Contents

1 Executive Summary

This document is a quick start guide to startup with NSCC Job Scheduler environment.

1.1 Intended Audience

This guide is intended to use by NSCC users who has already enrolled for NSCC Cluster and ready to submit jobs

1.2 Assumptions

While you are reading this document we assume that you are able to Enroll for NSCC services and able to login to NSCC cluster using one of the login methods

1.3 Document Conventions

- Command line

```
@ node where you execute the commands

# | execute commands
  | ------------------------------------------------------------
  | the result of commands.
```

↑
# execute as a root
$ execute as a general user

- Configuration file

```
Submission file/script

Contents of the file.

red line : need to be changed
```

1.4 Support

To receive technical support from NSCC System administrator, please do write us help@nscc.sg or raise a ticket using https://help.nscc.sg ->User Services -> Service Desk Portal

2 Introduction to NSCC Job Scheduler (PBS Pro)

The jobs scheduler installed and configured for use on theNSCC Supercomputer is PBS Pro.

It provides a workload management solution that maximizes the efficiency and utilization of high-performance computing (HPC) resources and improves job turnaround.

Some of the important features of PBS Pro are below

2.1 Robust Workload Management

- Floating licenses
- Scalability, with flexible queues
- Job arrays
- User and administrator interface
- Job suspend/resume
- Application checkpoint/restart
- Automatic file staging
- Accounting logs
- Access control lists

2.2 Advanced Scheduling Algorithms

- Resource-based scheduling
- Preemptive scheduling
- Optimized node sorting
- Enhanced job placement
- Advance & standing reservations
- Cycle harvesting across workstations
- Scheduling across multiple complexes
- Network topology scheduling
- Manages both batch and interactive work
- Backfilling

2.3 Reliability, Availability and Scalability

- Server failover feature
- Automatic job recovery
- System monitoring
- Integration with MPI solutions
- Tested to manage 1,000,000+ jobs per day
- Tested to accept 30,000 Jobs per minute
- EAL3+ security
- Checkpoint support

3 Getting Started with PBS Pro

NSCC is using PBS Pro to schedule jobs on the cluster. This guide contains the basic information needed for users to start submitting jobs.

3.1 PBS Queues

Several queues have been created to satisfy the resource requirements of the various workloads which use the system. The subsequent sections provide an overview of the queue definitions and limits.

3.1.1 External Queues:

End users must submit their jobs to one of the external queues.

Queue Name	Resources Available	Remarks
normal	12 Compute nodes 24 cores per server 4GB/core memory or 96 GB per server	The queue normal is a routing queue and does not execute any jobs. It only routes the job to the internal queues based on the resource requirement
gpu	128 GPU nodes 24 Cores per server 4GB/core memory or 96 GB memory per server	This queue is a routing queue and does not execute any jobs. This queue routes the jobs to the internal queues based on resource requirement
largemem	9 Compute nodes 24/48 cores per server 1TB/4TB/6TB memory configurations	This is an execution queue which can take large memory jobs which requires more than 96GB of memory per server
production	480 compute cores reserved for GIS	Only GIS users can submit jobs to this queue
provision	Special Administrative queue	No access to users
iworkq	Special queue for visualization jobs	Jobs submitted from Display manager to be dispatched to this queue
imeq	Special queue for ime jobs	This queue is under testing

3.1.2 Internal Queues:

These queues are typically execution queues which actually schedule the jobs based on the resources that they request

These queues are reference purpose only and cannot accept jobs directly from users.

Queue Name	Queue Limitations	Run time	Remarks
long	One node per user at a time	120 Hrs	Queue for the jobs which cannot be checkpointed and cannot scale up more than a single node. Longer queue time
dev	Up to 48 cores	1 Hr	Designed to test application execution and jobs can be dispatched almost immediately
small	Upto 24 cores	24 Hrs	Small jobs which require maximum of 1 compute node. Less queue time
medium	More than 24 cores and less than 3000 cores	24 Hrs	Jobs that can scale more than 1 compute node up to 3000 cores
gpunormal	Minimum 1 GPU and maximum 128 GPUs	24 Hrs	Queue for normal GPU jobs which can request for 1 or more GPUs
gpulong	Minimum 1 GPU and maximum 128 GPUs	240 Hrs	Queue for long GPU jobs which cannot be checkpointed. Longer queue time expected when requested more GPU nodes

3.1.3 System wide restrictions

Below are few System wide restrictions that apply for job submission

Job submission to Internal Queues	:	Denied by Policy and System
Executing applications in Login nodes	:	Denied by Policy
Per core Memory	:	4GB/core applied by System
Cores per server/node	:	24 Cores per server applied by System
Cluster wide cores	:	3000 Cores per user
Cluster wide No of Jobs	:	300 Jobs in Queue with all states

3.2 Job Submission

The job submission in NSCC will be illustrated through few examples:

Example 1: Simple Job submission

Login node

```
$  echo sleep 10 |qsub -q normal -l select=1:ncpus=1
   ------------------------------------------------------------------------------------------------------
   154400.wlm01
```

In the above example, the job will be submitted with one command line: sleep 10

Example 2: Submit job using job submission script

Login node

```
$  qsub -q normal -l select=1:ncpus=1 submit.pbs
   -----------------------------------------------------------------
   154400.wlm01
```

Contents of submit.pbs are as below

Login node

```
$  cat submit.pbs
   -----------------------------------------------------------------
   #!/bin/bash
   #PBS -N Sleep Job
   #PBS -j oe
   echo sleep job for 30 seconds
   sleep 30
```

In the above example, resource requested for the job using command line and rest of the job execution commands are in the job submission script. Each of the job script commands are explained below.

```
#PBS is the pbs directive

-q normal          -> Choose the queue "normal"
-l select=1:ncpus=1 -> reserves 1 CPU core for this job
#PBS -N Sleep Job
                   -> -N states Name of the job
                   -> Sleep Job is the name of my job
#PBS -j oe
                   -> -j states to join output and error files together
```

Example 3: Submit job using job submission script, all options in the job submission script

Login node	

$	`qsub submit.pbs`
	`---------------------------------------------------------------------`
	`154400.wlm01`

Contents of submit.pbs are below

Login node	

$	`cat submit.pbs`
	`---------------------------------------------------------------------`
	`#!/bin/bash`
	`#PBS -q normal`
	`#PBS -l select=1:ncpus=1:mem=100M`
	`#PBS -l walltime=00:10:00`
	`#PBS -N Sleep_Job`
	`#PBS -o ~/outputfiles/Sleep_Job.o`
	`#PBS -e ~/errorfiles/Sleep_Job.e`
	`echo sleep job for 30 seconds`
	`sleep 30`

In the above example, the resources are requested using job script. Below is the explanation of each line.

```
#PBS is the pbs directive
#PBS -q normal
                  ->    Choose the queue "normal"

#PBS -l select=1:ncpus=1
                         -> reserves 1 cpus for this job

#PBS -l walltime=00:10:00
                        -> Request for a wall time of 10 Minutes

#PBS -N Sleep Job
                  -> -N states Name of the job
                  -> Sleep Job is the name of my job

#PBS -o ~/outputfiles/Sleep_Job.o
                  -> -o states to create output file "Sleep_Job.o" under the
directory ~/outputfiles

#PBS -e ~/errorfiles/Sleep_Job.e
                  -> -e states to create error file "Sleep_Job.e" under the
directory ~/errorfiles
```

Example4: Submit job for a pre-compiled application for demonstration purpose application R is used for serial job.

Login node	
$	`cat submit.pbs`

```
$   cat submit.pbs
    --------------------------------------------------------------------
    #!/bin/bash
    #PBS -q normal
    #PBS -l select=1:ncpus=1:mem=1G
    #PBS -l walltime=01:00:00
    #PBS -N R_Sample
    cd $PBS_O_WORKDIR
    module load R/3.2.4
    R CMD BATCH simple.R
    # Output of R is generated to simple.Rout
```

In the above example, the resources required for the job are requested using job script. Below is the explanation of each line.

```
#PBS is the pbs directive

#PBS -q normal
                  ->    Choose the queue "normal"

#PBS -l select=1:ncpus=1
                         -> reserves 1 unit of 1 CPU core and 1Gb of memory
                        for this job

#PBS -l walltime=01:00:00
                        -> Request a wall time of 1 Hour

#PBS -N R_Sample
                  -> -N states Name of the job
                  -> R_sample is the name of my job

cd $PBS_O_WORKDIR
                  -> Execute following commands in the current working
directory

module load R/3.2.4
                  -> Load the environment for R/3.2.4

R CMD BATCH simple.R
                  -> R execution command
```

*** To submit the job use the qsub syntax in example 3**

Example5: Submit job for a MPI application for demonstration purpose gromacs application is used.

Login node

```
$  cat submit.pbs
   -------------------------------------------------------------------
   #!/bin/bash
   #PBS -q normal
   #PBS -l select=2:ncpus=24:mem=96G:mpiprocs=24:ompthreads=1
   #PBS -l walltime=16:00:00
   #PBS -N gromacs_example
   module load gromacs/5.1.2/gcc493/impi
   cd $PBS_O_WORKDIR
   mpirun mdrun_mpi -pin on -ntomp $OMP_NUM_THREADS -s inputfile.tpr > gromacs.log
```

```
In the above script the resources are requested through the job script. Each
line of the job script is explained below.

#PBS is the pbs directive

#PBS -q normal
                  ->     Choose the queue “normal”

#PBS -l select=2:ncpus=24:mem=96G:mpiprocs=24:ompthreads=1
                        -> reserves 2 units of 24 cpu, 96Gb memory, 24 MPI
                        processes and one OpenMP thread for this job

#PBS -l walltime=16:00:00
                        -> Request a wall time of 16 Hour

#PBS -N gromacs_example
                        -> Name of the job defined as gromacs_example

module load gromacs/5.1.2/gcc493/impi
                  -> Loads the environment for gromacs/5.1.2/gcc493/impi

cd $PBS_O_WORKDIR
                  -> Execute commands in current working directory

mpirun mdrun_mpi -pin on -ntomp $OMP_NUM_THREADS -s inputfile.tpr
>gromacs.log
```

-> gromacs execution command with mpirun

*** To submit the job use the qsub syntax in example 3**

*** In this command notice mpirun does not have any arguments as PBS Pro is tightly integrated with mpirun command**

*** In case of application enabled with openmp and MPI, it is suggested to use mpiprocs times ompthreads less than or equals to 24.**

Example5: Submit job for a MPI application for gpu queue. For demonstration purpose gromacs application is used.

Login node	
$	cat submit.pbs

```
$ cat submit.pbs
  -----------------------------------------------------------------
  #!/bin/bash
  #PBS -q gpu
  #PBS -l select=2:ncpus=24:mem=96G:ngpus=1:mpiprocs=2:ompthreads=12
  #PBS -l walltime=16:00:00
  #PBS -N gromacs_example
  module load gromacs/5.1.2/gcc493/impi_cuda
  cd $PBS_O_WORKDIR
  mpirun mdrun_mpi -pin on -ntomp $OMP_NUM_THREADS -s inputfile.tpr >mdrun.log
```

In the above script the resources are requested through the job script. Each line of the job script is explained below.

```
#PBS is the pbs directive

#PBS -q gpu
                  -> requesting for gpu queue

#PBS -l select=2:ncpus=24:mem=96G:ngpus=1:mpiprocs=2:ompthreads=12
                  -> Request 2 units of 24 CPUs, 96GB of memory, 1 gpu, and
12 threads with 2 MPI Processes

#PBS -l walltime=16:00:00
                  -> Request for a wall time of 16 Hours

#PBS -N gromacs_example
                  -> Name of the job is gromacs_example

module load gromacs/5.1.2/gcc493/impi_cuda
                  -> Load the environment for gromacs cuda version

cd $PBS_O_WORKDIR
                  -> Change to current working directory

mpirun mdrun_mpi -pin on -ntomp $OMP_NUM_THREADS -s inputfile.tpr >mdrun.log
                  -> Execute mpi version of gromacs command
```

*** To submit the job use the qsub syntax in example 3**

*** In this command notice mpirun does not have any arguments as PBS Pro is tightly integrated with mpirun command**

*** In case of application enabled with openmp and MPI, it is suggested to use mpiprocs times ompthreads less than or equals to 24.**

Example 6: Array Job

To maximize the usage of a serial application it is useful to submit array jobs. Array jobs are usually index driven where the index number is generated by PBS. The index number can be used as an input argument for the input file/application to spawn multiple jobs to run parallel.

Login node

```
$ cat submit.pbs
  ---------------------------------------------------------------------
  #!/bin/bash
  #PBS -q normal
  #PBS -l select=1:ncpus=24:mem=96G
  #PBS -l walltime=16:00:00
  #PBS -N Array_Job
  #PBS -J 1-10
  echo "Main script: index " $PBS_ARRAY_INDEX
  /opt/AppA -input /home/user01/runcase1/scriptlet_$PBS_ARRAY_INDEX
```

The jobs script is explained below

```
#PBS -> this is the PBS directive
#PBS -q normal
                  -> selecting queue as normal

#PBS -l select=1:ncpus=24:mem=96G
                  -> for this job 1 unit of 24 cpus and 96GB of memory

#PBS -l walltime=16:00:00
                  -> Requested wall time is 16 hours

#PBS -N Array_Job
                  -> Specify the name of the job is Array_job

#PBS -J 1-10
                  -> Array index is defined from 1 to 10

echo "Main script: index " $PBS_ARRAY_INDEX
                  -> for debug purpose echo $PBS_ARRAY_INDEX variable

/opt/AppA -input /home/user01/runcase1/scriptlet_$PBS_ARRAY_INDEX
                  -> execute the application APPA with requires the input
file suffix with $PBS_ARRAY_INDEX
```

*** To submit the job use the qsub syntax in example 3**

3.3 PBS Pro Job Environment variables:

While PBS jobs are executing there are few variables that are defined by PBS which can be used during the job execution. Below table provide list of few such variables.

Variable	Description
PBS_JOBID	The identifier assigned to the job
PBS_JOBNAME	The job name supplied by the user
PBS_NODEFILE	The filename containing a list of vnodes assigned to the job
PBS_O_PATH	Value of the PATH from submission environment
PBS_O_WORKDIR	The absolute path of directory where qsub is to execute
TMPDIR	The job-specific temporary directory for this job

Environment variables can used in Array jobs

Variable	Used for	Description
PBS_ARRAY_INDEX	Subjobs	The identifier assigned to the job
PBS_ARRAY_ID	Subjobs	The job name supplied by the user
PBS_JOBID	Jobs, Subjobs	The filename containing a list of vnodes assigned to the job

3.4 Querying job

Once the job is submitted to PBS, and wanted to know the status of the job, the command that need to be used is **"qstat".** The **`qstat`** command shows the lists the jobs that are currently running or jobs that are in the queue for execution.

Information about the jobs that finished more than 24 hours ago is not displayed in the output of qstat command.

Syntax: qstat <-a, -n [-1], -s [-1], -f ,-x, -w, -r, -l, ..>

To view jobs that are running or in the queue use

Login node

```
$ qstat
  --------------------------------------------------------------------
  Job id                  Name              User              Time Use S Queue
  ---------------------   ----------------  ----------------  -------- - -----
  161201.wlm01            STDIN             fsg5              00:00:00 R dev
```

In the above example, we can see a list of jobs that are currently being executed which are submitted by me.

All other users' jobs are hidden and so to get the list of other users' jobs use the command **"gstat".**

Another example is demonstrated below. The argument -H provides more information such as nodes, tasks, and memory requested.

Login node

```
$ qstat -H
  ---------------------------------------------------------------------------------
                                                            Req'd  Req'd   Elap
  Job ID          Username Queue    Jobname    SessID NDS TSK Memory Time  S Time
  --------------- -------- -------- ---------- ------ --- --- ------ ----- - -----
  113965.wlm01    fsg5     medium   Sleep        9440   8 192  512gb 24:00 F 00:00
  113966.wlm01    fsg5     medium   mpitest     18335   8 192  512gb 24:00 F 00:00
  113999.wlm01    fsg5     medium   GMX          9600   8 192  512gb 24:00 F 00:00
```

To view the jobs regardless of the job state use the below example:

Login node

```
$ qstat -x
  ---------------------------------------------------------------------------------
  Job id                 Name              User              Time Use S Queue
  ---------------------  ----------------  ----------------  -------- - -----
  100948[].wlm01         amber89           fsg5                     0 B long
  104843[].wlm01         submitTCGAediti   fsg5                     0 B small
  135905.wlm01           ipmi-reboot.pbs   fsg5              00:00:00 F dev
```

Login node

```
144581.wlm01          CFS              fsg5                574:49:0 F small
158001.wlm01          clean_HK-pfg052  fsg5                       0 H dev
154264.wlm01          aa               fsg5                       0 H gpunormal
104720.wlm01          parallelSearch.  fsg5                       0 H medium
136070.wlm01          ipmi-reboot.pbs  fsg5                       0 Q dev
154047.wlm01          CY-gpu-p         fsg5                       0 Q gpunormal
159191.wlm01          nap_br2          fsg5                       0 Q long
127876.wlm01          U7-DDES          fsg5                       0 Q medium
160377.wlm01          mono_SnTe        fsg5                       0 Q small
161031.wlm01          STDIN            fsg5                00:00:33 R dev
137437.wlm01          D1LPSPep         fsg5                2172:18: R gpulong
154263.wlm01          aa               fsg5                1724:47: R gpunormal
161213.wlm01          STDIN            fsg5                00:00:00 R largemem
127563.wlm01          InvertedFoil     fsg5                2797:41: R long
112987.wlm01          QuantumEspresso  fsg5                       0 R medium
```

The output of the gstat command will be something similar to that shown below

Login node

```
$ Gstat
  ----------------------------------------------------------------------------------------
  Node States:

              free : 158
          job-busy : 1138
     job-exclusive : 0
           offline : 3
  ********************************************************************************
  Mon Aug  8 12:58:01 SGT 2016
                                             Req'd  Req'd   Elap
  Job ID           Queue     SessID NDS  TSK Memory Time  S Time
  ---------------- --------- ------ --- ----- ------ ----- - -----
  100948[].wlm01   long      --     1    24   105gb  120:0 B --
  100949[].wlm01   long      --     1    24   105gb  120:0 B --
  104720.wlm01     medium    --     66   66   586gb  02:00 H --
  104721.wlm01     medium    --     66   66   586gb  02:00 H --
  104722.wlm01     medium    --     66   66   586gb  02:00 H --
  104723.wlm01     medium    --     66   66   586gb  02:00 H --
  104724.wlm01     medium    --     66   66   586gb  02:00 H --
  104725.wlm01     medium    --     66   66   586gb  02:00 H --
  104727.wlm01     medium    --     66   66   586gb  02:00 H --
  104843[].wlm01   small     --     1    24   105gb  24:00 B --
  105116.wlm01     medium    --     66   66   586gb  02:00 H --
  105117.wlm01     medium    --     66   66   586gb  02:00 H --
  105118.wlm01     medium    --     66   66   586gb  02:00 H --
  105119.wlm01     medium    --     66   66   586gb  02:00 H --
  105120.wlm01     medium    --     66   66   586gb  02:00 H --
  105121.wlm01     medium    --     66   66   586gb  02:00 H --
  105122.wlm01     medium    --     66   66   586gb  02:00 H --
  105123.wlm01     medium    --     66   66   586gb  02:00 H --
  105124.wlm01     medium    --     66   66   586gb  02:00 H --
```

3.5 Job Dependencies

Users can specify dependencies between their jobs, such as:

- Specify order of execution
- Execute the next job only if previous job finished
- Place jobs on hold until a particular job starts or finishes

  Syntax: qsub -W depend=<type>:<arg_lists> <job ID> myjobscript.sh

For multiple dependencies, <arg_list> items are colon-separated

  Example: qsub -W depend=afterok:1.pbsworks:2.pbsworks myjobscript.pbs

3.6 Deleting jobs

When the job is in queued state or running state, it may be necessary to terminate the job. In case if the job need to be terminated, the command to be used is qdel.

Syntax:	qdel <job id>
Usage:	qdel 2.wlm01

To delete the job forcefully, -W argument need to be supplied to the qdel command.

Syntax:	qdel -W force <job id>
Usage:	qdel -W force 2.wlm01