

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

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](job-submission-and-execution.html#PBSSaved). 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.	Nodes	cpu_freq attribute
Intel Xeon E5-2665	2.4GHz	cn[1-180], cn[208-209]	24
Intel Xeon E5-2470	2.3GHz	cn[181-207]	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, 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](../network.html) 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](../hardware-overview.html) 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:mpi
procs=16:ompthreads=1
-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

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 $0.96 \times 32 = 30.93$ 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

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

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.

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.

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

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_0_WORKDIR/input .
cp $PBS_0_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_0_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_0_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.

Consider preloading inputs and executables onto [shared scratch](../storage.html) 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.

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](../software/mpi-1/Running_OpenMPI.html) and [Running MPICH2](../software/mpi-1/running-mpich2.html) sections.

Example Jobscript for Single Node Calculation

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](../storage.html) on the node:

...

```
#!/bin/bash
```

```
# change to local scratch directory
cd /lscratch/$PBS_JOBID || exit
```

```
# copy input file to scratch
cp $PBS_0_WORKDIR/input .
cp $PBS_0_WORKDIR/myprog.x .
```

```
# execute the calculation
./myprog.x
```

```
# copy output file to home
cp output $PBS_0_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_0_WORKDIR) 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](../software.1.html) section and the [Capacity computing](capacity-computing.html) section.

Â