot be split across an HPCBP Server and another node

10.4.1.3 Number of CPUs and `mpiprocs`

For each chunk, the aggregate number of requested `ncpus` must match the aggregate number of requested `mpiprocs`. The default value per chunk for both `ncpus` and `mpiprocs` is 1. If you request 1 CPU per chunk, you do not have to specify the `mpiprocs`. If the requested values for `ncpus` and `mpiprocs` are different, an error message is logged to the HPCBP MOM log file and the job is rejected. So for example if you request

```
qsub -l select=3:ncpus=2:mem=8gb
```

the job is rejected because no `mpiprocs` were requested.

10.4.1.4 Number of `ompthreads`

For a job with more than one chunk that requests `ompthreads`, each chunk must request the same value for `ompthreads`. Otherwise, an error message is logged to the HPCBP MOM log file and the job is rejected.

10.4.1.5 Restrictions on Requesting `arch` Resource

Requesting a value for `arch` in an HPCBP job means requesting a node or nodes with that architecture from among the nodes controlled by the HPCBP Server. It is not necessary for a job to request a value for `arch`. An HPCBP job can request any `arch` value that can be satisfied by the HPCBP Server.

10.4.2 Using the **qsub** Command for HPCBP Jobs

Job submission for non-HPCBP jobs is unchanged. However, when you submit an HPCBP job, you must do the following:

- Specify only one executable and its arguments
- Specify executable and arguments in the **qsub** command line

10.4.2.1 **qsub** Syntax for HPCBP Jobs

```
qsub [-a date_time] [-A account_string] [-c interval] [-C directive_prefix]
[-e path] [-h ] [-I] [-j oe|eo] [-J X-Y[:Z]] [-k o|e|oe] [-l
resource_list] [-m mail_options] [-M user_list] [-N jobname] [-o path]
[-p priority] [-q queue] [-r y|n] [-S path] [-u user_list] [-W otherat-
tributes=value...] [-v variable_list] [-V ] [-z] -- cmd [arg1...]
```

or

```
qsub --version
```

where **cmd** is the executable, and **arg1** is the first argument in the list.

10.4.2.2 **qsub** Options for HPCBP Jobs

The options to the **qsub** command set the attributes for the job. The following table shows a list of PBS job attributes and their behavior for HPCBP jobs.

Table 10-2: Behavior of Job Attributes for HPCBP Jobs

PBS Job attribute	Behavior
interactive	Job is rejected with transient error
Resource List	See section 10.4.3, “Requesting Resources”, on page 260
Output path	Standard output is staged out to specified location
Error_path	Standard error is staged out to specified location
no_stdio_sockets	Unsupported
Shell_Path_List	Unsupported
Variable_List	User’s environment is passed to HPCBP Server

Table 10-2: Behavior of Job Attributes for HPCBP Jobs

PBS Job attribute	Behavior
alt_id	Set to job ID returned by HPC Server
exec_host	Same as standard. Set to list of hosts, with number of CPUs for each
exec_vnode	Same as standard. Set to list of vnodes, with number of CPUs and amount of memory
job_state	See section 10.5.1.1, “Job Status Reporting”, on page 263
resources_used	Set to cputime used and amount of memory requested
session_id	Returns process ID of process started by the HPCBP MOM for job management, not of HPCBP job itself
stime	Reported start time of job; may be inexact
substate	The job substate may not be same in HPC Basic Profile Server and PBS
group_list	Unsupported
stagein	Specified files are staged in
stageout	Specified files are staged out
umask	Unsupported

10.4.3 Requesting Resources

The following table shows the behavior for of PBS resources HPCBP jobs:

Table 10-3: PBS Resources and Their Behavior for HPCBP Jobs

PBS Resource	Behavior
arch	Same as standard.
cput	Amount of disk space for job
file	Same as standard

Table 10-3: PBS Resources and Their Behavior for HPCBP Jobs

PBS Resource	Behavior
host	Same as standard
mem	Same as standard
mpiprocs	Number of CPUs to be allocated to job
mppwidth	Unsupported
mppdepth	Unsupported
mppnppn	Unsupported
mppnodes	Unsupported
mpplabels	Unsupported
mppmem	Unsupported
mpphost	Unsupported
mpparch	Unsupported
ncpus	Same as standard
nice	Unsupported
nodect	Unsupported
ompthreads	Must specify equal number of ompthreads in all chunks of multi-chunk job
pcput	Same as standard
pmem	Same as standard
pvmem	Same as standard
software	Unsupported
vmem	Same as standard
vnode	Same as standard
walltime	Supported

Table 10-3: PBS Resources and Their Behavior for HPCBP Jobs

PBS Resource	Behavior
cpupercent	Unsupported
custom resources	Unsupported

10.4.4 Specifying Job Destination

If necessary, you can specify where your job should run. You can specify on which nodes you want to run your job by specifying a host name:

```
-lselect=host=<host name>
```

If your application can run only on Windows, then you should request PBS to run the job only on Windows HPC Server nodes by specifying the architecture:

```
-lselect=arch=<arch value returned from HPCBP MOM>
```

Similarly, if you want to run your application on Linux, then you need to specify that architecture:

```
-lselect=arch=linux
```

If you don't specify a value for the `arch` resource at the time of job submission, PBS will select vnodes based on availability and run your application there.

10.5 Managing HPCBP Jobs

10.5.1 Monitoring HPCBP Jobs

You can use `qstat -f <job ID>` to see a listing of your job's executable and its argument list.

For example, if your job request was:

```
qsub -- ping -n 100 127.0.0.1
```

The output of `qstat -f <job ID>` will be:

```
executable = <jSDL-hpcpa:Executable>ping</jSDL-hpcpa:Executable>
argument_list = <jSDL-hpcpa:Argument>-n</jSDL-hpcpa:Argument> <jSDL-hpcpa:Argument>100</jSDL-hpcpa:Argument> <jSDL-hpcpa:Argument>127.0.0.1</jSDL-hpcpa:Argument>
```

10.5.1.1 Job Status Reporting

PBS provides status reporting for HPC Basic Profile jobs via the `qstat` command. The HPCBP MOM contacts the HPC Basic Profile Server and returns status information to the PBS Server. The only information available is via the HPC Basic Profile.

The job states returned from HPC Basic Profile Server can be one of the following:

- Pending
- Running
- Failed
- Finished
- Terminated

However, the only states that are reported by `qstat` are

- Running
- Exiting

The HPCBP Server reports that the job is in *Running* state whether the job is waiting to run or is running.

Once a job transitions to any of the states *Terminated*, *Failed* or *Finished*, the HPCBP MOM will no longer query for the status of that job.

A job whose status is *Running* can become *Terminated*, *Failed*, or *Finished*, or *Exiting*.

10.5.1.2 Deleting jobs running at HPC Basic Profile Server

You can delete your jobs via the `qdel` command:

```
qdel <job ID>
```

10.6 Errors, Logging and Troubleshooting

10.6.1 Job Submission Password Problems

If you specify the wrong password, or the password is different from the one at the HPC Basic Profile Server:

- The HPCBP MOM rejects the job and the PBS Server sets the job's comment
- The PBS Server logs a message in the server log
- The PBS Server changes the state of the job to *Hold* and the substate to *waiting on*

dependency and keeps it in the queue

10.6.2 Job Format Problems

If you submit only a job script, without any executable and argument list, and PBS attempts to run the job on the HPCBP Server, the HPCBP MOM will log a message and return an error.

If you submit a job requesting non-HPCBP vnodes and HPCBP nodes, or requesting nodes from two different HPCBP Servers:

- The job is rejected
- The HPCBP MOM logs an error message

10.6.3 Password-related Job Deletion Issues

If any problem, such as bad user credentials, occurs during an attempt to delete a job:

- The `qdel` command displays an error message
- The PBS server writes the error message to the server log
- The HPCBP MOM logs an error message

10.6.4 Error Log Messages at Job Submission, Querying, and Deletion

The HPCBP MOM logs a warning message in the MOM log file whenever it gets any error or warning at the time of:

- Job submission
- Contacting the HPC Basic Profile Server to find job status
- Job deletion

The HPCBP MOM logs job errors in the file `<PBS job ID>.log`. The HPCBP MOM stages this file out to the location specified for `stdout` and `stderr` files.

The HPCBP MOM generates log messages depending on their event type and event class. You can use the `tracejob` command to see these log messages.

The following table shows the warning and error messages logged by the HPCBP MOM and the PBS Server:

Table 10-4: Warning and Error Messages Logged by HPCBP MOM

Error Condition	Logged by	Message
Password-related issues		
Bad user credential at the time of <code>qdel</code>	HPCBP MOM, PBS Server	<username>: unable to terminate the job with user's credentials
Cannot determine job state when finding status of jobs running at HPC Basic Profile Server	HPCBP MOM	<pbsnobody>: unable to determine the state of the job
Conversion of PBS job request to JSDL		
Problem with parsing job request	HPCBP MOM	unable to parse the job request
Job request contains a script	HPCBP MOM	can't submit job to HPC Basic Profile Server, HPCBP MOM doesn't accept job script
JSDL script file problem	HPCBP MOM	unable to create JSDL document
gSOAP-related problems		
cannot create SSL-based channel	HPCBP MOM	unable to create ssl-based channel to connect to the Web Service endpoint
Username token problem	HPCBP MOM	unable to add username/password to soap message
Cannot initialize gSOAP runtime environment	HPCBP MOM	unable to initialize gsoap runtime environment
Problems encountered during job submission		

Table 10-4: Warning and Error Messages Logged by HPCBP MOM

Error Condition	Logged by	Message
Cannot add SOAP Header	HPCBP MOM	unable to add soap header to the 'create activity' request message
Bad JSDL script file	HPCBP MOM	unable to open JSDL document
Problem with JSDL attribute	HPCBP MOM	error in reading contents of the JSDL document
Problem with HPCBP Server connection	HPCBP MOM	unable to submit job to the hpcbp web service endpoint
Problem with user's password	HPCBP MOM & PBS Server	unable to submit job with user's credential
Problem reading SOAP response	HPCBP MOM	unable to read HPCBP job identifier from create activity response
Problems encountered when deleting job		
Cannot add SOAP Header	HPCBP MOM	unable to add SOAP Header to the 'terminate activities' request message
Problem reading HPCBP job ID	HPCBP MOM	unable to read HPCBP job identifier
Bad HPC Basic Profile Server connection	HPCBP MOM	unable to connect to the HPCBP web service endpoint
Problem with user's password	HPCBP MOM, PBS Server	unable to terminate job with user's credentials
Received malformed response from HPCBP Server	HPCBP MOM	unable to parse the response received for job deletion request from HPCBP Server
Problems encountered when finding status of job		

Table 10-4: Warning and Error Messages Logged by HPCBP MOM

Error Condition	Logged by	Message
Cannot add SOAP Header	HPCBP MOM	unable to add SOAP Header to the 'get activity statuses' request message
Problem reading HPCBP JOB ID	HPCBP MOM	unable to read HPCBP job identifier
Bad HPC Basic Profile Server connection	HPCBP MOM	unable to connect to the HPCBP web service endpoint
Received malformed response from HPCBP Server	HPCBP MOM	unable to parse the job status response received from HPCBP Server
Problems encountered when finding node status		
Cannot add SOAP Header	HPCBP MOM	unable to add SOAP Header to the 'get factory attributes document' request message
Problem with reading the node status information	HPCBP MOM	unable to parse node status information received from the HPC Basic Profile Server
HPC Basic Profile Server Connection	HPCBP MOM	unable to connect to the HPCBP web service endpoint
mpiprocs-related error		
unequal ncpus & mpiprocs	HPCBP MOM	can't submit job to the HPC Basic Profile Server; total number of ncpus and mpiprocs requested are not equal
ompthreads error		
ompthreads are not equal across chunks	HPCBP MOM	can't submit job to the HPC Basic Profile Server; number of 'ompthreads' are not equal in multi-chunk job request
Generic Problems		

Table 10-4: Warning and Error Messages Logged by HPCBP MOM

Error Condition	Logged by	Message
No reply from HPCBP Server	HPCBP MOM	unable to receive response from hpcbp web service endpoint
OpenSSL library issues		
Cannot find OpenSSL libraries on system	HPCBP MOM	unable to find openssl libraries on the system.

10.6.5 Job State Transition Log Messages

See the following table for a list of the job transitions in the HPCBP Server and the associated actions by the HPCBP MOM:

Table 10-5: Job Transitions in HPCBP Server and Associated Actions by HPCBP MOM

Job Transitions in HPC Basic Profile Server		Message Logged By HPCBP MOM
Start State	End State	
<i>Pending</i>	<i>Running</i>	“job transitioned from pending to running”
<i>Pending</i>	<i>Terminated</i>	“job transitioned from pending to terminated”
<i>Running</i>	<i>Terminated</i>	“job transitioned from running to terminated”
<i>Running</i>	<i>Failed</i>	“job transitioned from running to failed”
<i>Running</i>	<i>Finished</i>	“job completed successfully”
<i>Pending</i>	<i>Finished</i>	“job transitioned from pending to finished”
<i>Pending</i>	<i>Failed</i>	“job transitioned from pending to failed”
<i>(none)</i>	<i>Failed</i>	“job first appeared in “Failed” state”

Whenever a job is submitted to the HPC Basic Profile Server, the HPCBP MOM logs the following message:

```
job submitted to HPCBP Server as jobid <hpcbp-jobid> in state <state>
```

10.7 Advice and Caveats

10.7.1 Differences Between PBS and HPCBP

- The `stime` attribute in the PBS accounting logs may not represent the exact start time for an HPCBP job.
- The HPCBP MOM does not use the `pbs_rcp` command for staging operations, regardless of whether the `PBS_SCP` environment variable has been set in the configuration file.

10.7.2 PBS Features Not Supported With HPCBP

- Peer Scheduling
- Job operations:
 - Suspend/resume
 - Checkpoint

10.7.2.1 Unsupported Commands

If the user or administrator runs the `pbsdsh` command for a job running on the HPCBP Server, the HPCBP MOM logs an error message to the MOM file and rejects the job.

The following commands and their API equivalents are not supported for jobs that end up running on the HPCBP Server:

- `qalter`
- `qsig`
- `qmsg`
- `pbsdsh`
- `pbs-report`
- `printjob`
- `pbs_rcp`
- `tracejob`
- `pbs_rsub`
- `pbs_rstat`
- `pbs_rdel`
- `qhold`
- `qrls`
- `qrerun`

10.8 See Also

For a description of how job attributes are translated into JSDL, see the PBS Professional External Reference Specification.

10.8.1 References

1. OGSA High Performance Computing Profile Working Group (OGSA-HPCP-WG) of the Open Grid Forum
<https://forge.gridforum.org/sf/projects/ogsa-hpcp-wg>
The HPC Basic Profile specification is GFD.114:
<http://www.ogf.org/documents/GFD.114.pdf>.
2. OGSA High Performance Computing Profile Working Group (OGSA-HPCP-WG) of the Open Grid Forum
<https://forge.gridforum.org/sf/projects/ogsa-hpcp-wg>
The HPC File Staging Profile Version 1.0:
<http://forge.ogf.org/sf/go/doc15024?nav=1>
3. OGSA Job Submission Description Language Working Group (JSDL - WG) of the Open Grid Forum
http://www.ogf.org/gf/group_info/view.php?group=jsdl-wg
The JSDL HPC Profile Application Extension, Version 1.0 is GFD 111:
<http://www.ogf.org/documents/GFD.111.pdf>
4. OGSA Usage Record Working Group (UR-WG) of the Open Grid Forum
The Usage Record - Format Recommendation is GFD.98
<http://www.ogf.org/documents/GFD.98.pdf>
5. Network Working Group, Uniform Resource Identifier (URI) : Generic Syntax
<http://www.rfc-editor.org/rfc/rfc3986.txt>

Chapter 11

Submitting Cray Jobs

11.1 Introduction

You can submit jobs that are designed to run on the Cray, using the PBS select and place syntax.

11.2 PBS Jobs on the Cray

When you submit a job that is designed to run on the Cray, you create a job script that contains the same `aprun` command as a non-PBS job, but submit the job using the PBS select and place syntax. You can translate the `mpp*` syntax into select and place syntax using the rules described in [section 11.3.2, “Automatic Translation of mpp* Resource Requests”, on page 277](#).

You can submit a PBS job using `mpp*` syntax, but `mpp*` syntax is deprecated.

If a job does not request a login node, one is chosen for it. A login node is assigned to each PBS job that runs on the Cray. The job script runs on this login node.

Jobs requesting a `vntype` of `cray_compute` are expected to have an `aprun` in the job script to launch the job on the compute nodes. PBS does not verify that the job script contains an `aprun` statement.

11.3 PBS Resources for the Cray

11.3.1 Built-in and Custom Resources for the Cray

PBS provides built-in and custom resources specifically created for jobs on the Cray. The custom resources are created by PBS to reflect Cray information such as segments or labels. PBS also provides some built-in resources for all platforms that have specific uses on the Cray.

11.3.1.1 Built-in Resources for All Platforms

accelerator

Indicates whether this vnode is associated with an accelerator. Host-level. Can be requested only inside of a select statement. On Cray, this resource exists only when there is at least one associated accelerator. On Cray, this is set to *True* when there is at least one associated accelerator whose state is *UP*. On Cray, set to *False* when all associated accelerators are in state *DOWN*. Used for requesting accelerators.

Format: *Boolean*

Python type: bool

accelerator_memory

Indicates amount of memory for accelerator(s) associated with this vnode. Host-level. Can be requested only inside of a select statement. On Cray, PBS sets this resource only on vnodes with at least one accelerator whose state is *UP*. For Cray, PBS sets this resource on the 0th NUMA node (the vnode with `PBScrayseg=0`), and the resource is shared by other vnodes on the compute node.

For example, on vnodeA_2_0:

```
resources_available.accelerator_memory=4196mb
```

On vnodeA_2_1:

```
resources_available.accelerator_memory=@vnodeA_2_0
```

Consumable.

Format: *size*

Python type: pbs.size

accelerator_model

Indicates model of accelerator(s) associated with this vnode. Host-level. On Cray, PBS sets this resource only on vnodes with at least one accelerator

whose state is *UP*. Can be requested only inside of a select statement. Non-consumable.

Format: *String*

Python type: str

naccelerators

Indicates number of accelerators on the host. Host-level. On Cray, should not be requested for jobs; PBS does not pass the request to ALPS. On Cray, PBS sets this resource only on vnodes whose hosts have at least one accelerator whose state is *UP*. PBS sets this resource to the number of accelerators whose state is *UP*. For Cray, PBS sets this resource on the 0th NUMA node (the vnode with `PBScrayseg=0`), and the resource is shared by other vnodes on the compute node.

For example, on vnodeA_2_0:

```
resources_available.naccelerators=1
```

On vnodeA_2_1:

```
resources_available.naccelerators=@vnodeA_2_0
```

Can be requested only inside of a select statement, but should not be requested.

Consumable.

Format: *Long*

Python type: int

nchunk

This is the number of chunks requested between plus symbols in a select statement. For example, if the select statement is `-lselect 4:ncpus=2+12:ncpus=8`, the value of `nchunk` for the first part is `4`, and for the second part it is `12`. The `nchunk` resource cannot be named in a select statement; it can only be specified as a number preceding the colon, as in the above example. When the number is omitted, `nchunk` is 1.

Non-consumable.

Settable by Manager and Operator; readable by all.

Format: *Integer*

Python type: int

Default value: *1*

11.3.1.2 Built-in Resources for the Cray

vntype

This resource represents the type of the vnode. Automatically set by PBS to one of two specific values for Cray vnodes. Has no meaning for non-Cray vnodes.

Non-consumable.

Format: *String array*

Automatically assigned values for Cray vnodes:

cray_compute

This vnode represents part of a compute node.

cray_login

This vnode represents a login node.

Default value: None

Python type: **str**

PBScrayhost

On CLE 2.2, this is set to “*default*”.

On CLE 3.0 and higher, used to delineate a Cray system, containing ALPS, login nodes running PBS MOMs, and compute nodes, from a separate Cray system with a separate ALPS. Non-consumable. The value of **PBScrayhost** is set to the value of **mpp_host** for this system.

Format: *String*

Default: CLE 2.2: “*default*”; CLE 3.0 and higher: None

PBScraylabel_<label name>

Custom resource created by PBS for the Cray. Tracks labels applied to compute nodes. For each label on a compute node, PBS creates a custom resource whose name is a concatenation of **PBScraylabel_** and the name of the label. PBS sets the value of the resource to *True* on all vnodes representing the compute node.

Format: *PBScraylabel_<label name>*

For example, if the label name is *Blue*, the name of this resource is **PBScraylabel_Blue**.

Format: *Boolean*

Default: None

PBScraynid

Custom resource created by PBS for the Cray. Used to track the node ID of the associated compute node. All vnodes representing a particular compute node share a value for **PBScraynid**. Non-consumable.

The value of **PBScraynid** is set to the value of `node_id` for this compute node.

Non-consumable.

Format: *String*

Default: None

PBScrayorder

Custom resource created by PBS for the Cray. Used to track the order in which compute nodes are listed in the Cray inventory. All vnodes associated with a particular compute node share a value for **PBScrayorder**. Non-consumable.

Vnodes for the first compute node listed are assigned a value of 1 for **PBScrayorder**. The vnodes for each subsequent compute node listed are assigned a value one greater than the previous value.

Do not use this resource in a resource request.

Format: *Integer*

Default: None

PBScrayseg

Custom resource created by PBS for the Cray. Tracks the segment ordinal of the associated NUMA node. For the first NUMA node of a compute host, the segment ordinal is 0, and the value of **PBScrayseg** for the associated vnode is 0. For the second NUMA node, the segment ordinal is 1, **PBScrayseg** is 1, and so on. Non-consumable.

Format: *String*

Default: None

11.3.2 Automatic Translation of mpp* Resource Requests

When a PBS job or reservation is submitted using the **mpp*** syntax, PBS translates the **mpp*** resource request into PBS select and place statements. The translation uses the following rules:

- For each chunk on a vnode representing a compute node, the **vntype** resource is set to *cray_compute*. (Using **mpp*** implies the use of compute nodes.)
- If the job requests `-lvnode=<value>`, the following becomes or is added to the

equivalent chunk request:

`:vnode=<value>`

- If the job requests `-lhost=<value>`, the following becomes or is added to the equivalent chunk request:

`:host=<value>`

- Translating `mppwidth`:

When the job requests `mppwidth`:

- If `mppnppn` is specified, the following happen:
 - `nchunk` (number of chunks) is set to $mppwidth / mppnppn$
 - `mpiprocs` is set to `mppnppn`
 - `-lplace=scatter` is added to the request
- If `mppnppn` is not specified, the following happen:
 - `mppnppn` is treated as if it is `1`
 - `nchunk` (number of chunks) is set to $mppwidth$
 - `-lplace=free` is added to the request

- Translating `mppnppn`:

If `mppnppn` is not specified, it defaults to `1`.

- Translating `mppdepth`:

If `mppdepth` is not specified, it defaults to `1`.

- `ncpus` is set to $mppdepth * mppnppn$
- If `mpphost` is specified as a submit argument, PBS adds a custom resource called `PBScrayhost` to the select statement, requesting the same value as for `mpphost`.
- The `mppnodes` resource is translated by PBS into the corresponding `vnodes`.
- When a job requests `mpplabels`, PBS adds a custom resource called `PBScraylabel_<label name>` to each chunk that requests a `vnode` from the compute node with that label. For example, if the job requests:

`-l mppwidth=1,mpplabels=\"small,red\"`

the translated request is:

`-l select=1: PBScraylabel_small=True:PBScraylabel_red=True`

- The following table summarizes how each `mpp*` resource is translated into select and

place statements:

Table 11-1: Mapping mpp* Resources to select and place

mpp* Resource	Resulting PBS Resource	How Value of PBS Resource is Derived
mpparch	arch	<i>arch=mpparch</i>
mppdepth*mppnppn (mppdepth defaults to 1 if not specified.)	ncpus	<i>ncpus = mppdepth*mppnppn</i>
mpphost	PBScrayhost	<i>PBScrayhost=mpphost</i>
mpplabels, for example mpplabels =\`red,small\`	PBScraylabel_red = <i>True</i> PBScraylabel_small = <i>True</i>	PBS creates custom Boolean resources named <i>PBScraylabel_<label></i> , and sets them to <i>True</i> on associated vnodes
mppmem	mem	<i>mem=mppmem</i>
mppnodes	Corresponding vnodes	PBS uses vnodes representing requested nodes
mppnppn (Defaults to 1 if not specified.)	mpiprocs	<i>mpiprocs=mppnppn</i>

Table 11-1: Mapping mpp* Resources to select and place

mpp* Resource		Resulting PBS Resource	How Value of PBS Resource is Derived
mppwidth	mppnppn specified	nchunk	$nchunk = mppwidth / mppnppn$ $place = scatter$ $mpiexecs = mppnppn$ Example: if mppwidth=8 and mppnppn=2, nchunk=4
	mppnppn not specified	nchunk	$nchunk = mppwidth$ $place = free$ $mpiexecs$ not set Example: if mppwidth=8, nchunk=8

11.3.2.1 Examples of Mapping mpp* Resources to select and place

Example 11-1: You want 8 PEs. The `aprun` statement is the following:

```
aprun -n 8
```

The old resource request using `mpp*` is the following:

```
qsub -l mppwidth=8
```

The translated select and place is the following:

```
qsub -lselect=8:vntype=cray_compute
```

Example 11-2: You want 8 PEs with only one PE per compute node. The `aprun` statement is the following:

```
aprun -n 8 -N 1
```

The old resource request using `mpp*` is the following:

```
qsub -l mppwidth=8, mppnppn=1
```

The translated select and place is the following:

```
qsub -lselect=8:ncpus=1:mpiprocs=1:vntype=cray_compute -lplace=scatter
```

Example 11-3: You want 8 PEs with 2 PEs per compute node. This equates to 4 chunks of 2 `ncpus` per chunk, scattered across different hosts. The `aprun` statement is the following:

```
aprun -n 8 -N 2
```

The old resource request using `mpp*` is the following:

```
qsub -l mppwidth=8, mppnppn=2
```

The translated select and place is the following:

```
qsub -lselect=4:ncpus=2:mpiprocs=2:vntype=cray_compute -lplace=scatter
```

Example 11-4: Specifying host:

The old resource request using `mpp*` is the following:

```
qsub -l mppwidth=8, mpphost=examplehost
```

The translated select and place is the following:

```
qsub -lselect=8:PBScrayhost=examplehost
```

Example 11-5: Specifying labels:

The old resource request using `mpp*` is the following:

```
-l mppwidth=1, mpplabels=\"small, red\"
```

The translated select and place is the following:

```
-l select=1: PBScraylabel_small=True:PBScraylabel_red=True
```

11.3.3 Resource Accounting

Jobs that request only compute nodes are not assigned resources from login nodes. PBS accounting logs do not show any login node resources being used by these jobs.

Jobs that request login nodes are assigned resources from login nodes, and those resources appear in the PBS accounting logs for these jobs.

PBS performs resource accounting on the login nodes, under the control of their MOMs.

Comprehensive System Accounting (CSA) runs on the compute nodes, under the control of the Cray system.

11.4 Rules for Submitting Jobs on the Cray

11.4.1 Always Specify Node Type

If you want your job to run on Cray nodes, you must specify a Cray node type for your job. You do this by requesting a value for the `vntype` vnode resource. On each vnode on a Cray, the `vntype` resource includes one of the following values:

cray_login, for a login node

cray_compute, for a compute node

Each chunk of a Cray job that must run on a login node must request a `vntype` of *cray_login*.

Each chunk of a Cray job that must run on a compute node must request a `vntype` of *cray_compute*.

Example 11-6: Request any login node, and two compute-node vnodes. The job is run on the login node selected by the scheduler:

```
qsub -lselect=1:ncpus=2:vntype=cray_login +2:ncpus=2:vntype=cray_compute
```

Example 11-7: Launch a job on a particular login node by specifying the login node vnode name first in the select line. The job script runs on the specified login node:

```
qsub -lselect=1:ncpus=2:vnode=login1 +2:ncpus=2:vntype=cray_compute
```

For a description of the `vntype` resource, see [“Built-in Resources” on page 299 of the PBS Professional Reference Guide](#).

11.4.2 Always Reserve Required Vnodes

Always reserve at least as many PEs as you request in your `aprun` statement.

11.4.3 Requesting Login Node Where Job Script Runs

If you request a login node as part of your resource request, the login node resource request must be the first element of the select statement. The job script is run on the login node. If you request more than one login node, the job script runs on the first login node requested.

11.4.4 Login Nodes in PBS Reservations

If the jobs that are to run in a PBS reservation require a particular login node, you must do the following:

- The reservation must request the specific login node
- Each job that will run in the reservation must request the same login node that the reservation requested

11.4.5 Specifying Number of Chunks

You specify the number of chunks by prefixing each chunk request with an integer. If not specified, this integer defaults to `1`. For example, to specify 4 chunks with 2 CPUs each, and 8 chunks with 1 CPU each:

```
qsub -lselect=4:ncpus=2+8:ncpus=1
```

You cannot request the `nchunk` resource.

If you request fewer chunks, the scheduling cycle is faster. See [section 11.7.10, “Request Fewer Chunks”](#), on page 297.

11.4.6 When Requesting Accelerators

PBS does not pass requests for the `naccelerators` resource to ALPS. To request accelerators for your job, use the `accelerator` resource, not the `naccelerators` resource.

For example, you want a total of 40 PEs with 4 PEs per compute node and one accelerator per compute node:

```
-lselect=10:ncpus=4:accelerator=True
```

See [section 11.5.11, “Requesting Accelerators”](#), on page 289.

11.4.7 Requesting mppnppn Equivalent

If your job requires the equivalent of mppnppn, you can do either of the following:

- When using select and place statements, use the translation information provided in [Table 11-1, “Mapping mpp* Resources to select and place,” on page 279](#), and include `-lplace=scatter` in the job request.
- Include mppnppn in the qsub line (mppnppn is deprecated.)

11.4.8 Do Not Mix mpp* and select/place

Jobs cannot use both `-lmpp*` syntax and `-lselect/-lplace` syntax.

11.4.9 Specify Host for Interactive Jobs

You can run an interactive job only on a login node. To ensure that your job runs on a login node, specify the host name. You can do so using a PBS directive, or the command line. For example:

```
qsub -lhost=<login node's resources_available.host name> -I job.sh
```

11.5 Techniques for Submitting Cray Jobs

11.5.1 Specifying Number of PEs per NUMA Node

The Cray `aprun -S` option allows you to specify the number of PEs per NUMA node for your job. PBS allows you to make the equivalent request using select and place statements. PBS jobs on the Cray should scatter chunks across vnodes.

To calculate the select and place requirements, do the following:

- Set `nchunk` equal to n (the width) divided by S (the number of PEs per NUMA node):

$$nchunk = n/S$$
- Set `ncpus` equal to S (the number PEs per NUMA node):

$$ncpus=S$$
- Set `mpiprocs` equal to S (same as `ncpus`)

mpiprocs=S

Example 11-8: You want a total of 6 PEs with 2 PEs per NUMA node. The `aprun` command is the following:

```
aprun -S 2 -n 6 myjob
```

The equivalent select and place statements are:

```
qsub -lselect=3:ncpus=2:mpiprocs=2 -lplace=vscatter
```

Given two compute nodes, each with two NUMA nodes, where each NUMA node has four PEs, two PEs from each of three of the NUMA nodes are assigned to the job.

Example 11-9: To request 8 PES, with 4 PEs per NUMA node, the `aprun` statement is the following:

```
aprun -S 4 -n 8
```

The equivalent select statement is the following, not including the scatter-by-vnode and exclusive-by-host placement language:

```
qsub -lselect=2:ncpus=4:mpiprocs=4
```

11.5.1.1 Caveats For `aprun -S`

When you use `aprun -S`, you must request `mpiprocs`, and request the same value as for `ncpus`.

11.5.2 Reserving *N* NUMA Nodes Per Compute Node

The Cray `aprun -sn` option allows you to specify the number of NUMA nodes per compute node for your job. PBS allows you to make the equivalent request using select and place statements.

To request *N* NUMA nodes per compute node, you place your job by requesting a resource that specifies the number of NUMA nodes per compute node. This resource is set up by your administrator. We suggest that the resource is named *craysn*, and the value you specify is the number of vnodes per compute node. For example, to request 2 segments per compute node, specify a value of 2 for *craysn*.

To make a request equivalent to `aprun -sn 3 -n 24`, and match the compute node exclusive behavior of the Cray, you can specify the following:

```
qsub -lselect=24:ncpus=1:craysn=3 -lplace=exclhost
```

11.5.3 Reserving Specific NUMA Nodes on Each Compute Node

The Cray `aprun -s1` option allows you to reserve specific NUMA nodes on the compute nodes your job uses. PBS allows you to make the equivalent request using `select` and `place` statements.

How you request resources depends on the number of NUMA nodes you want per compute node, and how the administrator has set up the resource that allows you to request specific compute nodes.

11.5.3.1 Requesting a Single NUMA Node Per Compute Node

You can request the `PBScrayseg` resource to request one particular NUMA node per compute node. PBS automatically creates a custom string resource called `PBScrayseg`, and sets the value for each vnode to be the segment ordinal for the associated NUMA node. See [“Custom Cray Resources” on page 306 of the PBS Professional Reference Guide](#).

Example 11-10: You want 8 PEs total, using only NUMA node 1 on each compute node. The `aprun` statement is the following:

```
aprun -s1 1 -n 8
```

An equivalent resource request for a PBS job is the following:

```
qsub -lselect=8:ncpus=1:PBScrayseg=1
```

See [section 11.3.1, “Built-in and Custom Resources for the Cray”](#), on page 274.

11.5.3.2 Requesting Multiple NUMA Nodes Per Compute Node

If you want to request multiple NUMA nodes per compute node, you have choices. For example, if your `aprun` statement looks like the following:

```
aprun -s1 0,1 -n 8
```

You can do any of the following:

- You can request separate chunks for each segment:

```
qsub -lselect=4:ncpus=1:PBScrayseg=0 +4:ncpus=1:PBScrayseg=1
```
- If you know about the underlying hardware, the PBS resource request can take advantage of that. On a homogenous system with 2 NUMA nodes per compute node and 4 PEs per NUMA node, you can use the following PBS resource request:

```
qsub -lselect=8:ncpus=1 -lplace=pack
```
- If the administrator has set up a resource that allows you to request NUMA node combi-

nations, called for example *segmentcombo*, you request a value for the resource that is the list of vnodes you want. The equivalent select statement which uses this resource is the following:

```
qsub -lselect=8:ncpus=1:segmentcombo=01 jobscript
```

11.5.3.3 Caveat When Using Combination or Number Resources

You **must** use the same resource string values as the ones set up by the administrator. “012” is **not** the same as “102” or “201”.

11.5.4 Requesting Groups of Login Nodes

If you want to use groups of esLogin nodes and internal login nodes, your administrator can set the *vntype* resource on these nodes to a special value, for example *cray_compile*.

To submit a job requesting any combination of esLogin nodes and internal login nodes, you specify the special value for the *vntype* resource in your select statement. For example:

```
qsub -lselect=4:ncpus=1:vntype=cray_compile job
```

11.5.5 Using Internal Login Nodes Only

Compiling, preprocessing, and postprocessing jobs can run on internal login nodes. Internal login nodes have a *vntype* value of *cray_login*. If you want to run a job that needs to use the resources on internal login nodes only, you can specify *vntype=cray_login* in your select statement. For example:

```
qsub -lselect=4:ncpus=1:vntype=cray_login job
```

11.5.6 Using Compute Nodes

If your job script contains an *aprun* launch, you must run your job on compute nodes. To run your job on compute nodes, specify a *vntype* of *cray_compute*. For example:

```
lselect=2:ncpus=2:vntype=cray_compute
```


11.5.7 Using Login and Compute Nodes

You can request both login and compute nodes for your job. You must specify the login node(s) before the compute nodes. You can specify a `vntype` of `cray_login` for the chunks requiring login nodes, and a `vntype` of `cray_compute` for the chunks requiring compute nodes. For example:

```
qsub -lselect=1:ncpus=2:vntype=cray_login +2:ncpus=2:vntype=cray_compute
```

11.5.8 Requesting Specific Groups of Nodes

You can use `select` and `place` to request the groups of vnodes you want. This replaces the behavior provided by `mppnodes`.

Users may need to group their nodes by the some criteria, for example:

- Certain nodes are fast nodes
- Certain nodes share a required or useful characteristic
- Some combination of nodes gives the best performance for an application

Your administrator can set up either of the following:

- Custom Boolean resources on each vnode, which reflect how the nodes are labeled, and allow you to request the vnodes that represent the group of nodes you want. These resources are named `PBScraylabel_<label name>`, and set to `True` on the vnodes that represent the labeled nodes.

Your administrator must label the groups of nodes. For example, if a node is both fast and best for App1, it can have two labels, *fast*, and *BestForApp1*.

To request the fast nodes in this example, add the following to each chunk request:

```
:PBScraylabel_fast=True
```

- Other custom resources on each vnode, which are set to reflect the vnode's characteristics. For example, if a vnode is fast, it can have a custom string resource called "*speed*", with a value of *fast* on that vnode. You must ask your administrator for the name and possible values for the resource.

11.5.9 Requesting Nodes in Specific Order

Your application may perform better when the ranks are laid out on specific nodes in a specific order. If you want to request vnodes so that the nodes are in a specific order, you can specify the host for each chunk of the job. For example, if you need nodes ordered “nid0, nid2, nid4”, you can request the following:

```
qsub -l select=2:ncpus=2:host=nid0 +2:ncpus=2:host=nid2
      +2:ncpus=2:host=nid4
```

11.5.10 Requesting Interlagos Hardware

PBS allows you to specifically request (or avoid) Interlagos hardware. Your administrator must create a Boolean resource on each vnode, and set it to *True* where the vnode has Interlagos hardware. We recommend that the Boolean is called “*PBScraylabel_interlagos*”.

You request or avoid this resource using *PBScraylabel_interlagos=True* or *PBScraylabel_interlagos=False*. For example:

```
qsub -lselect=3:ncpus=2:PBScraylabel_interlagos=true myjob
```

11.5.11 Requesting Accelerators

Accelerators are associated with vnodes when those vnodes represent NUMA nodes on a host that has at least one accelerator in state *UP*. PBS allows you to request vnodes with associated accelerators. PBS sets the Boolean host-level resource *accelerator* to *True* on vnodes that have an associated accelerator. To request a vnode with an associated accelerator, include the following in the job’s select statement:

```
accelerator = True
```

11.5.11.1 Examples of Requesting Accelerators

Example 11-11: You want 30PEs and a Tesla_x2090 accelerator on each host, and the accelerator should have at least 4000MB, and you don't care how many hosts the job uses:

```
-lselect=30:ncpus=1:accelerators=True:accelerator_model="Tesla_x2090"  
:accelerator_memory=4000MB myjob
```

Example 11-12: You want a total of 40 PEs with 4 PEs per compute node and one accelerator per compute node:

```
-lselect=10:ncpus=4:accelerator=True
```

Example 11-13: Your system has some compute nodes with one type of accelerator (GPU1), and another type of compute node with a different type of accelerator (GPU2), and you want to request 10 PEs and 1 accelerator of model “GPU1” per compute node and 4 PEs and 1 accelerator of model “GPU2” per compute node. Your job request would look like this:

```
-lselect=10:ncpus=1:accelerator=True:accelerator_model="GPU1"  
+4:ncpus=1:accelerator=True:accelerator_model="GPU2" myjob
```

Do not request the `naccelerators` resource. This resource request is not passed to ALPS.

11.6 Viewing Cray Job Information

11.6.1 Finding Out Where Job Was Launched

To determine the internal login node where the job was launched, use the `qstat -f` command:

```
qstat -f <job ID>
```

Look at the `exec_host` line of the output. The first vnode is the login node where the job was launched.

11.6.2 Finding Out How mpp* Request Was Translated

- To find out how the `mpp*` job request was translated into select and place statements, use

the `qstat -f` command:

`qstat -f[x] <job ID>`

Look at the `Resource_List.select` job attribute. The original is in the `Submit_arguments` job attribute.

- To find out how the `mpp*` reservation request was translated into select and place statements, use the `pbs_rstat` command:

`pbs_rstat -F <reservation ID>`

Look at the `Resource_List` attribute.

11.6.3 Viewing Original mpp* Request

To see the original `mpp*` request, use the `qstat` command:

`qstat -f[x] <job ID>`

The `Submit_arguments` field contains the original `mpp*` request.

11.6.4 Listing Jobs Running on Vnode

To see which jobs are running on a vnode, use the `pbsnodes` command:

`pbsnodes -av`

The `jobs` attribute of each vnode lists the jobs running on that vnode. Jobs launched from an internal login node, requesting a `vntype` of *cray_compute* only, are not listed in the internal login node's vnode's `jobs` attribute. Jobs that are actually running on a login node, which requested a `vntype` of *cray_login*, do appear in the login node's vnode's `jobs` attribute.

11.6.4.1 Caveats When Listing Jobs

Jobs that requested a `vntype` of *cray_compute* that were launched from an internal login node are not listed in the `jobs` attribute of the internal login node.

11.6.4.2 Example Output

Example of `pbsnodes -av` output for segments 0 and 1 on the same compute node:

```
examplehost_8_0
  Mom = exampleMom
  ntype = PBS
  state = free
  pcpus = 6
  resources_available.accelerator = True
  resources_available.accelerator_memory = 4096mb
  resources_available.accelerator_model = Tesla_x2090
  resources_available.arch = XT
  resources_available.host = examplehost_8
  resources_available.mem = 8192000kb
  resources_available.naccelerators = 1
  resources_available.ncpus = 6
  resources_available.PBScrayhost = examplehost
  resources_available.PBScraynid = 8
  resources_available.PBScrayorder = 1
  resources_available.PBScrayseg = 0
  resources_available.vnode = examplehost_8_0
  resources_available.vntype = cray_compute
  resources_assigned.accelerator_memory = 0kb
  resources_assigned.mem = 0kb
  resources_assigned.mem = 0kb
  resources_assigned.naccelerators = 0
  resources_assigned.ncpus = 0
  resources_assigned.netwins = 0
  resources_assigned.vmem = 0kb
  resv_enable = True
  sharing = force_exclhost

examplehost_8_1
  Mom = exampleMom
  ntype = PBS
  state = free
```

```

pcpus = 6
resources_available.accelerator = True
resources_available.accelerator_memory = @examplehost_8_0
resources_available.accelerator_model = Tesla_x2090
resources_available.arch = XT
resources_available.host = examplehost_8
resources_available.mem = 8192000kb
resources_available.naccelerators = @examplehost_8_0
resources_available.ncpus = 6
resources_available.PBScrayhost = examplehost
resources_available.PBScraynid = 8
resources_available.PBScrayorder = 1
resources_available.PBScrayseg = 1
resources_available.vnode = examplehost_8_1
resources_available.vntype = cray_compute
resources_assigned.accelerator_memory = @examplehost_8_0
resources_assigned.mem = 0kb
resources_assigned.naccelerators = @examplehost_8_0
resources_assigned.ncpus = 0
resources_assigned.netwins = 0
resources_assigned.vmem = 0kb
resv_enable = True
sharing = force_exclhost

```

11.6.5 How ALPS Request Is Constructed

The reservation request that is sent to the Cray is constructed from the contents of the `exec_vnode` and `Resource_List.select` job attributes. If the `exec_vnode` attribute contains chunks asking for the same `ncpus` and `mem`, these are grouped into one section of an ALPS request. Cray requires one CPU per thread. The ALPS request is constructed using the following rules:

Table 11-2: How Cray Elements Are Derived From `exec_vnode` Terms

Cray Element	<code>exec_vnode</code> Term
Processing Element (PE)	<code>mpiprocs</code>

Table 11-2: How Cray Elements Are Derived From `exec_vnode` Terms

Cray Element	<code>exec_vnode</code> Term
Requested number of PEs / compute node in this section of job request (width)	Total <code>mpiprocs</code> on <code>vnodes</code> representing compute node involved in this section of job request
Number of threads per PE (depth)	(total assigned <code>ncpus</code> on <code>vnodes</code> representing a compute node) / (total <code>mpiprocs</code> on <code>vnodes</code> representing a compute node)
Memory per PE (<code>mem</code>)	(total memory in chunk request)/total <code>mpiprocs</code> in chunk
Number of PEs per compute node (<code>nppn</code>)	Sum of <code>mpiprocs</code> on <code>vnodes</code> representing a compute node
Number of PEs per segment (<code>npps</code>)	Not used.
Number of segments per node (<code>nspn</code>)	Not used.
NUMA node (segments)	Not used.

11.6.6 Viewing Accelerator Information

There is no `aprun` interface for requesting accelerator memory or model, so this information is not translated into Cray elements. To see this information, look in the MOM logs for the job's login node.

11.7 Caveats and Advice

11.7.1 Use `select` and `place` Instead of `mpp*`

It is recommended to use `select` and `place` instead of `mpp*` resources. The `mpp*` resources are deprecated.

11.7.2 Using Combination or Number Resources

When requesting a resource that is set up by the administrator, you **must** use the same resource string values as the ones set up by the administrator. “012” is **not** the same as “102” or “201”. For example, when requesting a resource that allows you to request NUMA nodes 0 and 1, and the administrator used the string *01*, you must request `<resource name>=01`. If you request `<resource name>=10`, this will not work.

11.7.3 Avoid Invalid Cray Requests

It is possible to create a select and place statement that meets the requirements of PBS but not of the Cray.

Example 11-14: The Cray width and depth values cannot be calculated from `ncpus` and `mpiprocs` values. For example, if `ncpus` is 2 and `mpiprocs` is 4, the `depth` value is calculated by dividing `ncpus` by `mpiprocs`, and is one-half. This is not a valid `depth` value for Cray.

Example 11-15: ALPS cannot run jobs with some complex select statements. In particular, a multiple program, multiple data (MPMD) ALPS reservation where two groups span a compute node will produce an ALPS error, because the `nid` shows up in two Reserve-Param sections.

11.7.4 Visibility of Jobs Launched from Login Nodes

Jobs that requested a `vntype` of *cray_compute* that were launched from an internal login node are not listed in the jobs attribute of the internal login node.

11.7.5 Resource Restrictions and Deprecations

11.7.5.1 Restriction on Translation of mpp* Resources

PBS translates only the following mpp* resources into select and place syntax:

- mppwidth
- mppdepth
- mppnppn
- mppmem
- mpparch
- mpphost
- mpplabels
- mppnodes

11.7.5.2 mpp* Resources Deprecated

The mpp* syntax is deprecated. See [section 1.3, "Deprecations and Removals" on page 8 in the PBS Professional Administrator's Guide](#).

11.7.6 Do Not Mix mpp* and select/place

Jobs cannot use both -lmpp* syntax and -lselect/-lplace syntax.

11.7.7 Do Not Request PBScrayorder

Do not use PBScrayorder in a resource request.

11.7.8 Do Not Request naccelerators

Do not use naccelerators in a resource request. See [section 11.5.11, "Requesting Accelerators", on page 289](#).

11.7.9 Do Not Suspend Jobs

Do **not** attempt to use `qsig -s suspend` on the Cray. Attempting to suspend a job on the Cray will cause errors.

11.7.10 Request Fewer Chunks

The more chunks in each translated job request, the longer the scheduling cycle takes. Jobs that request a value for `mppnppn` or `ncpus` effectively direct PBS to use the size of `mppnppn` or `ncpus` as the value for `ncpus` for each chunk, thus dividing the number of chunks by `mppnppn` or `ncpus`.

If you are on a homogeneous system, we recommend that chunks use the value for `ncpus` for a vnode or for a compute node.

Example 11-16: Comparison of larger vs. smaller chunk size and the effect on scheduling time:

Submit job with chunk size 1 and 8544 chunks:

```
qsub -lmpwidth=8544 job
```

Job's Resource_List:

```
Resource_List.mppwidth = 8544
```

```
Resource_List.ncpus = 8544
```

```
Resource_List.place = free
```

```
Resource_List.select = 8544:vntype=cray_compute
```

```
Submit_arguments = -lmpwidth=8544 job
```

Scheduling took 6 seconds:

```
12/05/2011 16:46:10;0080;pbs_sched;Job;23.example;considering job to run
```

```
12/05/2011 16:46:16;0040;pbs_sched;Job;23.example;Job run
```

Submit job with chunk size 8 and 1068 chunks:

```
qsub -lmpwidth=8544,mppnppn=8 job
```

Job's Resource_List:

```
Resource_List.mpi_procs = 8544
```

```
Resource_List.mppnppn = 8
```

```
Resource_List.mppwidth = 8544
```

```
Resource_List.ncpus = 8544
```

```
Resource_List.place = scatter
```

```
Resource_List.select = 1068:ncpus=8:mpi_procs=8:vntype=cray_compute
```

Scheduling took 1 second:

```
12/05/2011 16:54:38;0080;pbs_sched;Job;24.example;Considering job to run
12/05/2011 16:54:39;0040;pbs_sched;Job;24.example;Job run
```

If you are on a heterogeneous system, with varying sizes for vnodes or compute nodes, you can request chunk sizes that fit available hardware, but this may not be feasible.

11.8 Errors and Logging

11.8.1 Invalid Cray Requests

When a select statement does not meet Cray requirements, and the Cray reservation fails, the following error message is printed in MOM's log, at log event class 0x080:

```
"Fatal MPP reservation error preparing request"
```

11.8.2 Job Requests More Than Available

If `do_not_span_psets` is set to *True*, and a job requests more resources than are available in one placement set, the following happens:

- The job's comment is set to the following:

```
"Not Running: can't fit in the largest placement set, and can't span psets"
```
- The following message is printed to the scheduler's log:

```
"Can't fit in the largest placement set, and can't span placement sets"
```

11.8.3 All Requested mppnodes Not Found

If `mppnodes` are requested, but there are no vnodes that match the requested `mppnodes` (i.e. 0% of the `mppnodes` list is found), the job or reservation is rejected with the following message:

```
"The following error was encountered: No matching vnodes for the given
mppnodes <mppnodes>"
```

A log message is printed to the server log at event class 0x0004:

```
"translate mpp: ERROR: could not find matching vnodes for the given
mppnodes <mppnodes (as input)>"
```

11.8.4 Some Requested mppnodes Not Found

If mppnodes are requested, and only some of the mppnodes are found to match the vnodes, then the job or reservation is accepted, but the following is printed in the server log at event class 0x0004:

```
"translate mpp: could not find matching vnodes for these given mppnodes
  [<comma separated list of mppnodes>]"
```

The job may or may not run depending on whether the vnodes that were matched up to the requested mppnodes have enough resources for the job.

11.8.5 Bad mppnodes Range

If the resource request specifies an mppnodes range with the value on the right hand side of the range less than or equal to the value on the left hand side of the range, the job or reservation is rejected with the following message:

The following error was encountered:

```
Bad range '<range>', the first number (<left_side>) must be less than the
second number (<right_side>)
```

A log message is printed to the server log at event class 0x0004:

```
"translate mpp: ERROR: bad range '<range>', the first number (<left_side>)
must be less than the second number (<right_side>)"
```

11.8.6 Resource Request Containing Both mpp* and select/place

If a resource request contains both mpp* and select/place, the job or reservation is rejected, and the following error is printed:

The following error was encountered:

```
mpp resources cannot be used with "select" or "place"
```


Chapter 12

Using Provisioning

PBS provides automatic provisioning of an OS or application on vnodes that are configured to be provisioned. When a job requires an OS that is available but not running, or an application that is not installed, PBS provisions the vnode with that OS or application.

12.1 Definitions

AOE

The environment on a vnode. This may be one that results from provisioning that vnode, or one that is already in place

Provision

To install an OS or application, or to run a script which performs installation and/or setup

Provisioned Vnode

A vnode which, through the process of provisioning, has an OS or application that was installed, or which has had a script run on it

12.2 How Provisioning Works

Provisioning can be performed only on vnodes that have provisioning enabled, shown in the vnode's `provision_enable` attribute.

Provisioning can be the following:

- Directly installing an OS or application
- Running a script which may perform setup or installation

Each vnode is individually configured for provisioning with a list of available AOE's, in the vnode's `resources_available.aoe` attribute.

Each vnode's `current_aoe` attribute shows that vnode's current AOE. The scheduler queries each vnode's `aoe` resource and `current_aoe` attribute in order to determine which vnodes to provision for each job.

Provisioning can be used for interactive jobs.

A job's `walltime` clock starts when provisioning for the job has finished.

12.2.1 Causing Vnodes To Be Provisioned

An AOE can be requested for a job or a reservation. When a job requests an AOE, that means that the job will be run on vnodes running that AOE. When a reservation requests an AOE, that means that the reservation reserves vnodes that have that AOE available. The AOE is instantiated on reserved vnodes only when a job requesting that AOE runs.

When the scheduler runs each job that requests an AOE, it either finds the vnodes that satisfy the job's requirements, or provisions the required vnodes. For example, if SLES is available on a set of vnodes that otherwise suit your job, you can request SLES for your job, and regardless of the OS running on those vnodes before your job starts, SLES will be running at the time the job begins execution.

12.2.2 Using an AOE

When you request an AOE for a job, the requested AOE must be one of the AOE's that has been configured at your site. For example, if the AOE's available on vnodes are "*rhel*" and "*sles*", you can request only those; you cannot request "*suse*".

You can request a reservation for vnodes that have a specific AOE available. This way, jobs needing that AOE can be submitted to that reservation. This means that jobs needing that AOE are guaranteed to be running on vnodes that have that AOE available.

Each reservation can have at most one AOE specified for it. Any jobs that run in that reservation must not request a different AOE from the one requested for the reservation. That is, the job can run in the reservation if it either requests no AOE, or requests the same AOE as the reservation.

12.2.3 Job Substates and Provisioning

When a job is in the process of provisioning, its substate is *provisioning*. This is the description of the substate:

provisioning

The job is waiting for vnode(s) to be provisioned with its requested AOE. Integer value is 71. See [“Job Substates” on page 412 of the PBS Professional Reference Guide](#) for a list of job substates.

The following table shows how provisioning events affect job states and substates:

Table 12-1: Provisioning Events and Job States/Substates

Event	Initial Job State, Substate	Resulting Job State, Substate
Job submitted		<i>Queued and ready for selection</i>
Provisioning starts	<i>Queued, Queued</i>	<i>Running, Provisioning</i>
Provisioning fails to start	<i>Queued, Queued</i>	<i>Held, Held</i>
Provisioning fails	<i>Running, Provisioning</i>	<i>Queued, Queued</i>
Provisioning succeeds and job runs	<i>Running, Provisioning</i>	<i>Running, Running</i>
Internal error occurs	<i>Running, Provisioning</i>	<i>Held, Held</i>

12.3 Requirements and Restrictions

12.3.1 Host Restrictions

12.3.1.1 Single-vnode Hosts Only

PBS will provision only single-vnode hosts. Do not attempt to use provisioning on hosts that have more than one vnode.

12.3.1.2 Server Host Cannot Be Provisioned

The Server host cannot be provisioned: a MOM can run on the Server host, but that MOM's vnode cannot be provisioned. The `provision_enable` vnode attribute, `resources_available.aoe`, and `current_aoe` cannot be set on the Server host.

12.3.2 AOE Restrictions

Only one AOE can be instantiated at a time on a vnode.

Only one kind of `aoe` resource can be requested in a job. For example, an acceptable job could make the following request:

```
-l select=1:ncpus=1:aoe=suse+1:ncpus=2:aoe=suse
```

12.3.2.1 Vnode Job Restrictions

A vnode with any of the following jobs will not be selected for provisioning:

- One or more running jobs
- A suspended job
- A job being backfilled around

12.3.2.2 Provisioning Job Restrictions

A job that requests an AOE will not be backfilled around.

12.3.2.3 Vnode Reservation Restrictions

A vnode will not be selected for provisioning for job MyJob if the vnode has a confirmed reservation, and the start time of the reservation is before job MyJob will end.

A vnode will not be selected for provisioning for a job in reservation R1 if the vnode has a confirmed reservation R2, and an occurrence of R1 and an occurrence of R2 overlap in time and share a vnode for which different AOE's are requested by the two occurrences.

12.3.3 Requirements for Jobs

12.3.3.1 If AOE is Requested, All Chunks Must Request Same AOE

If any chunk of a job requests an AOE, all chunks must request that AOE.

If a job requesting an AOE is submitted to a reservation, that reservation must also request the same AOE.

12.4 Using Provisioning

12.4.1 Requesting Provisioning

You request a reservation with an AOE in order to reserve the resources and AOE required to run a job. You request an AOE for a job if that job requires that AOE. You request provisioning for a job or reservation using the same syntax.

You can request an AOE for the entire job/reservation:

```
-l aoe = <AOE>
```

Example:

```
-l aoe = suse
```

The `-l <AOE>` form cannot be used with `-l select`.

You can request an AOE for a single-chunk job/reservation:

```
-l select=<chunk request>:aoe=<AOE>
```

Example:

```
-ls select=1:ncpus=2:aoe=rhel
```

You can request the same AOE for each chunk of a job/reservation:

```
-l select=<chunk request>:aoe=<AOE> + <chunk request>:aoe=<AOE>
```

Example:

```
-l select=1:ncpus=1:aoe=suse + 2:ncpus=2:aoe=suse
```

12.4.2 Commands and Provisioning

If you try to use PBS commands on a job that is in the *provisioning* substate, the commands behave differently. The provisioning of vnodes is not affected by the commands; if provisioning has already started, it will continue. The following table lists the affected commands:

Table 12-2: Effect of Commands on Jobs in Provisioning Substate

Command	Behavior While in Provisioning Substate
qdel	(Without force) Job is not deleted
	(With force) Job is deleted

Table 12-2: Effect of Commands on Jobs in Provisioning Substate

Command	Behavior While in Provisioning Substate
<code>qsig -s suspend</code>	Job is not suspended
<code>qhold</code>	Job is not held
<code>qrerun</code>	Job is not queued
<code>qmove</code>	Cannot be used on a job that is provisioning
<code>qalter</code>	Cannot be used on a job that is provisioning
<code>qrun</code>	Cannot be used on a job that is provisioning

12.4.3 How Provisioning Affects Jobs

A job that has requested an AOE will not preempt another job. Therefore no job will be terminated in order to run a job with a requested AOE.

A job that has requested an AOE will not be backfilled around.

12.5 Caveats and Errors

12.5.1 Requested Job AOE and Reservation AOE Should Match

Do not submit jobs that request an AOE to a reservation that does not request the same AOE. Reserved vnodes may not supply that AOE; your job will not run.

12.5.2 Allow Enough Time in Reservations

If a job is submitted to a reservation with a duration close to the walltime of the job, provisioning could cause the job to be terminated before it finishes running, or to be prevented from starting. If a reservation is designed to take jobs requesting an AOE, leave enough extra time in the reservation for provisioning.

12.5.3 Requesting Multiple AOE's For a Job or Reservation

Do not request more than one AOE per job or reservation. The job will not run, or the reservation will remain unconfirmed.

12.5.4 Held and Requeued Jobs

The job is held with a system hold for the following reasons:

- Provisioning fails due to invalid provisioning request or to internal system error
- After provisioning, the AOE reported by the vnode does not match the AOE requested by the job

The hold can be released by the PBS Administrator after investigating what went wrong and correcting the mistake.

The job is requeued for the following reasons:

- The provisioning hook fails due to timeout
- The vnode is not reported back up

12.5.5 Conflicting Resource Requests

The values of the resources `arch` and `vnode` may be changed by provisioning. Do not request an AOE and either `arch` or `vnode` for the same job.

12.5.6 Job Submission and Alteration Have Same Requirements

Whether you use the `qsub` command to submit a job, or the `qalter` command to alter a job, the job must eventually meet the same requirements. You cannot submit a job that meets the requirements, then alter it so that it does not.

Appendix A: Converting NQS to PBS

For those converting to PBS from NQS or NQE, PBS includes a utility called **nqs2pbs** which converts an existing NQS job script so that it will work with PBS. (In fact, the resulting script will be valid to both NQS and PBS.) The existing script is copied and PBS directives (“#PBS”) are inserted prior to each NQS directive (either “#QSUB” or “#Q\$”) in the original script.

```
nqs2pbs existing-NQS-script new-PBS-script
```

Section ["Setting Up Your UNIX/Linux Environment" on page 13](#) discusses PBS environment variables.

A queue complex in NQS was a grouping of queues within a batch Server. The purpose of a complex was to provide additional control over resource usage. The advanced scheduling features of PBS eliminate the requirement for queue complexes.

13.1 Converting Date Specifications

Converting NQS date specifications to the PBS form may result in a warning message and an incomplete converted date. PBS does not support date specifications of “today”, “tomorrow”, or the name of the days of the week such as “Monday”. If any of these are encountered in a script, the PBS specification will contain only the time portion of the NQS specification (i.e. #PBS -a hhmm[.ss]). It is suggested that you specify the execution time on the `qsub` command line rather than in the script. All times are taken as local time. If any unrecognizable NQS directives are encountered, an error message is displayed. The new PBS script will be deleted if any errors occur.

Appendix B: License Agreement

CAUTION!

PRIOR TO INSTALLATION OR USE OF THE SOFTWARE YOU MUST CONSENT TO THE FOLLOWING SOFTWARE LICENSE TERMS AND CONDITIONS BY CLICKING THE “I ACCEPT” BUTTON BELOW. YOUR ACCEPTANCE CREATES A BINDING LEGAL AGREEMENT BETWEEN YOU AND ALTAIR. IF YOU DO NOT HAVE THE AUTHORITY TO BIND YOUR ORGANIZATION TO THESE TERMS AND CONDITIONS, YOU MUST CLICK “I DO NOT ACCEPT” AND THEN HAVE AN AUTHORIZED PARTY IN THE ORGANIZATION THAT YOU REPRESENT ACCEPT THESE TERMS.

IF YOU, OR THE ORGANIZATION THAT YOU REPRESENT, HAS A MASTER SOFTWARE LICENSE AGREEMENT (“MASTER SLA”) ON FILE AT THE CORPORATE HEADQUARTERS OF ALTAIR ENGINEERING, INC. (“ALTAIR”), THE MASTER SLA TAKES PRECEDENCE OVER THESE TERMS AND SHALL GOVERN YOUR USE OF THE SOFTWARE.

MODIFICATION(S) OF THESE SOFTWARE LICENSE TERMS IS EXPRESSLY PROHIBITED. ANY ATTEMPTED MODIFICATION(S) WILL BE NONBINDING AND OF NO FORCE OR EFFECT UNLESS EXPRESSLY AGREED TO IN WRITING BY AN AUTHORIZED CORPORATE OFFICER OF ALTAIR. ANY DISPUTE RELATING TO THE VALIDITY OF AN ALLEGED MODIFICATION SHALL BE DETERMINED IN ALTAIR’S SOLE DISCRETION.

Altair Engineering, Inc. - Software License Agreement

THIS SOFTWARE LICENSE AGREEMENT, including any Additional Terms (together with the “Agreement”), shall be effective as of the date of YOUR acceptance of these software license terms and conditions (the “Effective Date”) and is between ALTAIR ENGINEERING, INC., 1820 E. Big Beaver Road, Troy, MI 48083-2031, USA, a Michigan corporation (“Altair”), and YOU, or the organization on whose behalf you have authority to accept these terms (the “Licensee”). Altair and Licensee, intending to be legally bound, hereby agree as follows:

1. DEFINITIONS. In addition to terms defined elsewhere in this Agreement, the following terms shall have the meanings defined below for purposes of this Agreement:

Additional Terms. Additional Terms are those terms and conditions which are determined by an Altair Subsidiary to meet local market conditions.

Documentation. Documentation provided by Altair or its resellers on any media for use with the Products.

Execute. To load Software into a computer's RAM or other primary memory for execution by the computer.

Global Zone: Software is licensed based on three Global Zones: the Americas, Europe and Asia-Pacific. When Licensee has Licensed Workstations located in multiple Global Zones, which are connected to a single License (Network) Server, a premium is applied to the standard Software License pricing for a single Global Zone.

ISV/Independent Software Vendor. A software company providing its products, (“ISV Software”) to Altair's Licensees through the Altair License Management System using Altair License Units.

License Log File. A computer file providing usage information on the Software as gathered by the Software.

License Management System. The license management system (LMS) that accompanies the Software and limits its use in accordance with this Agreement, and which includes a License Log File.

License (Network) Server. A network file server that Licensee owns or leases located on Licensee's premises and identified by machine serial number and/or HostID on the Order Form.

License Units. A parameter used by the LMS to determine usage of the Software permitted under this Agreement at any one time.

Licensed Workstations. Single-user computers located in the same Global Zone(s) that Licensee owns or leases that are connected to the License (Network) Server via local area network or Licensee's private wide-area network.

Maintenance Release. Any release of the Products made generally available by Altair to its Licensees with annual leases, or those with perpetual licenses who have an active maintenance agreement in effect, that corrects programming errors or makes other minor changes to the Software. The fees for maintenance and support services are included in the annual license fee but perpetual licenses require a separate fee.

Order Form. Altair's standard form in either hard copy or electronic format that contains the specific parameters (such as identifying Licensee's contracting office, License Fees, Software, Support, and License (Network) Servers) of the transaction governed by this Agreement.

Products. Products include Altair Software, ISV Software, and/or Suppliers' software; and Documentation related to all of the forgoing.

Proprietary Rights Notices. Patent, copyright, trademark or other proprietary rights notices applied to the Products, packaging or media.

Software. The Altair software identified in the Order Form and any Updates or Maintenance Releases.

Subsidiary. Subsidiary means any partnership, joint venture, corporation or other form of enterprise in which a party possesses, directly or indirectly, an ownership interest of fifty percent (50%) or greater, or managerial or operational control.

Suppliers. Any person, corporation or other legal entity which may provide software or documents which are included in the Software.

Support. The maintenance and support services provided by Altair pursuant to this Agreement.

Templates. Human readable ASCII files containing machine-interpretable commands for use with the Software.

Term. The term of licenses granted under this Agreement. Annual licenses shall have a 12-month term of use unless stated otherwise on the Order Form. Perpetual licenses shall have a term of twenty-five years. Maintenance agreements for perpetual licenses have a 12-month term.

Update. A new version of the Products made generally available by Altair to its Licensees that includes additional features or functionalities but is substantially the same computer code as the existing Products.

2. LICENSE GRANT. Subject to the terms and conditions set forth in this Agreement, Altair hereby grants Licensee, and Licensee hereby accepts, a limited, non-exclusive, non-transferable license to: a) install the Products on the License (Network) Server(s) identified on the Order Form for use only at the sites identified on the Order Form; b) execute the Products on Licensed Workstations in accordance with the LMS for use solely by Licensee's employees, or its onsite Contractors who have agreed to be bound by the terms of this Agreement, for Licensee's internal business use on Licensed Workstations within the Global Zone(s) as iden-

tified on the Order Form and for the term identified on the Order Form; c) make backup copies of the Products, provided that Altair's and its Suppliers' and ISV's Proprietary Rights Notices are reproduced on each such backup copy; d) freely modify and use Templates, and create interfaces to Licensee's proprietary software for internal use only using APIs provided that such modifications shall not be subject to Altair's warranties, indemnities, support or other Altair obligations under this Agreement; and e) copy and distribute Documentation inside Licensee's organization exclusively for use by Licensee's employees and its onsite Contractors who have agreed to be bound by the terms of this Agreement. A copy of the License Log File shall be made available to Altair automatically on no less than a monthly basis. In the event that Licensee uses a third party vendor for information technology (IT) support, the IT company shall be permitted to access the Software only upon its agreement to abide by the terms of this Agreement. Licensee shall indemnify, defend and hold harmless Altair for the actions of its IT vendor(s).

3. RESTRICTIONS ON USE. Notwithstanding the foregoing license grant, Licensee shall not do (or allow others to do) any of the following: a) install, use, copy, modify, merge, or transfer copies of the Products, except as expressly authorized in this Agreement; b) use any back-up copies of the Products for any purpose other than to replace the original copy provided by Altair in the event it is destroyed or damaged; c) disassemble, decompile or “unlock”, reverse translate, reverse engineer, or in any manner decode the Software or ISV Software for any reason; d) sublicense, sell, lend, assign, rent, distribute, publicly display or publicly perform the Products or Licensee's rights under this Agreement; e) allow use outside the Global Zone(s) or User Sites identified on the Order Form; f) allow third parties to access or use the Products such as through a service bureau, wide area network, Internet location or time-sharing arrangement except as expressly provided in Section 2(b); g) remove any Proprietary Rights Notices from the Products; h) disable or circumvent the LMS provided with the Products; or (i) link any software developed, tested or supported by Licensee or third parties to the Products (except for Licensee's own proprietary software solely for Licensee's internal use).

4. OWNERSHIP AND CONFIDENTIALITY. Licensee acknowledges that all applicable rights in patents, copyrights, trademarks, service marks, and trade secrets embodied in the Products are owned by Altair and/or its Suppliers or ISVs. Licensee further acknowledges that the Products, and all copies thereof, are and shall remain the sole and exclusive property of Altair and/or its Suppliers and ISVs. This Agreement is a license and not a sale of the Products. Altair retains all rights in the Products not expressly granted to Licensee herein. Licensee acknowledges that the Products are confidential and constitute valuable assets and trade secrets of Altair and/or its Suppliers and ISVs. Licensee agrees to take the same precautions necessary to protect and maintain the confidentiality of the Products as it does to protect its own information of a confidential nature but in any event, no less than a reasonable degree of care, and shall not disclose or make them available to any person or entity except as expressly provided in this Agreement. Licensee shall promptly notify Altair in the event any unauthorized person obtains access to the Products. If Licensee is required by any governmental

authority or court of law to disclose Altair's or its ISV's or its Suppliers' confidential information, then Licensee shall immediately notify Altair before making such disclosure so that Altair may seek a protective order or other appropriate relief. Licensee's obligations set forth in Section 3 and Section 4 of this Agreement shall survive termination of this Agreement for any reason. Altair's Suppliers and ISVs, as third party beneficiaries, shall be entitled to enforce the terms of this Agreement directly against Licensee as necessary to protect Supplier's intellectual property or other rights.

Altair and its resellers providing support and training to licensed end users of the Products shall keep confidential all Licensee information provided to Altair in order that Altair may provide Support and training to Licensee. Licensee information shall be used only for the purpose of assisting Licensee in its use of the licensed Products. Altair agrees to take the same precautions necessary to protect and maintain the confidentiality of the Licensee information as it does to protect its own information of a confidential nature but in any event, no less than a reasonable degree of care, and shall not disclose or make them available to any person or entity except as expressly provided in this Agreement.

5. MAINTENANCE AND SUPPORT. **Maintenance.** Altair will provide Licensee, at no additional charge for annual licenses and for a maintenance fee for paid-up licenses, with Maintenance Releases and Updates of the Products that are generally released by Altair during the term of the licenses granted under this Agreement, except that this shall not apply to any Term or Renewal Term for which full payment has not been received. Altair does not promise that there will be a certain number of Updates (or any Updates) during a particular year. If there is any question or dispute as to whether a particular release is a Maintenance Release, an Update or a new product, the categorization of the release as determined by Altair shall be final. Licensee agrees to install Maintenance Releases and Updates promptly after receipt from Altair. Maintenance Releases and Updates are subject to this Agreement. Altair shall only be obligated to provide support and maintenance for the most current release of the Software and the most recent prior release. **Support.** Altair will provide support via telephone and email to Licensee at the fees, if any, as listed on the Order Form. If Support has not been procured for any period of time for paid-up licenses, a reinstatement fee shall apply. Support consists of responses to questions from Licensee's personnel related to the use of the then-current and most recent prior release version of the Software. Licensee agrees to provide Altair with sufficient information to resolve technical issues as may be reasonably requested by Altair. Licensee agrees to the best of its abilities to read, comprehend and follow operating instructions and procedures as specified in, but not limited to, Altair's Documentation and other correspondence related to the Software, and to follow procedures and recommendations provided by Altair in an effort to correct problems. Licensee also agrees to notify Altair of a programming error, malfunction and other problems in accordance with Altair's then current problem reporting procedure. If Altair believes that a problem reported by Licensee may not be due to an error in the Software, Altair will so notify Licensee. Questions must be directed to Altair's specially designated telephone support numbers and email addresses. Support will also be available via email at Internet addresses designated by Altair. Support is available

Monday through Friday (excluding holidays) from 8:00 a.m. to 5:00 p.m. local time in the Global Zone where licensed, unless stated otherwise on the Order Form. **Exclusions.** Altair shall have no obligation to maintain or support (a) altered, damaged or Licensee-modified Software, or any portion of the Software incorporated with or into other software not provided by Altair; (b) any version of the Software other than the current version of the Software or the immediately prior release of the Software; (c) problems caused by Licensee's negligence, abuse or misapplication of Software other than as specified in the Documentation, or other causes beyond the reasonable control of Altair; or (d) Software installed on any hardware, operating system version or network environment that is not supported by Altair. Support also **excludes** configuration of hardware, non-Altair Software, and networking services; consulting services; general solution provider related services; and general computer system maintenance.

6. WARRANTY AND DISCLAIMER. Altair warrants for a period of ninety (90) days after Licensee initially receives the Software that the Software will perform under normal use substantially as described in then current Documentation. Supplier software included in the Software and ISV Software provided to Licensee shall be warranted as stated by the Supplier or the ISV. Copies of the Suppliers' and ISV's terms and conditions of warranty are available on the Altair Support website. Support services shall be provided in a workmanlike and professional manner, in accordance with the prevailing standard of care for consulting support engineers at the time and place the services are performed.

ALTAIR DOES NOT REPRESENT OR WARRANT THAT THE PRODUCTS WILL MEET LICENSEE'S REQUIREMENTS OR THAT THEIR OPERATION WILL BE UNINTERRUPTED OR ERROR-FREE, OR THAT IT WILL BE COMPATIBLE WITH ANY PARTICULAR HARDWARE OR SOFTWARE. ALTAIR EXCLUDES AND DISCLAIMS ALL EXPRESS AND IMPLIED WARRANTIES NOT STATED HEREIN, INCLUDING THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NON-INFRINGEMENT. THE ENTIRE RISK FOR THE PERFORMANCE, NON-PERFORMANCE OR RESULTS OBTAINED FROM USE OF THE PRODUCTS RESTS WITH LICENSEE AND NOT ALTAIR. ALTAIR MAKES NO WARRANTIES WITH RESPECT TO THE ACCURACY, COMPLETENESS, FUNCTIONALITY, SAFETY, PERFORMANCE, OR ANY OTHER ASPECT OF ANY DESIGN, PROTOTYPE OR FINAL PRODUCT DEVELOPED BY LICENSEE USING THE PRODUCTS.

7. INDEMNITY. Altair will defend and indemnify, at its expense, any claim made against Licensee based on an allegation that the Software infringes a patent or copyright ("Claim"); provided, however, that this indemnification does not include claims which are based on Supplier software or ISV software, and that Licensee has not materially breached the terms of this Agreement, Licensee notifies Altair in writing within ten (10) days after Licensee first learns of the Claim; and Licensee cooperates fully in the defense of the claim. Altair shall have sole control over such defense; provided, however, that it may not enter into any settlement bind-

ing upon Licensee without Licensee's consent, which shall not be unreasonably withheld. If a Claim is made, Altair may modify the Software to avoid the alleged infringement, provided however, that such modifications do not materially diminish the Software's functionality. If such modifications are not commercially reasonable or technically possible, Altair may terminate this Agreement and refund to Licensee the prorated license fee that Licensee paid for the then current Term. Perpetual licenses shall be pro-rated over a 36-month term. Altair shall have no obligation under this Section 7, however, if the alleged infringement arises from Altair's compliance with specifications or instructions prescribed by Licensee, modification of the Software by Licensee, use of the Software in combination with other software not provided by Altair and which use is not specifically described in the Documentation, and if Licensee is not using the most current version of the Software, if such alleged infringement would not have occurred except for such exclusions listed here. This section 7 states Altair's entire liability to Licensee in the event a Claim is made. No indemnification is made for Supplier and/or ISV Software.

8. LIMITATION OF REMEDIES AND LIABILITY. Licensee's exclusive remedy (and Altair's sole liability) for Software that does not meet the warranty set forth in Section 6 shall be, at Altair's option, either (i) to correct the nonconforming Software within a reasonable time so that it conforms to the warranty; or (ii) to terminate this Agreement and refund to Licensee the license fees that Licensee has paid for the then current Term for the nonconforming Software; provided, however that Licensee notifies Altair of the problem in writing within the applicable Warranty Period when the problem first occurs. Any corrected Software shall be warranted in accordance with Section 6 for ninety (90) days after delivery to Licensee. The warranties hereunder are void if the Software has been improperly installed, misused, or if Licensee has violated the terms of this Agreement.

Altair's entire liability for all claims arising under or related in any way to this Agreement (regardless of legal theory), shall be limited to direct damages, and shall not exceed, in the aggregate for all claims, the license and maintenance fees paid under this Agreement by Licensee in the 12 months prior to the claim on a prorated basis, except for claims under Section 7. **ALTAIR AND ITS SUPPLIERS AND ISVS SHALL NOT BE LIABLE TO LICENSEE OR ANYONE ELSE FOR INDIRECT, SPECIAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES ARISING HEREUNDER (INCLUDING LOSS OF PROFITS OR DATA, DEFECTS IN DESIGN OR PRODUCTS CREATED USING THE SOFTWARE, OR ANY INJURY OR DAMAGE RESULTING FROM SUCH DEFECTS, SUFFERED BY LICENSEE OR ANY THIRD PARTY) EVEN IF ALTAIR OR ITS SUPPLIERS OR ITS ISVS HAVE BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.** Licensee acknowledges that it is solely responsible for the adequacy and accuracy of the input of data, including the output generated from such data, and agrees to defend, indemnify, and hold harmless Altair and its Suppliers and ISVs from any and all claims, including reasonable attorney's fees, resulting from, or in connection with Licensee's use of the Software. No

action, regardless of form, arising out of the transactions under this Agreement may be brought by either party against the other more than two (2) years after the cause of action has accrued, except for actions related to unpaid fees.

9. UNITED STATES GOVERNMENT RESTRICTED RIGHTS. This section applies to all acquisitions of the Products by or for the United States government. By accepting delivery of the Products except as provided below, the government or the party procuring the Products under government funding, hereby agrees that the Products qualify as “commercial” computer software as that term is used in the acquisition regulations applicable to this procurement and that the government's use and disclosure of the Products is controlled by the terms and conditions of this Agreement to the maximum extent possible. This Agreement supersedes any contrary terms or conditions in any statement of work, contract, or other document that are not required by statute or regulation. If any provision of this Agreement is unacceptable to the government, Vendor may be contacted at Altair Engineering, Inc., 1820 E. Big Beaver Road, Troy, MI 48083-2031; telephone (248) 614-2400. If any provision of this Agreement violates applicable federal law or does not meet the government's actual, minimum needs, the government agrees to return the Products for a full refund.

For procurements governed by DFARS Part 227.72 (OCT 1998), the Software, except as described below, is provided with only those rights specified in this Agreement in accordance with the Rights in Commercial Computer Software or Commercial Computer Software Documentation policy at DFARS 227.7202-3(a) (OCT 1998). For procurements other than for the Department of Defense, use, reproduction, or disclosure of the Software is subject to the restrictions set forth in this Agreement and in the Commercial Computer Software - Restricted Rights FAR clause 52.227-19 (June 1987) and any restrictions in successor regulations thereto.

Portions of Altair's PBS Professional Software and Documentation are provided with RESTRICTED RIGHTS. Use, duplication, or disclosure by the Government is subject to restrictions as set forth in subdivision(c)(1)(ii) of the rights in the Technical Data and Computer Software clause in DFARS 252.227-7013, or in subdivision (c)(1) and (2) of the Commercial Computer Software-Restricted Rights clause at 48 CFR52.227-19, as applicable.

10. CHOICE OF LAW AND VENUE. This Agreement shall be governed by and construed under the laws of the state of Michigan, without regard to that state's conflict of laws principles except if the state of Michigan adopts the Uniform Computer Information Transactions Act drafted by the National Conference of Commissioners of Uniform State Laws as revised or amended as of June 30, 2002 (“UCITA”) which is specifically excluded. This Agreement shall not be governed by the United Nations Convention on Contracts for the International Sale of Goods, the application of which is expressly excluded. Each Party waives its right to a jury trial in the event of any dispute arising under or relating to this Agreement. Each party agrees that money damages may not be an adequate remedy for breach of the provisions of

this Agreement, and in the event of such breach, the aggrieved party shall be entitled to seek specific performance and/or injunctive relief (without posting a bond or other security) in order to enforce or prevent any violation of this Agreement.

11. [RESERVED]

12. Notice. All notices given by one party to the other under the Agreement or these Additional Terms shall be sent by certified mail, return receipt requested, or by overnight courier, to the respective addresses set forth in this Agreement or to such other address either party has specified in writing to the other. All notices shall be deemed given upon actual receipt.

Written notice shall be made to:

Altair: Licensee Name & Address:

Altair Engineering, Inc. _____

1820 E. Big Beaver Rd _____

Troy, MI 48083 _____

Attn: Tom M. Perring Attn: _____

13. TERM. For annual licenses, or Support provided for perpetual licenses, renewal shall be automatic for each successive year (“Renewal Term”), upon mutual written execution of a new Order Form. All charges and fees for each Renewal Term shall be set forth in the Order Form executed for each Renewal Term. All Software licenses procured by Licensee may be made coterminous at the written request of Licensee and the consent of Altair.

14. TERMINATION. Either party may terminate this Agreement upon thirty (30) days prior written notice upon the occurrence of a default or material breach by the other party of its obligations under this Agreement (except for a breach by Altair of the warranty set forth in Section 8 for which a remedy is provided under Section 10; or a breach by Licensee of Section 5 or Section 6 for which no cure period is provided and Altair may terminate this Agreement immediately) if such default or breach continues for more than thirty (30) days after receipt of such notice. Upon termination of this Agreement, Licensee must cease using the Software and, at Altair's option, return all copies to Altair, or certify it has destroyed all such copies of the Software and Documentation.

15. GENERAL PROVISIONS. Export Controls. Licensee acknowledges that the Products may be subject to the export control laws and regulations of the United States and other countries, and any amendments thereof. Licensee agrees that Licensee will not directly or indirectly export the Products into any country or use the Products in any manner except in compliance with all applicable U.S. and other countries export laws and regulations. **Notice.** All notices given by one party to the other under this Agreement shall be sent by certified mail, return receipt requested, or by overnight courier, to the respective addresses set forth in this Agreement or to such other address either party has specified in writing to the other. All

notices shall be deemed given upon actual receipt. **Assignment.** Neither party shall assign this Agreement without the prior written consent of other party, which shall not be unreasonably withheld. All terms and conditions of this Agreement shall be binding upon and inure to the benefit of the parties hereto and their respective successors and permitted assigns. **Waiver.** The failure of a party to enforce at any time any of the provisions of this Agreement shall not be construed to be a waiver of the right of the party thereafter to enforce any such provisions. **Severability.** If any provision of this Agreement is found void and unenforceable, such provision shall be interpreted so as to best accomplish the intent of the parties within the limits of applicable law, and all remaining provisions shall continue to be valid and enforceable. **Headings.** The section headings contained in this Agreement are for convenience only and shall not be of any effect in constructing the meanings of the Sections. **Modification.** No change or modification of this Agreement will be valid unless it is in writing and is signed by a duly authorized representative of each party. **Conflict.** In the event of any conflict between the terms of this Agreement and any terms and conditions on a Licensee Purchase Order or comparable document, the terms of this Agreement shall prevail. Moreover, each party agrees any additional terms on any Purchase Order or comparable document other than the transaction items of (a) item(s) ordered; (b) pricing; (c) quantity; (d) delivery instructions and (e) invoicing directions, are not binding on the parties. In the event of a conflict between the terms of this Agreement, and the Additional Terms, the Agreement shall take precedence. **Entire Agreement.** This Agreement, the Additional Terms, and the Order Form(s) attached hereto constitute the entire understanding between the parties related to the subject matter hereto, and supersedes all proposals or prior agreements, whether written or oral, and all other communications between the parties with respect to such subject matter. This Agreement may be executed in one or more counterparts, all of which together shall constitute one and the same instrument. **Execution.** Copies of this Agreement executed via original signatures, facsimile or email shall be deemed binding on the parties.

Index

A

- accelerator [274](#)
- accelerator_memory [274](#)
- accelerator_model [274](#)
- Accounting
 - job arrays [252](#)
- accounting [226](#)
- ACCT_TMPDIR [226](#)
- Administrator Guide [11](#)
- Advance reservation
 - creation [211](#)
- advance reservation [209](#)
- AIX [93](#)
 - Large Page Mode [228](#)
- Altering
 - job arrays [249](#)
- Ames Research Center [ix](#)
- AOE [301](#)
 - using [302](#)
- API [3](#)
- application licenses
 - floating [36](#)
 - node-locked
 - per-CPU [37](#)
 - per-host [36](#)
 - per-use [37](#)
- arrangement [43](#)

Attribute

- account_string [76](#)
- priority [70](#)
- rerunnable [69](#)

attributes

- modifying [153](#)

B

Batch

- job [10](#)

block [194](#)

Boolean Resources [34](#)

Built-in Resources [24](#)

C

Changing

- order of jobs [161](#)

Checking status

- of jobs [170](#)
- of queues [173](#)
- of server [172](#)

Checkpointable [72](#)

Checkpointing

- interval [72](#)
- job arrays [252](#), [252](#)

checkpointing [157](#)

chunk [33](#)

Index

CLI [11](#), [11](#)

Command line interface [11](#)

Commands [2](#)

commands

 and provisioning [305](#)

comment [179](#)

count_spec [212](#)

credential [228](#)

CSA [226](#)

csh [14](#)

Custom resources [32](#)

D

DCE [227](#), [228](#)

Dedicated Time [225](#)

Default Resources [35](#)

Deleting

 job array range [249](#)

 job arrays [249](#)

 subjob [249](#)

Deleting Jobs [159](#)

Deprecations [9](#)

Destination

 specifying [64](#)

devtype [95](#)

directive [10](#), [18](#), [57](#), [57](#), [58](#), [58](#), [147](#), [208](#),
[309](#), [309](#)

Directives [28](#)

directives [25](#)

Display

 nodes assigned to job [178](#)

 non-running jobs [177](#)

 queue limits [180](#)

 running jobs [176](#)

 size in gigabytes [177](#)

 size in megawords [177](#)

 user-specific jobs [175](#)

Distributed

 workload management [1](#)

E

Email

 notification [67](#)

euidevice [95](#)

euilib [95](#)

exclhost [44](#)

exclusive [44](#)

Executor [3](#)

Exit Status

 job arrays [253](#)

F

Fairshare

 job arrays [253](#)

File

 output [198](#)

 output and error [66](#)

 rhosts [16](#)

 specify name of [65](#)

 staging [198](#)

Files

 cshrc [13](#)

 hosts.equiv [17](#)

 login [13](#)

 pbs.conf [18](#), [149](#)

 profile [13](#)

 rhosts [17](#)

 xpbsrc [149](#), [149](#)

files

 .login [13](#)

 .logout [14](#)

floating licenses [36](#)

free [44](#)

freq_spec [212](#)

G

Graphical user interface [11](#)

group=resource [43](#), [44](#)

grouping [43](#)

GUI [11](#)

Index

H

here document [31](#)
hfile [95](#)
Hitchhiker's Guide [97](#)
Hold
 or release job [156](#)
Holding a Job Array [249](#)
hostfile [95](#)
HPC Basic Profile [255](#)
HPC Basic Profile Job [255](#)
HPC Basic Profile Server [255](#)
HPCBP
 Executable location [257](#)
 Job resources [260](#)
 Job submission requirements [258](#)
 Monitoring jobs [262](#)
 Password requirement [257](#)
 qsub command [259](#)
 qsub syntax [259](#)
 Submitting jobs [257](#)
 Unsupported commands [269](#)
 User account [256](#)
HPCBP Job [255](#)
HPCBP MOM [255](#)

I

identifier [29](#)
Identifier Syntax [236](#)
InfiniBand [120](#), [121](#)
instance [210](#)
instance of a standing reservation [210](#)
instances
 option [95](#)
Intel MPI
 examples [102](#)
Interactive job submission
 job arrays [237](#)
Interactive-batch jobs [77](#)
interval_spec [212](#)

J

ja [226](#)
Job
 checkpointable [72](#)
 comment [179](#)
 dependencies [195](#)
 identifier [29](#)
 name [69](#)
 selecting using xpbs [188](#)
 sending messages to [159](#)
 sending signals to [160](#)
 submission options [62](#)
 tracking [189](#)
Job Array
 Attributes [238](#)
 identifier [235](#)
 range [235](#)
 States [238](#)
Job Array Run Limits [251](#)
Job Arrays [235](#)
 checkpointing [252](#)
 deleting [249](#)
 exit status [253](#)
 interactive submission [237](#)
 PBS commands [243](#)
 placement sets [253](#)
 prologues and epilogues [252](#)
 qalter [249](#)
 qdel [249](#)
 qhold [249](#)
 qmove [249](#)
 qorder [249](#)
 qrerun [250](#)
 qrls [250](#)
 qrun [250](#)
 qselect [251](#)
 run limits [251](#)
 starving [252](#)
 status [245](#)
 submitting [237](#)
Job Arrays and xpbs [251](#)
Job Script [25](#)

Index

Job Submission Description Language [255](#)

Job Submission Options [62](#)

job-wide [34](#)

JSDL [255](#)

K

Kerberos [228](#)

qsub -W cred=DCE [227](#)

KRB5 [228](#)

krb5 [228](#)

L

Large Page Mode [228](#)

Limits on Resource Usage [42](#)

Listbox [134](#)

M

man pages

SGI [14](#)

MANPATH [14](#)

max_walltime [233](#)

min_walltime [233](#)

Modifying Job Attributes [153](#)

MOM [3](#)

Monitoring [1](#)

Moving [249](#)

jobs between queues [163](#)

Moving a Job Array [249](#)

MP_DEVTYPE [95](#)

MP_EUIDEVICE [95](#)

MP_EUILIB [95](#)

MP_HOSTFILE [95](#)

MP_INSTANCES [95](#)

MP_PROCS [96](#)

MPI

Intel MPI

examples [102](#)

MPICH_GM

rsh/ssh

examples [110](#)

MPICH2

examples [118](#), [122](#)

MPICH-GM

MPD

examples [109](#)

MPICH-MX

MPD

examples [112](#)

rsh/ssh

examples [114](#)

MVAPICH1 [119](#)

examples [120](#)

MPI, LAPI [93](#)

MPICH [106](#)

MPICH_GM

rsh/ssh

examples [110](#)

MPICH2

examples [118](#), [122](#)

MPICH-GM

MPD

examples [109](#)

MPICH-MX

MPD

examples [112](#)

rsh/ssh

examples [114](#)

MPI-OpenMP [129](#)

MRJ Technology Solutions [ix](#)

MVAPICH1 [119](#)

examples [120](#)

N

naccelerators [275](#)

name [69](#)

NASA

and PBS [ix](#)

nchunk [275](#)

Network Queueing System

nqs2pbs [309](#)

Index

Node Grouping

 job arrays [253](#)

Node Specification Conversion [54](#)

Node specification format [54](#)

nqs2pbs [11](#)

O

OpenMP [127](#)

Ordering job arrays [249](#)

Ordering Job Arrays in the Queue [249](#)

override [28](#)

P

pack [44](#)

Parallel Virtual Machine (PVM) [126](#)

password

 single-signon [61](#)

 Windows [60](#), [60](#)

PBS commands

 job arrays [243](#)

PBS Environmental Variables [239](#)

PBS_ARRAY_ID [239](#)

PBS_ARRAY_INDEX [239](#)

PBS_DEFAULT [18](#)

PBS_DEFAULT_SERVER [149](#)

PBS_DPREFIX [18](#)

PBS_ENVIRONMENT [13](#), [13](#), [18](#)

pbs_hostn [11](#)

PBS_JOBID [239](#)

pbs_migrate_users [11](#)

PBS_O_WORKDIR [18](#)

pbs_password [11](#), [61](#), [61](#)

pbs_probe [11](#)

pbs_rdel [11](#)

pbs_rstat [11](#)

pbs_rsub [11](#), [215](#)

pbs_tclsh [11](#)

PBScrayhost [276](#)

PBScraylabel [276](#)

PBScraynid [277](#)

PBScrayorder [277](#)

pbsdsh [11](#)

pbsfs [11](#)

pbsnodes [11](#)

pbs-report [11](#)

Peer Scheduling

 job arrays [253](#)

per-CPU node-locked licenses [37](#)

per-host node-locked licenses [36](#)

per-use node-locked licenses [37](#)

place statement [43](#)

placement sets

 job arrays [253](#)

POE [93](#)

poe

 examples [98](#)

Preemption

 job arrays [253](#)

printjob [11](#)

procs [96](#)

PROFILE_PATH [16](#)

Prologues and Epilogues

 job arrays [252](#)

provision [301](#)

provisioned vnode [301](#)

provisioning [303](#)

 allowing time [306](#)

 and commands [305](#)

 AOE restrictions [304](#)

 host restrictions [303](#)

 requesting [305](#)

 using AOE [302](#)

 vnodes [302](#)

PVM (Parallel Virtual Machine) [126](#)

Q

qalter [11](#), [144](#)

 job array [249](#)

qdel [11](#), [144](#)

 job arrays [249](#)

qdisable [11](#), [144](#)

qenable [11](#), [144](#)

Index

qhold [11](#), [144](#), [156](#), [158](#)
 job arrays [249](#)
qmgr [11](#)
qmove [11](#), [144](#), [163](#)
 job array [249](#)
qmsg [11](#), [144](#), [159](#), [250](#)
qorder [11](#), [145](#), [161](#), [162](#)
 job arrays [249](#)
qrerun [11](#), [144](#)
 job arrays [250](#)
qrls [11](#), [144](#), [157](#), [158](#)
 job arrays [250](#)
qrun [12](#), [144](#)
 job array [250](#)
qselect [11](#), [150](#), [150](#), [150](#), [151](#), [151](#), [187](#),
[187](#), [188](#)
 job arrays [251](#)
qsig [12](#), [144](#), [160](#)
qstart [12](#), [144](#)
qstat [12](#), [144](#), [155](#), [155](#), [157](#), [162](#), [162](#), [169](#),
[170](#), [170](#), [170](#), [170](#), [172](#), [172](#), [173](#), [173](#),
[175](#), [176](#), [176](#), [176](#), [177](#), [177](#), [177](#), [178](#),
[178](#), [179](#), [179](#), [180](#), [180](#), [187](#), [187](#), [188](#)
qstop [12](#), [144](#)
qsub [12](#), [12](#), [57](#), [58](#), [59](#), [62](#), [144](#), [144](#), [194](#),
[195](#), [228](#)
 Kerberos [227](#)
qsub options [62](#)
qterm [12](#), [144](#)
Queuing [1](#)
Quick Start Guide [xi](#)

R

rcp [12](#)
recurrence rule [212](#)
Releasing a Job Array [250](#)
report [226](#)
requesting provisioning [305](#)
Requeuing a Job Array [250](#)
Reservation
 deleting [220](#)

reservation
 advance [209](#), [211](#)
 degraded [210](#)
 instance [210](#)
 Setting start time & duration [213](#)
 soonest occurrence [210](#)
 standing [210](#)
 instance [210](#)
 soonest occurrence [210](#)
 standing reservation [212](#)
 Submitting jobs [220](#)

reservations
 time for provisioning [306](#)
Resource Specification Conversion [56](#)
Resource specification format [56](#)
resource_list [63](#)
resources [28](#)
restrictions
 AOE [304](#)
 provisioning hosts [303](#)
resv_nodes [210](#)
rhosts [16](#), [16](#)
run limits
 job arrays [251](#)
Running a Job Array [250](#)

S

scatter [44](#)
Scheduler [3](#)
Scheduling [1](#)
 job Arrays [253](#)
scp [12](#)
Selection of Job Arrays [251](#)
selection statement [33](#)
Sequence number [235](#)
Server [3](#)
setting job attributes [28](#)
share [44](#)
sharing [43](#)
shell [25](#)
SIGKILL [160](#)
SIGNULL [160](#)

Index

SIGTERM [160](#)
single-signon [61](#)
Single-Signon Password Method [61](#)
soonest occurrence [210](#)
spec [55](#)
spec_list [54](#)
stageout [63](#)
staging
 Windows
 job arrays [243](#)
Standing Reservation [209](#)
standing reservation [210](#), [212](#)
Starving
 job arrays [252](#)
States
 job array [238](#)
states [150](#), [188](#)
Status
 job arrays [245](#)
stepping factor [237](#)
Subjob [235](#)
Subjob index [235](#)
submission options [62](#)
Submitting a job array [237](#)
Submitting a PBS Job [21](#)
suffix [54](#)
Suppressing job identifier [76](#)
syntax
 identifier [236](#)

T

TCL [133](#)
TGT [228](#)
time between reservations [224](#)
TK [133](#)
TMPDIR [18](#)
tracejob [12](#)
tracking [189](#)

U

umask [194](#)

Unset Resources [24](#)
until_spec [213](#)
User Guide [xi](#)
user job accounting [226](#)
username [16](#)
 maximum [13](#)

V

Viewing Job Information [173](#)
Vnode Types [22](#)
vnodes
 provisioning [302](#)
vntype [276](#)
vscatter [44](#)





W

Wait for Job Completion [194](#)
Widgets [134](#)
Windows [14](#), [16](#)
 job arrays
 staging [243](#)
 password [60](#), [60](#)
 staging
 job arrays [243](#)

X

xpbs [12](#), [145](#), [149](#), [149](#), [150](#), [152](#)
 buttons [144](#)
 configuration [149](#)
 job arrays [251](#)
 usage [133](#), [160](#), [161](#), [187](#), [188](#), [197](#)
xpbsmon [12](#)
xpbsrc [148](#)

Index

-  PBSPro Programmer's Guide
-  PBSPro Quick Start Guide
-  PBSPro Reference Guide
-  PBSPro User's Guide

PBS Professional® 12.0

Quick Start Guide



PBS Works™

PBS Works is a division of  Altair

PBS Professional Quick Start Guide

Altair® PBS Professional 12, updated 25 January 2013

Copyright © 2003-2012 Altair Engineering, Inc. All rights reserved.

PBS™, PBS Works™, PBS GridWorks®, PBS Professional®, PBS Analytics™, PBS Catalyst™, e-Compute™, and e-Render™ are trademarks of Altair Engineering, Inc. and are protected under U.S. and international laws and treaties. All other marks are the property of their respective owners.

ALTAIR ENGINEERING INC. Proprietary and Confidential. Contains Trade Secret Information. Not for use or disclosure outside ALTAIR and its licensed clients. Information contained herein shall not be decompiled, disassembled, duplicated or disclosed in whole or in part for any purpose. Usage of the software is only as explicitly permitted in the end user software license agreement.

Copyright notice does not imply publication.

For documentation and the PBS Works forums, go to:

Web: www.pbsworks.com

For more information, contact Altair at:

Email: pbssales@altair.com

Technical Support

Location	Telephone	e-mail
North America	+1 248 614 2425	pbssupport@altair.com
China	+86 (0)21 6117 1666	es@altair.com.cn
France	+33 (0)1 4133 0992	francesupport@altair.com
Germany	+49 (0)7031 6208 22	hwsupport@altair.de
India	+91 80 66 29 4500	pbs-support@india.altair.com
Italy	+39 0832 315573 +39 800 905595	support@altairengineering.it
Japan	+81 3 5396 2881	pbs@altairjp.co.jp
Korea	+82 31 728 8600	support@altair.co.kr
Scandinavia	+46 (0)46 286 2050	support@altair.se
UK	+44 (0)1926 468 600	pbssupport@uk.altair.com

This document is proprietary information of Altair Engineering, Inc.

PBS Professional® Quick Install for UNIX and Linux

This section describes how to install socket licenses and PBS Professional on UNIX and Linux.

This process covers installation on both stand-alone machines and complexes. For more information on software and license installation and pre-installation planning, consult the **PBS Professional Installation & Upgrade Guide**.

General Information

If you will use floating licenses for any of your hosts, PBS requires an LM-X Altair floating license server. You **cannot** use the old Flex license server. If you will use the floating license server and you are upgrading from pre-11.0 PBS, you must download, install and configure the LM-X Altair license server **before** installing PBS Professional.

Types of Installation

- For **standalone** machines, perform a full installation of PBS Professional. Further configuration is optional.
- For **cluster systems**, first install on the PBS server host, then install on each execution host.
- For **client machines** (those from which jobs will be submitted, but which will not execute jobs) you can install PBS any time *after* the other PBS installations have been completed.

Step 1: Choose Licensing

You can license PBS jobs in the following ways: you can license all hosts via socket licenses, you can license all hosts using floating licenses, or you can license some hosts with sockets and some with floating licenses.

Using Socket Licenses For All Hosts

If you will use socket licenses for all hosts, you can create the accounts required by PBS, download and install PBS, then obtain your socket license file and install it where the PBS server can find it. You do not need the Altair License Server. To download socket licenses, see section , Downloading Socket Licenses , on page 14. For licensing details, see the PBS Professional Installation and Upgrade Guide.

tion , Downloading Socket Licenses , on page 14. For licensing details, see the PBS Professional Installation and Upgrade Guide.

Using Floating Licenses for All Hosts

If you will use floating licenses for all hosts, you must first download and install the Altair license server. Then you can create the accounts required by PBS, then download and install PBS. To download floating licenses, see section , Downloading Floating Licenses , on page 13. For licensing details, see the PBS Professional Installation and Upgrade Guide.

Using a Mix of Socket and Floating Licenses

If you will use both socket and floating licenses, you must first download and install the Altair license server. Then you can create the accounts required by PBS, download and install PBS, then obtain your socket license file and install it where the PBS server can find it. To download socket and floating licenses, see section , Downloading Socket Licenses , on page 14 and section , Downloading Floating Licenses , on page 13. For licensing details, see the PBS Professional Installation and Upgrade Guide.

Step 2: Download, Install and Start the Altair License Server

For instructions on downloading and installing the Altair license server, see the instructions on page 10 and the Altair License Management System Installation and Operations Guide, available at the Altair website, www.altair.com.

Step 3: Prepare to Install PBS Professional

(See also *Installation & Upgrade Guide*.)

- Log in as Administrator (root). You must have administrator privileges to install PBS Professional.
- Create the data service account. See the PBS Professional Installation & Upgrade Guide for requirements.
- Download the appropriate package from the PBS Professional website. See Downloading Software on page 10. Extract the file to a directory of your choice.

Step 4: Start the Installation Program

(See also *Installation & Upgrade Guide*.)

- Change to the PBSPro_<version> subdirectory where you extracted the PBS file.
- To start the installation process, enter this command at the UNIX prompt:

```
./INSTALL
```

Step 5: Follow the Installation Program

(See also *Installation & Upgrade Guide*.)

- **Execution directory (PBS_EXEC)**
This step identifies the directory into which the executable programs and libraries will be installed. To change the default destination, enter a new pathname.
- **Home directory (PBS_HOME)**
This step identifies the directory into which the PBS Professional daemon configuration files and logfiles will be installed. To change the default destination, enter a new pathname.

- **PBS installation type**

Select the desired type of installation (see below):

- If this is a **stand-alone** system, select option 1.
- If this is a cluster **server host** (front-end), select option 1.
- If this is a cluster **execution host**, select option 2.
- If this is a **client machine**, select option 3.

Installation Begins

Approximate load time varies by the option selected and whether the installation is being done across the network.

Step 6: Enter the License File Location

(See also the the *Installation & Upgrade Guide*.)

If this is a **stand-alone** or **front-end machine**, you will be prompted to enter the location of the license server or license file. Type the location(s) at the prompt when requested.

Step 7: Finish Installation

If this is a cluster **front-end machine**, enter *n* (for “no”) when the installation program offers to start PBS. Then proceed directly to the next step for further configuration.

However, if this is a **client machine**, then the installation is complete.

If this is a **stand-alone** or an **execution machine**, the installation program will offer to start PBS. Enter *n* (for “no”).

At this time, you may repeat the install process for other **execution** or **client machines**.

Step 8: Perform Front-end Configuration

(See also *Installation & Upgrade Guide*.)

After the **front-end machine** has been installed, further configuration is required.

- If you do *not* intend to execute jobs on this **front-end machine**, then edit its global configuration file `/etc/pbs.conf`, setting the entry for `PBS_START_MOM=0`. This will tell PBS not to start the MoM daemon on this machine.

Step 9: Start PBS Professional

Run the PBS start script:

```
<path to script>/pbs start
```

This is often in `/etc/init.d`.

Step 10: Configure for Socket Licenses

- If you will use socket licenses, get a socket license file from Altair, and install it where the server can reach it. Then configure the server's `pbs_license_info` attribute to use it. For information on getting the socket license file, see the instructions on page 14 and the Altair License Management System Installation and Operations Guide.

Step 11: Perform Cluster Configuration

- Create the list of machines PBS will manage. Use `qmgr` to add one host name for each **execution host** in your cluster. (See also *Installation & Upgrade Guide*.)

Step 12: Test PBS Professional

The following commands are useful in testing that PBS Professional is up and running:

Command	Purpose
<code>pbsnodes -a</code>	Show status of nodes in cluster
<code>qsub</code>	Command to submit a job
<code>qdel</code>	Command to delete a job
<code>qstat</code>	Show job, queue, & Server status
<code>qstat -B</code>	Briefly show PBS Server status
<code>qstat -Bf</code>	Show detailed PBS Server status
<code>qstat -Q</code>	Briefly list all queues
<code>qstat -Qf</code>	Show detailed queue information
<code>tracejob</code>	Extracts job info from log files
<code>xpbs</code>	PBS graphical user interface
<code>xpbsmon</code>	PBS graphical monitoring tool

The above commands are discussed in the **PBS Professional Reference Guide**. You can find detailed information on managing PBS Professional in the **PBS Professional Administrator's Guide**.

Other Considerations

The following situations are *not* covered by this **PBS Professional Quick Start Guide**. We recommend you review the **PBS Professional Administrator's Guide** for details regarding the following:

- Performing an upgrade rather than a new install
- Installing MoM with SGI[®] cpuset support
- Installing on a Cray
- Installing on Solaris

PBS Professional® Quick Install for Windows

This section describes how to install PBS Professional on Windows.

This process covers installation on both stand-alone machines and complexes. For more information on software and license installation and pre-installation planning, consult the **PBS Professional Installation & Upgrade Guide**.

General Information

(See also *Installation & Upgrade Guide*.) PBS requires an LM-X Altair license server. You **can-not** use PBS with the previous FLEX license server. If you are upgrading from pre-11.0 PBS, you must download, install and configure the LM-X Altair license server **before** installing PBS Professional.

Socket licenses are not available for Windows.

- For **standalone** machines, perform a full install of PBS Professional. Further configuration is optional.
- For **cluster** systems, first install on the front-end machine (server host), then install on each execution host.
- For **client machines** (those from which jobs will be submitted, but which will not execute jobs), you can install PBS at any time *after* the other PBS installations have been completed.

Step 1: Download, Install and Start the Altair License Server

For instructions on downloading and installing the Altair license server, see the instructions on page 10 and the Altair License Management System Installation and Operations Guide, available at the Altair website, www.altair.com.

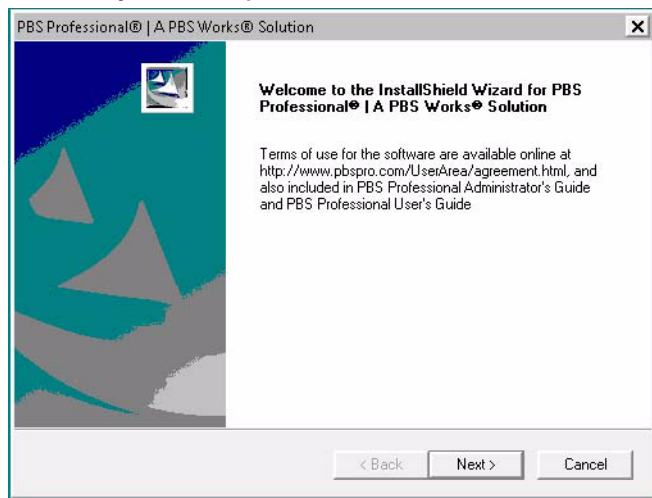
Step 2: Prepare to Install PBS Professional

(See also *Installation & Upgrade Guide*.)

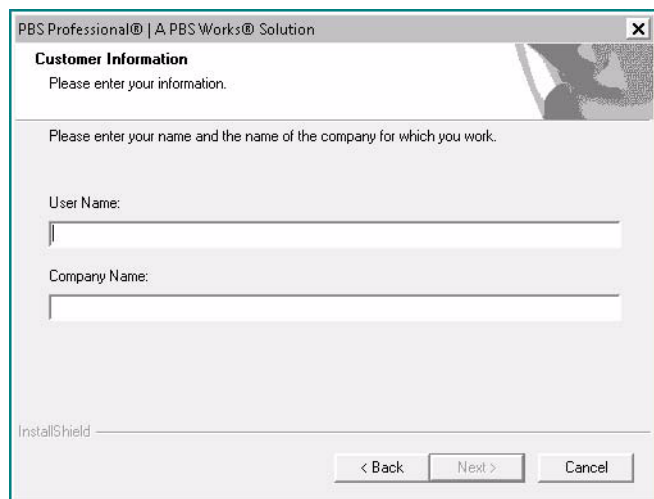
- Log in under an account that is a member of the local Administrators group on the local computer.
- Create the service account.
For domained environments, the service account must:
 - Be a domain account
 - Be a member of the *Domain Users* group
 - Have *domain read* privilege to all users and groups
- You must delegate *read access to all users and groups information* to the service account.
- For standalone environments, the service account must be a local account that is a member of the local Administrators group on the local computer.
- Download the PBS Professional package. Follow the instructions on page 10.
- Uncompress the PBS package you downloaded.
- Browse to the package (.exe file) and double-click on the **PBSPro_<version>**. This will launch the InstallShield Wizard.

Step 3: Follow the InstallShield Wizard

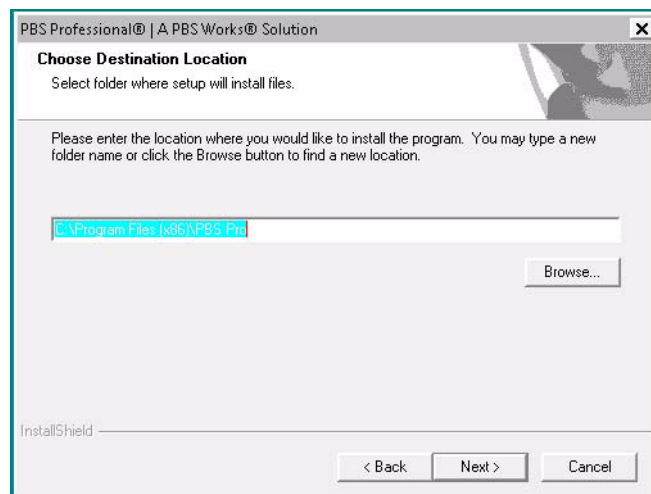
- On the **Welcome** screen, read **Terms of Use**, and if you accept, click **Next**:



- In the **Customer Information** screen, enter the **User Name** and **Company Name**. Click **Next**:

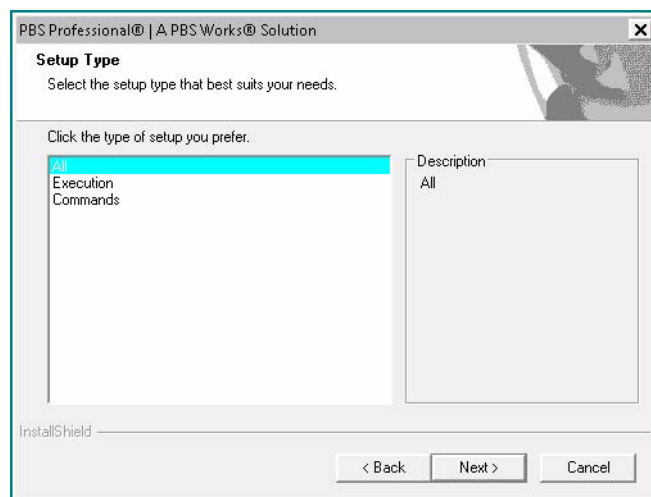


- In the **Choose Destination Location** screen, choose the folder where you want PBS Professional installed. Then click **Next**:



- In the **Setup Type** screen, select the type of installation you wish to perform.
 - If this is a **standalone** system, select **All**.
 - If this is a cluster **front-end machine** (Server host), select **All**.
 - If this is a cluster **execution machine** (Compute host), select **Execution**.
 - If this is a **client machine**, select **Commands**.

Then click **Next**:



- In the **User Account Information** screen, enter the administrator account name and password. Then click **Next**:

- You will see an **Information** screen listing privileges required for the administrator account. Click **Next**.
- You will see an **Information** screen showing that the administrator account has been validated and listing the privileges set for the administrator account. Click **Next**.
- In the **Editing HOSTS.EQUIV File** screen, specify any host(s) or user(s) who will need local filesystem access, then click **Next**:

Step 4: Perform Server-Specific Configuration

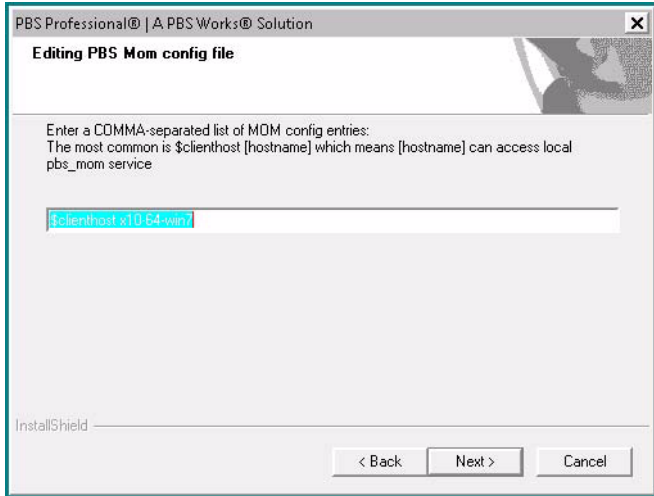
(See also the *Administrator's Guide*.)

- In the **Enter License File Location** screen, type your license file or server location, and click **Next**:

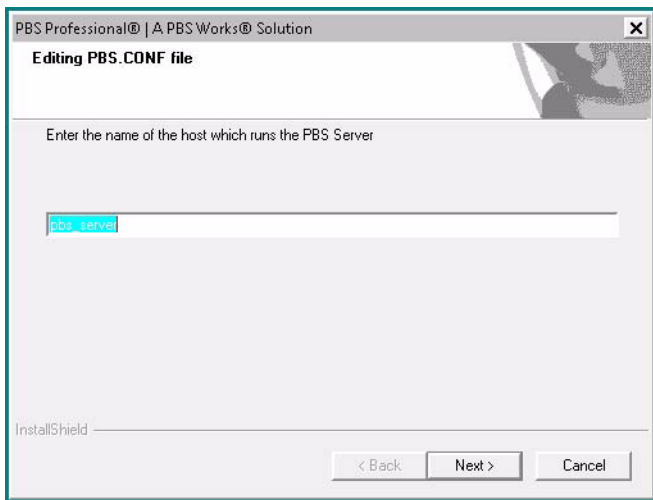
- In the **Editing PBS Server Nodes** file screen, enter the names of the nodes in your cluster, separated by commas, then click **Next**:

Step 5: Perform Host-specific Configuration

If this is an execution machine, you will see the following **Editing PBS MOM config file** screen. Enter any MoM configuration parameters you wish to set for this computer, then click **Next**:



- The **Editing PBS.CONF File** screen will appear for an execution-only install. Enter the name of the host that will be running the PBS server, then click **Next**:

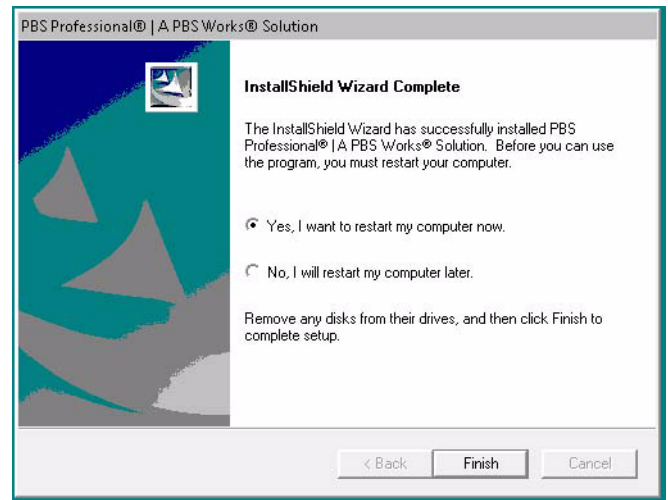


Installation Begins

Approximate load time varies by the option selected and whether the installation is being performed across the network.

Step 6: Finish Installation

Select **Yes** to reboot this computer, then click **Finish**:



At this time, you may repeat the install process for other **execution** or **client machines**. If this system is part of a cluster, further configuration may be required.

Step 7: Verify List of Hosts

(See also *Installation & Upgrade Guide*.)

After you install PBS on the **front-end machine**, you must perform further configuration of that machine.

- Use the `pbsnodes` command to verify the list of machines PBS will manage. Use `qmgr` to add any missing **execution hosts** in your cluster.
- If you do *not* intend to execute jobs on this **front-end machine**, then use `qmgr` to delete the front-end machine's name from the list of nodes, and run the following command to prevent MoM from running on this node:

```
pbs-config-add PBS_START_MOM=0
```

This will set the parameter that tells PBS *not* to start the MoM service on this machine. (See also *Installation & Upgrade Guide*.)

- Now start the PBS software by running the following commands as Administrator (Windows 2000) or as a user with Administrator privilege (Windows XP). (See also *Installation & Upgrade Guide*.)

```
net start pbs_server
```

```
net start pbs_mom (execution nodes only)
```

```
net start pbs_sched
```

```
net start pbs_rshd
```

Step 8: Test PBS Professional

The following commands are useful in testing that PBS Professional is up and running:

Command	Purpose
<code>pbsnodes -a</code>	Show status of nodes in cluster
<code>qsub</code>	Command to submit a job
<code>qdel</code>	Command to delete a job
<code>qstat</code>	Show job, queue, & Server status
<code>qstat -B</code>	Briefly show PBS Server status
<code>qstat -Bf</code>	Show detailed PBS Server status
<code>qstat -Q</code>	Briefly list all queues
<code>qstat -Qf</code>	Show detailed queue information
<code>tracejob</code>	Extracts job info from log files
<code>xpbs</code>	PBS graphical user interface
<code>xpbsmon</code>	PBS graphical monitoring tool

The above commands are discussed in the **PBS Professional Reference Guide**. You can find detailed information on managing PBS Professional in the **PBS Professional Administrator's Guide**.

PBS Professional® Software Download from Website

This section describes how to download packages for the Altair license server and PBS Professional.

General Information

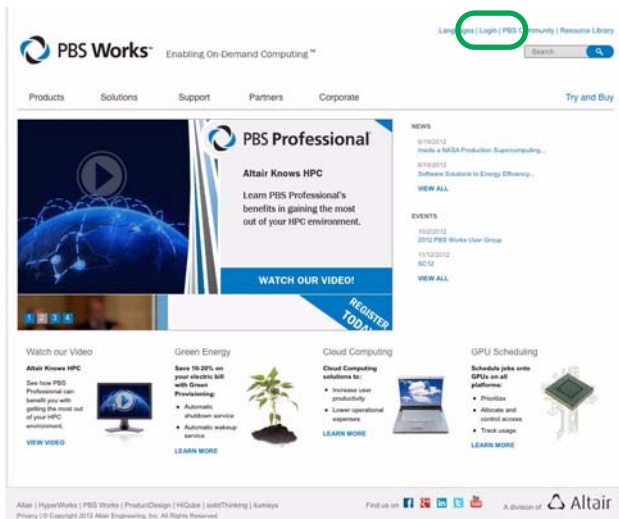
You can use any web browser to access the PBS Professional website. Both the PBS software and the license server are available here.

Step 1: Go To PBS Professional Website

Go to the PBS Professional website at www.pbsworks.com.

Step 2: Log in to the Customer Area

- Click on **Login** in the upper right corner of the website window. A window will pop up.
- Choose **Client Login/User Area**. Ignore the text entry slot.



- When prompted, enter your Site ID and password. This information is provided by your Altair PBS Professional sales representative.

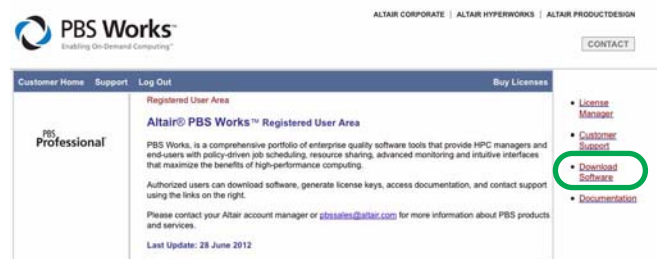


Step 3: Accept Terms

- You must accept the terms the first time you go to the site. Enter the required information.

Step 4: Go to Software Download Page

- Click on **Download Software**:



Step 5: Select Package to Download

- Click on **PBS Professional** to expand the folder.
- Choose the manufacturer to expand to a list of packages.
- Click on the package you wish to download. You can download a package for all PBS components or for the execution host(s) only.



PBS Professional® Documentation Download from Website

This section describes how to download the documentation for the Altair license server and PBS Professional.

General Information

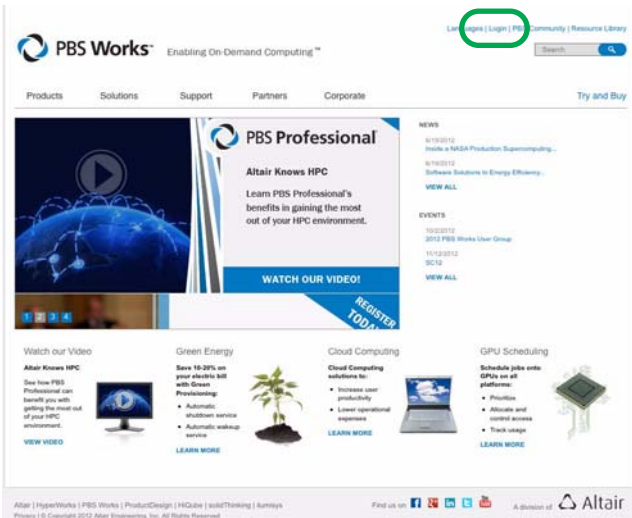
You can use any web browser to access the PBS Professional website.

Step 1: Go To PBS Professional Website

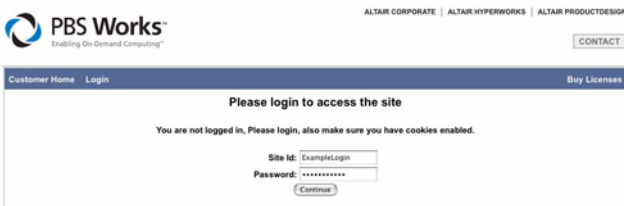
Go to the PBS Professional website at www.pbsworks.com

Step 2: Log in to the Customer Area

- Click on **Login** in the upper right corner of the website window. A window will pop up.
- Choose **Client Login/User Area**. Ignore the text entry slot.

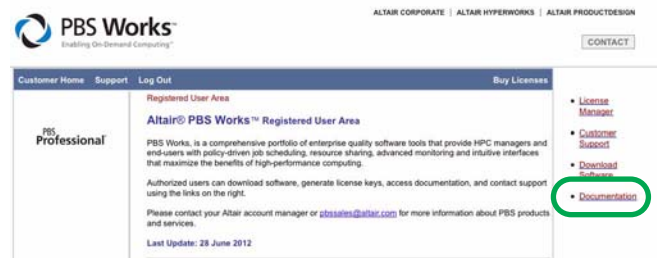


- When prompted, enter your Site ID and Password. This information is provided by your Altair PBS Professional sales representative.



Step 3: Select Documentation

- Click on **Documentation**:



Step 4: Select Documents to Download

- Click on the document you wish to download:

PBS Professional	Formats
PBS Professional 11.3 Installation and Upgrade Guide	PDF
PBS Professional 11.3 Administrator's Guide	PDF
PBS Professional 11.3 User's Guide	PDF
PBS Professional 11.3 Reference Guide	PDF
PBS Professional 11.3 Programmer's Guide	PDF
PBS Professional Common Criteria Administration and Usage Guide (References PBS Professional 10.1 Release)	PDF
PBS Professional 11.3 Release Notes (Update: 26 June 2012)	TEXT
PBS Application Services	Formats
PBS Application Services 11.0.1 Administrator's Guide	PDF
PBS Application Services 11.0.1 Diving Into Application Definitions	PDF
PBS Application Services 11.0.1 Release Notes (Update: 22 December 2011)	PDF
Altair License Manager	Formats
Altair License Manager 11.0.2 Installation and Operations Guide	PDF
Altair License Manager 11.0.2 Release Notes (Update: 12 November 2011)	PDF
Altair License Manager 11.0.1 Release Notes (Update: 02 June 2011)	PDF
HWE Compute Manager	Formats
HWE Compute Manager 11.0.0 Installation and Configuration Guide	ZIP
HWE Compute Manager 11.0.0 Documentation	ZIP

PBS Professional® License Download from Website

This section describes how to download socket and/or floating licenses from the Altair website.

General Information

You can use any web browser to access the PBS Professional website.

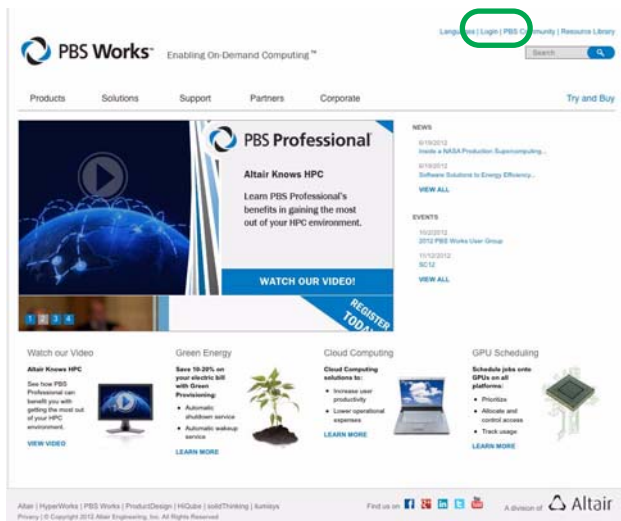
To purchase licenses, contact your PBS sales representative. After you have purchased the licenses, they are available for download on the PBS website.

Step 1: Go to PBS Professional Website

Go to the PBS Professional website at www.pbsworks.com.

Step 2: Log in to the Customer Area

- Click on **Login** in the upper right corner of the website window. A window will pop up.
- Choose **Client Login/User Area**. Ignore the text entry slot.



- When prompted, enter your Site ID and password. This information is provided by your Altair PBS Professional sales representative.



Step 3: Accept Terms

- You must accept the terms the first time you go to the site. Enter the required information.

Step 4: View Unused Licenses

- Click on **License Manager**:



- You will see a page showing the socket and floating licenses you have purchased. The **Scope** field in the **Unused Licenses** table shows the type of license: socket licenses show “socket”, and floating licenses show “HWU”.

Downloading Floating Licenses

Step 1: Begin License Download Process

- The following figure shows unused floating licenses. Click on **Create New License**:

Site id: ExampleLogin

Unused Licenses

Line	Class	Platform	Scope	Expiration	Available Quantity	Create New License
1	Commercial	All	HWU	2012-10-08	30.72	Create New License

Step 2: Generate Host ID and Choose License Manager

- Download and uncompress the **almutils** package.
- Run the appropriate **almutil** command to generate your host ID:

almutil -hostid

- You must select the LM-X license manager.

Please enter your license data below

The license you are about to create is a **Commercial, HWU** license for systems running **All**. If this is not correct, please return to the [previous page](#) and select a different license pool from which to create your new license.

1. Enter the **host id type**, **host id**, and **host name** of the machine where your PBS server will run.

For **v11.x** license and later use **LM-X**. Host ID Type and Host ID are determined by using the [almutils \(MD5\)](#) (note this is not the same as the FLEXlm Host ID).

All **v9.x** and **v10.x** license are **FLEXlm** based. Host ID Type and Host ID are determined by running the **lmhostid** or **"lmutil lmhostid"** command on the license server machine. There are several possible formats for the commands output. If the output contains the string **ID_STRING=** or something similar, please choose the appropriate type of **HOST ID** and do not include that portion of the output in the **HOST ID** field below. (Example output of **lmhostid**: **ID_STRING=9c766319-ade35464-asdf3444-0063afef999** then choose **IDString** as Host ID Type, and put **9c766319-ade35464-asdf3444-0063afef999** in the Host ID field)

License Type: FLEXlm ☐ **LM-X** ☒

Step 3: Enter Host Information

- Enter the required information.
- Enter host ID type, host ID and host name.
- If this is a three-server license, enter the information for all three servers.

☒ 1 Server ☐ 3 Servers

Host ID Type	Host ID	Host Name
Ethernet	0123456789	ExampleHost
Choose One		
Choose One		

Step 4: Assign Licenses to PBS

- Assign the licenses to PBS:

2. How many **HyperWorks Units**. (from 0.01 to 30.72) will this PBS installation need? You can input CPU Cores and PBS Catalyst users and PBS Analytics users that you are trying to run **concurrently** in the HWU Calculator below to generate the license for "Total HyperWorks Units".

HWU Calculator		
Product	Quantity	Total HWUs
PBS Professional:	1024 CPU Cores X 0.03 HWUs	30.72
PBS Catalyst:	Users X 1 HWUs	0
PBS Analytics Modeler:	Users X 6.5 HWUs	0
PBS Analytics		

Step 5: Generate Licenses

- Click on **Generate License**:

Please double check the information above. If you create a license using incorrect data it will be unusable and a support analyst will need to assist you in correcting the situation.

Generate License

Step 6: Download Floating Licenses

You will see a page showing the created licenses. You can download your licenses from this page:

License Successfully Created

Hostname	HostID	HWUs
ExampleHost	012345678901	30.72
License Key		
License File		

Please select a menu entry in the column to the right or [click here](#) to continue.

Downloading Socket Licenses

Step 1: Begin License Download Process

- The following figure shows unused socket licenses. It also shows any active socket licenses. In this example, there are no active socket licenses. Click on **Create New License**:

Site id: ExampleLogin

Unused Licenses

Line	Class	Platform	Scope	Expiration	Available Quantity	Create New License
1	Commercial	All	Socket	perpetual	512 Sockets	Create New License

Active Licenses

Line	Hostname (Hostid)						
	License Key						
	Class	Platform	Scope	License Type	Created	Expiration	HWUs
There are no licenses of this type.							

Step 2: Generate Host ID

- Download and uncompress the **almutils** package.
- Run the appropriate `almutil` command to generate your host ID:

almutil -hostid

Please enter your license data below

The license you are about to create is a **Commercial, Socket** license for systems running **All**. If this is not correct, please return to the [previous page](#) and select a different license pool from which to create your new license.

1. Enter the **host id type**, **host id**, and **host name** of the machine where your PBS server will run.

LM-X. Host ID Type and Host ID are determined by using the [almutils \(MD5\)](#) (Note this is not the same as the FLEXlm Host ID).

License Type: LM-X

Step 3: Enter License Data

- Enter the license data requested:

1. Primary Server

Host ID Type	Host ID	Host Name
<input type="text" value="Choose One"/>	<input type="text"/>	<input type="text"/>

2. Please check this box to enter a secondary server if you have a failover server

☐ Have a failover server

3. What is the total number of physical Sockets for the whole PBS Professional complex (from 1 to 512)?

Socket(s)

4. What is the total number of CPU cores for the whole PBS Professional complex?

Core(s)

Step 4: Generate License

- Click on **Generate License**:

Please double check the information above. If you create a license using incorrect data it will be unusable and a support analyst will need to assist you in correcting the situation.

Step 5: Download Socket Licenses

You will see a page showing the created licenses. You can download your licenses from this page:

License Successfully Created

Hostname	HostID	Sockets
ExampleHost	012345678901	512
License Key		
License File		

Please select a menu entry in the column to the right or [click here](#) to continue.