

## 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.\*\***

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.

Â

```
...  
$ 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

Â

```
...  
$ 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.

Â

...

```
$ qsub -A OPEN-0-0 -q qfree -l select=10:ncpus=24 ./myjob
```

In this example, we allocate 10Â 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:accelerator_model=phi7120 ./myjob
```

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  
walltime=56:00:00 -I
```

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

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 PBSÂ the UV2000 provides 14 chunks, a chunk per NUMA node (seeÂ [Resource allocation policy](resources-allocation-policy.html)). 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 ofÂ the NUMA nodes allocated to itself.

...

```
Â $ 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](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=24:host=r24u35n680+1:ncpus=24:host=r24u36n681 -I
```

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

Â

...

```
$ 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](../network-1/7d-enhanced-hypercube.html) 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 [InfinibandÂ network](../network-1.html) . 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

-----

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	Req'd Memory	Req'd Time	Elap S	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  $0.96 \times 32 = 30.93$  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 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 atÂ Â Â : Fri Aug 30 02:47:53 CEST 2013
Script nameÂ Â : 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.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

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=24 -N Name0 ./myjob
$ qstat -n -u username

```

isrv5:

Job ID	Username	Queue	Jobname	SessID	NDS	TSK	Req'd Memory	Req'd Time	Elap S	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.

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.

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

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_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=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 ./mympprog.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

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](../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.

HTML commented section #3 (Capacity computing need to be reworked)