Taiwania HPC System
User Operation Manual

Version	1.3
Updated on	2019/08/22

Change History

Version	Date	Change details	Changed by	Reviewed by	Approved by
0.1	8 th Feb 2018	Initial Version	Imura	Ishan	Yamada
1.0	28th Mar 2018	- Revised Chapter 2.3, 4.1, 5.2 - Added Chapter 6 Resource limit in login nodes	Imura	Ishan	Yamada
1.1	12th Apr 2018	- Modified Chapter 4.2.4 MPI parallel program - Modified Chapter 4.3.4 MPI parallel program - Added Chapter 4.4.2 MPI parallel CUDA program - Added Chapter 5.3.4 Specifying E-mail notification - Added Chapter 5.6 Creating and usind reservation	Yoshida	Ishan	Yamada
1.2	3rd Jul 2018	- Modify Job Script Example	Oscar	Oscar	Oscar
	25th Jul 2018	- Added ct160 queue	Oscar	Oscar	Oscar
1.3	28th Dec 2018	- Modify Chapter 5.1 Modified Job queue Added Queue policy	Oscar	Oscar	Oscar
	5th Jul 2019	- Modify Chapter 2.4.1 - Modify Chapter 4.2.1 - Modify Chapter 4.3.1 - Modify Chapter 2.3.1 - Remove Chapter 5.6 - Remove Chapter 6	Oscar	Oscar	Oscar
	22nd Aug 2019	- Modify Chapter 5.1 Modified Job queue	Oscar	Oscar	Oscar

Contents

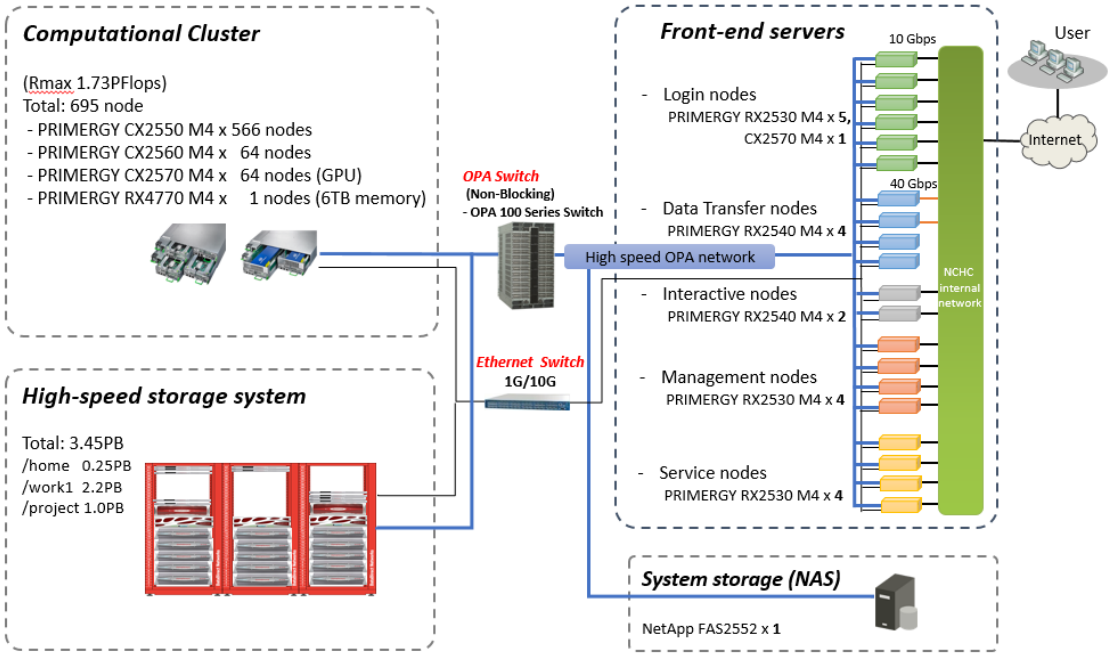
1. Introduction

In this User's Guide, we explain usage of the Peta HPC System installed at the National Center for High Performance Computing (NCHC). Please read this document carefully before using the system and get the latest version of the manual.

2. Peta HPC system

2.1. System overview

An overview of the general architecture of Peta HPC system is shown as below:



This HPC system consists of mainly 4 components:

1. Computational cluster
2. Front-end servers
3. High-speed storage system
4. System storage (NAS)

All these components are connected via Ethernet network and Intel Omni-Path high-speed network.

2.2. Available compute resources

There are 695 compute nodes (1392 processors and 27856 cores) used in Peta HPC system which in total delivers peak compute performance of approximately 1.73 PFLOPS. 694 compute nodes consist of dual CPU sockets, each socket comprises Xeon Gold 6148 CPU (20 cores, 2.4 GHz). 1 compute node (Big memory node) consists of quad sockets, each socket comprises Xeon Platinum 8160M CPU (24 cores, 2.1 GHz)
These 695 nodes can be categorized as below:

- **Thin Nodes**
  - For majority of HPC application
- **Fat nodes**
  - For HPC application which needs large memory
- **GPU nodes**
  - For HPC application which uses GPU accelerators
- **Big Memory node**
  - For special kinds of HPC application which needs very large memory

The summary of compute nodes and their respective resources are listed below:

Node Type	Node range	Total units (nodes)	Compute resources per unit (node)					
			CPU Sockets	CPU cores	Memory (GB)	Tesla P100	10Gbps interface	480 GB SSD
Thin nodes	cn0101 – cn0673	438	2	40	192	-	-	-
Thin nodes	cn0701 – cn0764	64	2	40	192	-	1	-
Fat nodes	cn0801 – cn0864	64	2	40	384	-	-	-
Fat nodes	cn0901 – cn0964	64	2	40	384	-	-	1
GPU nodes	cn1001 – cn1064	64	2	40	192	4	-	-
Big memory node	cnbm01	1	4	96	6000	-	-	-

2.3. Available storage resource

The following storage resources on high-speed storage system are available to users in this HPC system. They are mounted as lustre file system which is accessible from all the front end servers as well as from the compute nodes via the high-speed OPA network.

	Storage area	Mount point	Capacity
1	Home area	/home	0.25 PB
2	Temporary work area	/work1	2.2 PB
3	Project storage area	/project	1.0 PB

2.3.1. Home area

The total capacity of **0.25 PB** of home area is used by system users to store their private files. Users can compile their program and execute/manage their jobs in this home area. All users by default have 100GB of

quota in /home. Data under /home will be deleted only when our user officially sends us data removal request mail.

2.3.2. Temporary work area

This area has a total capacity of **2.2 PB** usable storage area. This area is primarily used for storing the active data of running jobs on the compute cluster.

All account holders by default have a /work1 disk space quota of 1.5 TB. This space on this clusters is intended for computing work and not for long term storage. In order to keep /work1 in a stable and efficient status, we will begin to regularly apply automated purge policy. There is no system backup for data in /work1, it is the user's responsibility to back up data. We cannot recover any data in /work1, including files lost to system crashes or hardware failure so it is important to make copies of your important data regularly. All inactive files that have not been written to or read within the last 28 days will be removed. We strongly urge users to regularly clean up their data in /work1 to decrease /work1 usage and to back up files you need to retain

e.g. data can be copied in a simple way from /work1 to /home or /project, using cp command as below.

```
[user@clogin1]$ cp /work1/<path to target file> /project/<destination path>
```

Some major options used with cp commands are:

-p Preserves modification times, access times, and modes from the original file.

-r Recursively copy entire directories.

2.3.3. Project storage area

To be announced (TBA).

2.4. Front-end servers for user access

Allow access from Taiwan IP addresses only.

2.4.1. Login nodes

```
140.110.148.11 clogin1.twnia.nchc.org.tw
140.110.148.12 clogin2.twnia.nchc.org.tw
140.110.148.15 glogin1.twnia.nchc.org.tw
```

There are three login nodes which are the primary access point for command line usage of the HPC service. Users access to the login nodes via each nodes' IP address. All login nodes are identically configured and no user data is kept on their disks. Users access their files on high-speed storage system which is mounted on each login node. Therefore it does not matter which login node a user is connected through to.

From login nodes, users are able to perform the following tasks:

- Submit/manage HPC jobs.
- Have full access to files resident on high-speed storage system.
- Compile HPC application.
- Run debugger for code development.

The login nodes have similar technical specification with compute nodes, and this is the reason why they provide complete compatibility for development and testing of application codes.

The summary of compute nodes and their respective resources are listed below:

Node Type	Node range	Total units (nodes)	Compute resources per unit (node)				
			CPU Sockets	CPU cores	Memory (GB)	Tesla P100	480 GB SSD
CPU login nodes	clogin1– clogin2	2	2	40	384	-	1
GPU login nodes	glogin1	1	2	40	192	4	-

To ensure optimal performance, process isolation and avoid situations where multiple processes unintentionally run on the same GPU, all GPUs on compute nodes are configured exclusive mode.

Do NOT use the login nodes for computation. If everyone does this, the login nodes will crash keeping other users from being able to login to this cluster.

2.4.2. Interactive nodes

To be announced (TBA).

2.4.3. Data transfer nodes

```
140.110.148.21 xdata1.twnia.nchc.org.tw
140.110.148.22 xdata2.twnia.nchc.org.tw
```

There are two data transfer nodes which are configured for transferring data from the external network in/out of the HPC system. Each node consists of a 40Gbps HCA card connected to external network, and an OPA interface connected to high-speed storage system as well as other nodes. By using this configuration, data can be rapidly transferred from/to the high-speed storage to/from users. For this purpose, users are allowed only scp/sftp access and cannot login to the HPC system via this nodes.

3. System access method

3.1. Member account & Peta system account registration

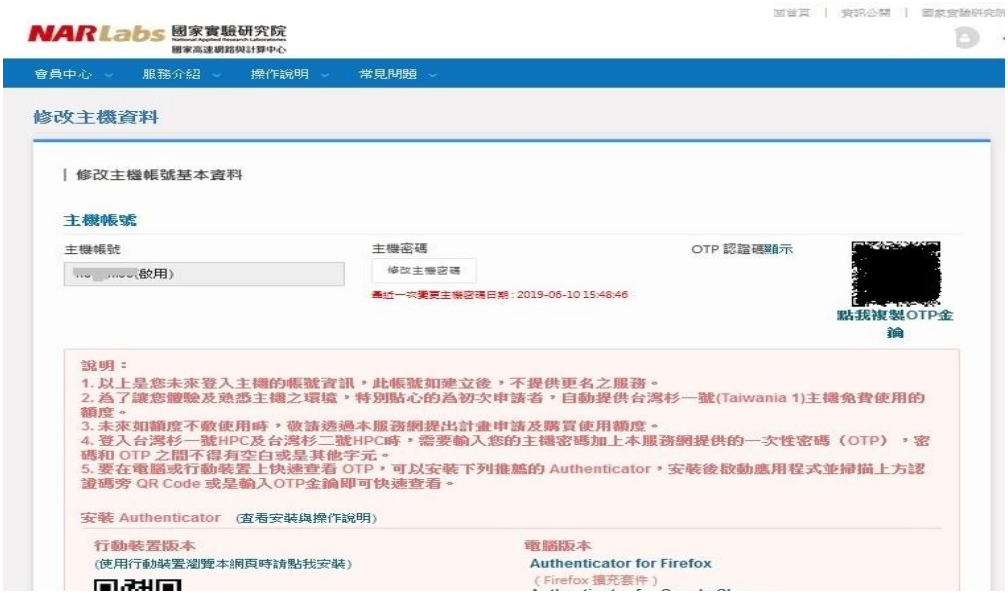
1. Registration website : https://iservice.nchc.org.tw/nchc_service/index.php

2. Member account apply now

3. Agreement confirmation

4. Fill in your basic information - setup your member account and password

5. Fill in your basic information - setup your Peta system account and password



6. E-mail account authentication

7. Select the [mobile authentication], dialog box to enter the SMS verification code received by the mobile phone.

8. Checking the project code by member account login

3.2. Command line login

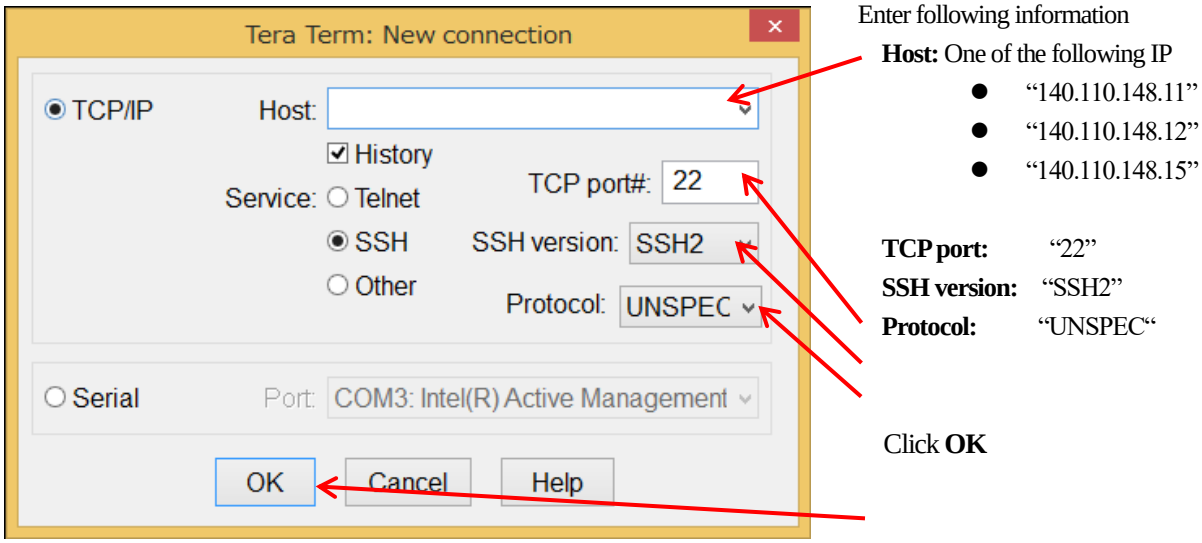
Users login to the system using issued account and One-Time Password. (OTP). Confirm that you have done the following before you start to remote access to the HPC system.

- Connect to member registration website to apply your login account and password for this system.
  Note: The password you get here is not OTP. It is an account password. The combination of account password and OTP will be used for login to the system.
- Get OTP QR code from login to member registration website.
- Download "SOPHOS authenticator" or "Google authenticator" apps in your smartphone.
- Scan the generated QR code using the downloaded apps and it will start to generate OTP once in every 30 second.

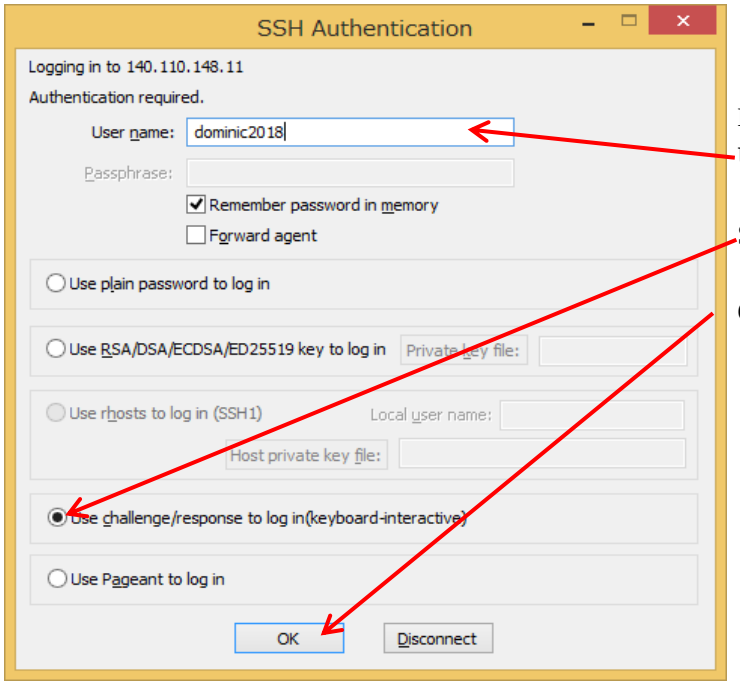
Incorrect time on your smartphone and PC can cause troubles. Please make sure your device has the right time.

Now, you can follow the steps below to access to the system using SSH client software such as TeraTerm from your PC.

1. Open SSH client software from your PC.
2. Input the Host IP address and port number.
   Note: The IP address below are directly accessible from anywhere in Taiwan. If you are outside Taiwan, you will not be able to access to the system.



3. Enter your account name in "username" box, select "keyboard-interactive" and click OK.



Enter following information

User name: issued username

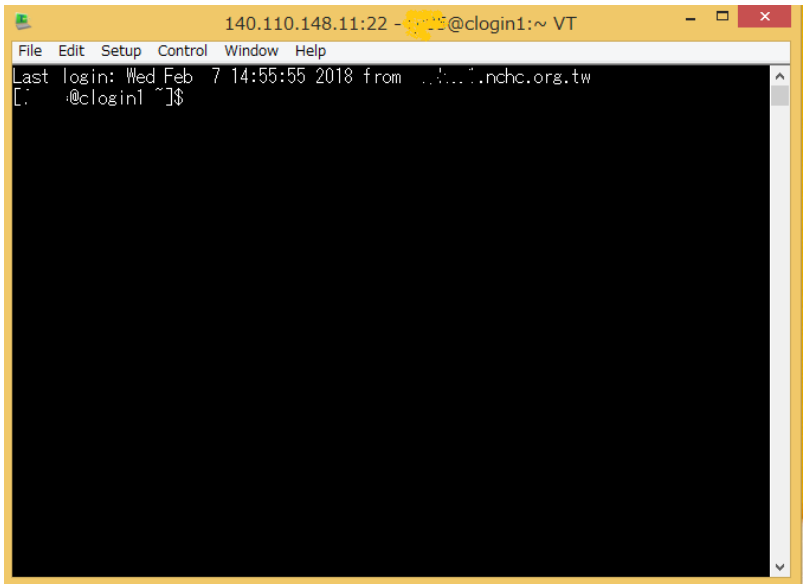
Select **keyboard-nteractive** mode

Click **OK** to access the system

4. Now, you will be asked to enter password. Enter your account password followed by OTP and press Enter.
   Password = Account password + OTP

On the first login to this system, a message about a key fingerprint is popped up. In this case, select "yes" and continue.

After the correct authentication, the login to any one login node will succeed as below.



3.3. Changing Password

If you need to change your account password, please go to member registration website.

3.4. Command line logout

Execute the logout or exit command.

```
[user@clogin1~]$ exit
```

3.5. File transfer

To transfer files from your PC or workstation to this system, use scp/sftp service. Linux/UNIX users use the scp or sftp command and Windows PC users use client software such as WinSCP.

3.5.1. Linux users

Use scp command and access to one of the data transfer nodes.

```
$ scp [option] <source host>:<local path of directory or file> ¥
<destination host>:<remote path of directory or file>
```

Some major options used with scp commands are:

-p Preserves modification times, access times, and modes from the original file.

-r Recursively copy entire directories.

Use sftp command and access to one of the data transfer nodes.

```
$ sftp [option] [username@]<destination host>
Connected to <destination host>.
sftp> get <remote path of directory or file>
                                        -> download file to local current directory
sftp> put <local path of directory or file>
                                        -> upload file to server current directory
sftp> bye                               -> quit sftp
```

Some major options used with sftp commands are:

-p Preserves modification times, access times, and modes from the original file.

-r Recursively copy entire directories.

Some major internal command used in sftp commands are:

cd <*path*> Change remote directory to <*path*>.

pwd Display remote working directory.

lcd <*path*> Change local directory to <*path*>.

lpwd Display local working directory.

3.5.2. Windows users

Start WinSCP and access to one of the data transfer node of the system. After the connection is established, you can transfer files just with drag & drop.

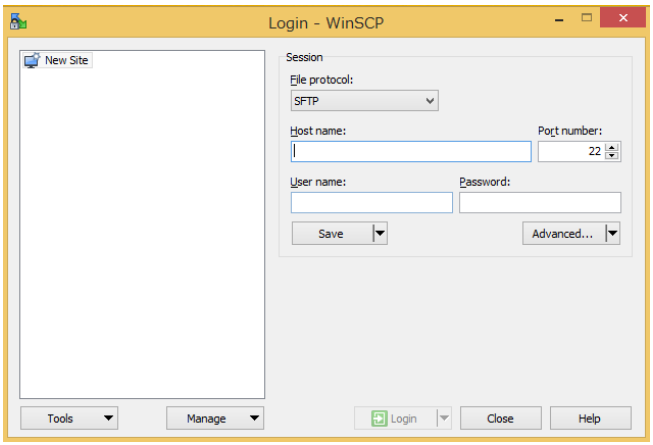
Below is the login window of WinSCP. Enter the following information and click "login" button.

Host name: "140.110.148.21" or "140.110.148.22"

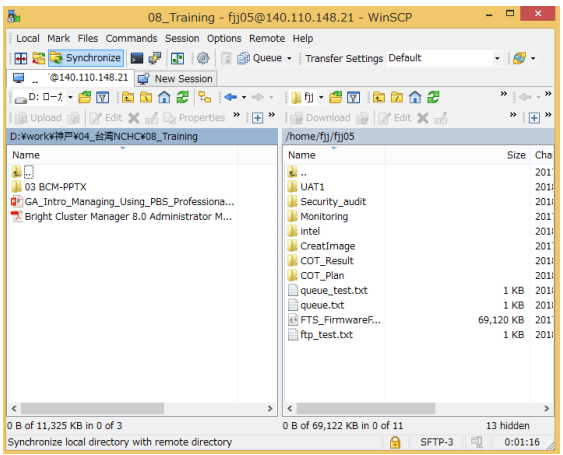
Port number: 22

User name: issued user name

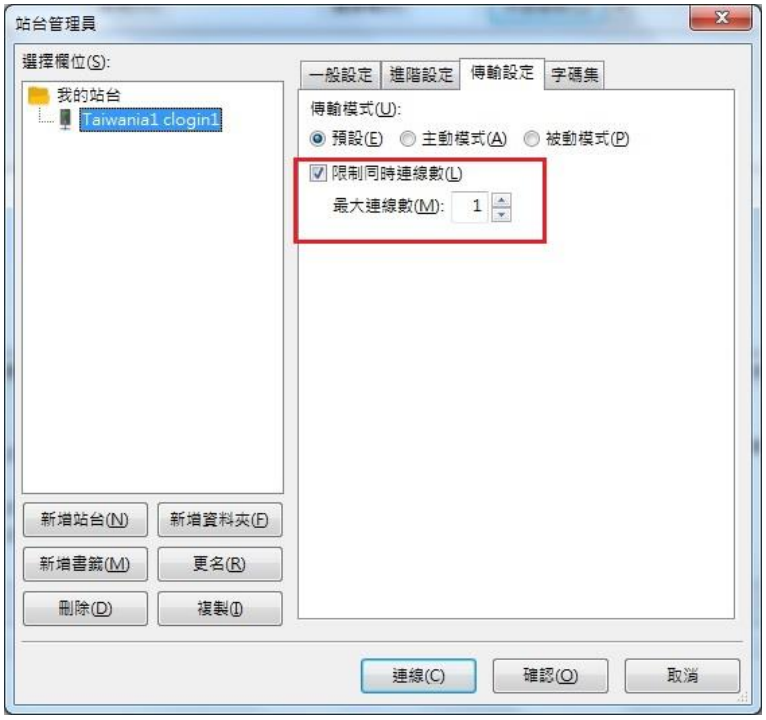
Password: issued username + one-time password



After the connection is established, WinSCP windows appears as below.



If you use FileZilla FTP client to transfer files, you have to go to Site Manager – Transfer Setting – Limit simultaneous connections to 1.



4. Compile and Link

4.1. Environment modules

The environment settings which are required for using the compiler, library, applications can be changed by using module commands.

1. Confirm the available module on login nodes

```
[user@clogin1]$ module avail
```

2. Load the module which are required for using the compiler, library, applications

```
[user@clogin1]$ module load <module name>
```

3. Add another module

```
[user@clogin1]$ module add <module name>
```

4. The following modules are available in this HPC system

Module Name	Description
blacs/openmpi/gcc/64/1.1patch03	Blacs library
blas/gcc/64/3.7.0	Basic Linear Algebra Subprograms for GNU

bonnie++/1.97.1	Bonnie++ library
cuda/8.0.61	CUDA library for GPU
fftw2/openmpi/gcc/64/double/2.1.5	FFTW library
fftw2/openmpi/gcc/64/float/2.1.5	FFTW library
fftw3/openmpi/gcc/64/3.3.6	FFTW library
gdb/7.12.1	GNU Cross Compilers
hdf5/1.10.1	Hierarchical Data Format
hwloc/1.11.6	Hardware Locality
intel/2017_u4	Intel Parallel Studio XE 2017 update 4
intel/2018_init	Intel Parallel Studio XE 2018 Initial
intel/2018_u1	Intel Parallel Studio XE 2018 update 1
iozone/3_465	File system benchmark tool
lapack/gcc/64/3.7.0	Linear Algebra package
mvapich2/gcc/64/2.2rc1	MVAPICH MPI library
netcdf/gcc/64/4.6.0	Network Common Data Form library
netperf/2.7.0	Network benchmark
openmpi/gcc/64/1.10.3	GCC compiled OpenMPI
openmpi/pgi/2.1.2/2017	PGI compiled OpenMPI
petsc/openmpi/gcc/3.8.0	PETSc data structure library
pgi/17.10	PGI compilers and development tools
scalapack/openmpi/gcc/64/2.0.2	Scalable Linear Algebra Library

Note: `openmpi/gcc` and `openmpi/pgi` have conflict setting. So, you cannot load both module at a time. Similarly, `intel/2017_u4`, `intel/2018_init` and `intel/2018_u1` also have conflict setting. So, you can load only one of these module at a time.

5. List all the modules currently loaded

```
[user@clogin1]$ module list
```

6. Unload the module

```
[user@clogin1]$ module unload <module name>
```

7. Unload all loaded modules

```
[user@clogin1]$ module purge
```

4.2 Intel Compiler

4.2.1 Loading compiler environment

1. Load Intel compiler environment

```
[user@clogin1]$ module load intel/2018_u1
```

Choose a module to match the version to use.

4.2.2 Serial program

1. Compile / link C program

```
[user@clogin1]$ icc –o sample.exe  sample.c
```

2. Compile / link C++ program

```
[user@clogin1]$ icpc –o sample.exe sample.c
```

3. Compile / link Fortran program

```
[user@clogin1]$ ifort –o sample.exe sample.f
```

4.2.3 Thread parallel program

1. Compile / link C program

```
[user@clogin1]$ icc -qopenmp –o sample_omp.exe sample_omp.c
```

2. Compile / link C++ program

```
[user@clogin1]$ icpc -qopenmp –o sample_omp.exe sample_omp.c
```

3. Compile / link Fortran program

```
[user@clogin1]$ ifort -qopenmp –o sample_omp.exe sample_omp.f
```

4.2.4 MPI parallel program

1. Build C source code called by MPI libary

```
[user@clogin1]$ mpiicc –o sample_mpi.exe sample_mpi.c
```

2. Build C++ source code called by MPI library

```
[user@clogin1]$ mpiicpc –o sample_mpi.exe sample_mpi.c
```

3. Build Fortran source code called by MPI library

```
[user@clogin1]$ mpiifort –o sample_mpi.exe sample_mpi.f
```

4. Job script example for running parallel program compiled by Intel library.

```
$ vim example01.sh
#!/bin/bash
#PBS -P TRI654321
#PBS -N sample_job
#PBS -l select=2:ncpus=40:mpiprocs=4
#PBS -l walltime=00:30:00
#PBS -q ctest
#PBS -j oe
```

```
module load intel/2018_u1
cd ${PBS_O_WORKDIR:-"."}
export I_MPI_HYDRA_BRANCH_COUNT=-1
mpirun ./sample_mpi.exe
```

5. Please export the environment value `export I_MPI_HYDRA_BRANCH_COUNT= - 1` , before issue the mpirun.

4.3 PGI compiler

4.3.1 Loading compiler environment

1. Load PGI compiler environment

```
[user@clogin1]$ module load pgi/17.10
```

4.3.2 Serial program

1. Compile/link C program

```
[user@clogin1]$ pgcc -o sample.exe sample.c
```

2. Compile/link C++ program

```
[user@clogin1]$ pgc++ -o sample.exe sample.c
```

3. Compile/link Fortran program

```
[user@clogin1]$ pgfortran -o sample.exe sample.f
```

4.3.3 Thread parallel program

1. Compile/link C program

```
[user@clogin1]$ pgcc -mp -o sample_omp.exe sample_omp.c
```

2. Compile/link C++ program

```
[user@clogin1]$ pgc++ -mp -o sample_omp.exe sample_omp.c
```

3. Compile/link Fortran program

```
[user@clogin1]$ pgfortran -mp -o sample_omp.exe sample_omp.f
```

4.3.4 MPI parallel program

1. Load compiler environment

```
[user@clogin1]$ module load openmpi/pgi/2.1.2/2017
```

2. Compile/link C program

```
[user@clogin1]$ mpicc –o sample_mpi.exe sample_mpi.c
```

3. Compile/link C++ program

```
[user@clogin1]$ mpic++ –o sample_mpi.exe sample_mpi.c
```

4. Compile/link Fortran program

```
[user@clogin1]$ mpifort –o sample_mpi.exe sample_mpi.f
```

5. Job script example of running parallel program compiled by PGI library.
   Refer to Chapter 5 for detail about job script.

```
$ vim example01.sh
#!/bin/bash
#PBS -P TRI654321
#PBS -N sample_job
#PBS -l select=2:ncpus=40:mpiprocs=4
#PBS -l walltime=00:30:00
#PBS -q ctest
#PBS -j oe

module load pgi/17.10
module load openmpi/pgi/2.1.2/2017
cd ${PBS_O_WORKDIR:-"."}
mpirun ./sample_mpi.exe
```

4.4 Compile with CUDA

4.4.1 CUDA program

1. Load CUDA compiler environment

```
[user@glogin1]$ module purge
[user@glogin1]$ module load cuda/8.0.61
```

Please choose a module to match the version to use.

2. Compile/link CUDA C program

```
[user@glogin1]$ nvcc –o sample.exe sample.cu
```

The extension of the source file must be ".cu".

3. Compile/link CUDA C++ program

```
[user@glogin1]$ nvcc –o sample.exe sample.cu
```

The extension of the source file must be ".cu".

4. Compile/link CUDA Fortran program

```
[user@glogin1]$ module add pgi
```

```
[user@glogin1]$ pgfortran -o sample.exe sample.cuf
```

The extension of the source file must be ".cuf".

4.4.2 MPI parallel CUDA program

This is an example of compiling and running CUDA MPI program using OpenMPI. For example, there are two programs, `simpleMPI.cu` (CUDA source file) and `simpleMPI.cpp` (C++ source file) to be compiled.

Compiling:

1. Load OpenMPI and CUDA environment module

```
$ module load openmpi/gcc/64/1.10.4
$ module load cuda/8.0.61
```

2. Check that the modules are loaded as below.

```
$ module list
Currently Loaded Modulefiles:
  1) openmpi/gcc/64/1.10.4   2) cuda/8.0.61
```

3. Check that you have loaded correct `nvcc` and `mpic++` command.

```
$ which nvcc
/pkg/cuda/8.0.61/bin/nvcc
$ which mpic++
/usr/mpi/gcc/openmpi-1.10.4-hfi/bin/mpic++
```

4. Compile `simpleMPI.cu` using `nvcc`.

```
$ nvcc -c simpleMPI.cu -o simpleMPI_cuda.o
```

5. Compile `simpleMPI.cpp` using `mpic++`.

```
mpic++ -c simpleMPI.cpp -o simpleMPI_mpi.o
```

6. Check that the following files are created by the respective compilers

```
$ ls *.o
simpleMPI_cuda.o
simpleMPI_mpi.o
```

7. Link the above two files.

```
$ mpic++ -o simpleMPI simpleMPI_cuda.o simpleMPI_mpi.o -L/pkg/cuda/8.0.61/lib64 -lcudart
```

8. Confirm that the executable `simpleMPI` is created.

```
$ ls simpleMPI
simpleMPI
```

Creating job script:

1. There are 4 GPUs installed on each GPU nodes in this system. So, you need to create the following script at first before creating PBS job script which distributes GPU processes on different GPUs within a node. So, there will be maximum 4 GPU processes running on a GPU node.
   Note: You can also specify CUDA device in your CUDA program instead. If you have done so, there is no need to create the following script.
   Specify the above executable `simpleMPI` in the script below.

```
$ vim run_4gpu.sh
#!/bin/bash


#load cuda library
module load cuda/8.0.61
#location of Binary
EXEC_DIR=`pwd`
APP=$EXEC_DIR/simpleMPI
lrank=$OMPI_COMM_WORLD_LOCAL_RANK
case ${lrank} in
[0])
  export CUDA_VISIBLE_DEVICES=0
  $APP
  ;;
[1])
  export CUDA_VISIBLE_DEVICES=1
  $APP
  ;;
[2])
  export CUDA_VISIBLE_DEVICES=2
  $APP
  ;;
[3])
  export CUDA_VISIBLE_DEVICES=3
  $APP
  ;;
esac
```

2. Then call the above script `run_4gpu.sh` in your PBS job script as below. The details of the PBS job script will be explained in chapter 5.
   Remember to specify `mpiprocs=4`. This will only allocate 4 processes per node maximum. If you try to set a number more than 4, the job will not run.
   In the example below, the script starts 8 processes which will be allocated to two different nodes, each nodes with 4 processes.

```
$ vim go.sh
```

```
#!/bin/bash
#PBS -P TRI654321
#PBS -N sample_cuda_mpi_job
#PBS -l select=2:ncpus=40:ngpus=4:mpiprocs=4
#PBS -l walltime=00:10:00
#PBS -q gp16
#PBS -j oe

module load openmpi/gcc/64/1.10.4
module load cuda/8.0.61
cd ${PBS_O_WORKDIR:-"."}

mpirun -np 8 -hostfile $PBS_NODEFILE --mca pml cm --mca mtl psm2 ./run_4gpu.sh
```

5. PBS Pro job operation

5.1 Job queue

Queue name	Resource range (CPU cores)	Memory per node	Resource range (GPUs)	Resource range (SSD)	Max walltime per job	High Priority	Max running jobs per user	Max running jobs
serial	1 (1 node)	384GB	–	1	96:00:00		10	
cf40	2-40 (1 node)	384GB	–	1	96:00:00		10	120
cf160	2-160 (1-4 nodes)	384GB	–		96:00:00		8	160
cf1200	161-1200 (5-30 nodes)	384GB	–		48:00:00	V	2	5
ct160	2-160 (1-4 nodes)	192GB	–		96:00:00		4	60
ct400	161-400 (5-10 nodes)	192GB	–		96:00:00		3	22
ct800	401-800 (11-20 nodes)	192GB	–		72:00:00		2	10
ct2k	801-2000 (21-50 nodes)	192GB	–		48:00:00	V	2	4
ct6k	2001-6000 (51-150 nodes)	192GB	–		24:00:00	V	1	2
ctest	1-80 (1-2 nodes)	192GB	–		00:30:00		10	
gtest	1-8 (1-2 nodes)	192GB	1-8		00:30:00		5	10
gp4	1-40 (1 node)	192GB	1-4		96:00:00		2	6
gp16	41-160 (2-4 nodes)	192GB	5-16		96:00:00		1	3
gp32	161-320 (5-8 nodes)	192GB	17-32		48:00:00		1	1
ct_ind	2-400				72:00:00	V		

gp_ind	1-16				72:00:00	V		

1. Users on this system can submit 50 jobs and use 6000 CPU cores at most; however, each queue has its own limitation of maximum number of running jobs. You jobs will be placed waiting in the queue when you reach the limit.
2. Some single node jobs such as Gaussian will benefit from using local SSD partition on a compute node as job-specific directory. Both cf40 and serial queues have a local SSD resource (max size 400GB). You can request this storage space by adding lscratch=?gb in select statement and access this temporary directory via the environment variable $PBS_JOBDIR in job script.
3. GPU resource are limited and we charge more than running pure CPU jobs. If your jobs will not access GPUs, do not assign gtest, gp4, gp16 queues in your job scripts. Otherwise, system administrators will terminate those jobs.
4. Please use the following resource combination when editing your GPU job scripts: ncpus=10:ngpus=1, ncpus=20:ngpus=2, ncpus=30:ngpus=3, or ncpus=40:ngpus=4
5. Both ct_ind and gp_ind are express queues and we demand extra charge for using them. After applying these two queues, your jobs will start sooner than others.

5.2 Queue List

```
$ qstat -Q
Queue              Max   Tot Ena Str   Que   Run   Hld   Wat   Trn   Ext Type
---------------- ----- ----- --- --- ----- ----- ----- ----- ----- ----- ----
serial               0     0 yes yes     0     0     0     0     0     0 Exec
cf40                 0     0 yes yes     0     0     0     0     0     0 Exec
cf160                0     0 yes yes     0     0     0     0     0     0 Exec
cf1200               0     0 yes yes     0     0     0     0     0     0 Exec
ct160                0     0 yes yes     0     0     0     0     0     0 Exec
ct400                0     0 yes yes     0     0     0     0     0     0 Exec
ct800                0     0 yes yes     0     0     0     0     0     0 Exec
ct2k                 0     0 yes yes     0     0     0     0     0     0 Exec
ct6k                 0     0 yes yes     0     0     0     0     0     0 Exec
ctest                0     0 yes yes     0     0     0     0     0     0 Exec
gp4                  0     0 yes yes     0     0     0     0     0     0 Exec
gp16                 0     0 yes yes     0     0     0     0     0     0 Exec
gp32                 0     0 yes yes     0     0     0     0     0     0 Exec
gtest                0     0 yes yes     0     0     0     0     0     0 Exec
```

5.3 Job submission

Before job submission, please make sure your Project ID (project name) has positive balance.

```
$ get_su_balance
```

```
   499023,TRI107693,試用計畫(ISSUE)
$ get_su_balance TRI107688
   -150
$ qsub testjob.sh
   qsub: No balance available for the User
```

5.3.1 PBS job script

A PBS job script consists of the following three components.

1. Shell specification
2. PBS directives
3. Programs or commands

e.g.

```
#!/bin/bash                                          -> Shell specification
#PBS -l walltime=1:00:00
#PBS -l select=2:ncpus=16:mpiprocs=16
#PBS -N sample_job
#PBS -q ctest                                        -> PBS directives
#PBS -P TRI654321
#PBS -j oe

cd $PBS_O_WORKDIR

module load intel/2018_u1                            -> Programs or commands
NODE=`cat $PBS_NODEFILE | wc`

mpirun ./myprogram
```

1. Shell specification:

The following line is added in the first line of a job script as shell specification.

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

2. PBS directives:

By specifying the PBS directives in a job script, users can set the job property.

Format:

```
#PBS -l <resource name>=<value>          -> specify resources
#PBS -N <job name>                       -> specify job name (optional)
#PBS -q <destination queue>               -> specify the queue
#PBS -P <project name>                   -> specify project name
#PBS -j eo                               -> merge std-err and std-out (optional)
```

e.g.

```
#PBS -l select=1:ncpus=1                    -> sequential job (1 core)
#PBS -l select=2:ncpus=8:mpiprocs=8         -> MPI job (2 nodes and 8 proc per node)
#PBS -l select=2:ncpus=8:mpiprocs=1:ompthreads=8
                                    -> MPI/OpenMP Hybrid job (2 MPI and 16 threads)
#PBS -l select=2:ncpus=40:ngpus=4:mpiprocs=4
                                             -> MPI/CUDA Hybrid job (4 gpus per node)
#PBS -l walltime=1:00:00                     -> processing wall time is one hour
```

Note: You have to specify correct resource (ncpus <= 40, ngpus <= 4 per node) limits for your job.

3. Program and command

The syntax of a job script is generally same as the syntax of a shell script.

```
cd $PBS_O_WORKDIR


module load intel/2018_u1
NODE=`cat $PBS_NODEFILE | wc`


mpirun ./myprogram
```

5.3.2 Batch job submission

PBS provides "qsub" command for submitting jobs. Batch jobs can be submitted by (a) job script, or pure (b) command line.

Format:

```
$ qsub <name of job script>
```

(a) Job script

1. Create the job script file.

```
$ vim example01.sh
#!/bin/bash
#PBS -P TRI107693
#PBS -N sample_job
#PBS -l select=2:ncpus=40:mpiprocs=40
#PBS -l walltime=00:30:00
#PBS -q ctest
#PBS -o jobresult.out
#PBS -e jobresult.err


module load intel/2018_u1
cd ${PBS_O_WORKDIR:-"."}


mpirun ./myprogram
```

2. Submit the job.

```
$ qsub example01.sh
```

(b) Command line

Users can specify the PBS directives from command line instead of specifying in a job script.

e.g.

```
$ qsub -l select=1:ncpus=1 -q ctest -P TRI654321 -j oe ./example01.sh
```

5.3.3 Array job (Bulk job) submission

Array is a feature of PBS which allows you to submit a series of jobs using a single submission command described by a single submission script. This feature is used when a large number of identical jobs which has similar inputs and outputs are to be submitted. To submit an array job, use -J option with "qsub" command.

Any normal jobs can be used in array job. Below is an example of a normal job which can be used in array job submission. There is no specific array variables to be used inside job script.

```
$ vim hello_mpi_1.sh
mpirun -np 80 /home/user/array/hello_mpi.exe
$ vim hello_mpi_2.sh
mpirun -np 80 /home/user/array/hello_mpi.exe
$ vim hello_mpi_3.sh
mpirun -np 80 /home/user/array/hello_mpi.exe

$ vim array.sh
#!/bin/bash
#PBS -l walltime=00:01:00
#PBS -l select=2:ncpus=4:mpiprocs=4
#PBS -N hello-mpi-array-job
#PBS -q ctest
#PBS -P TRI654321
#PBS -J 1-3
#PBS -j oe

echo "Main script: index " $PBS_ARRAY_INDEX
/home/user/array/hello_mpi_$PBS_ARRAY_INDEX.sh

$ qsub array.sh
```

The command below submits an array job whose sub-jobs are indexed from 1 to 100. It is similar to execute qsub command 100 times without -J option

```
$ qsub -J 1-100 example.sh
```

The command below submits an array job whose sub-jobs are indexed from 100 to 200 with step 2. i.e. 100.102.104 etc.

```
$ qsub -J 100-200:2 example.sh
```

5.3.4 Specifying E-mail Notification in a Job script

For each job, PBS can send email to designated recipients when that job reaches specific points in its lifecycle. There are two steps to realize this PBS functionality.

1. Use "-M" (capital M) option to set E-mail recipients in PBS directives as below.

```
#PBS -M user@example.com
```

2. Use "-m" (small m) option to specify mail point argument as below

```
#PBS -m be
```

Some of the important mail point arguments are listed below:

Mail point argument	Description
a	Send E-mail when job or subjob is aborted by batch system
b	Send E-mail when job or subjob begins execution
e	Send E-mail when job or subjob ends execution
n	Do not send E-mail

An example of specifying E-mail notification in a job script is as below

```
$ vim example01.sh
#!/bin/bash
#PBS -P TRI654321
#PBS -N sample_job
#PBS -l select=2:ncpus=40:mpiprocs=40
#PBS -l walltime=00:30:00
#PBS -q ctest
#PBS -j oe
#PBS -M user@example.com
#PBS -m be

module load intel/2018_u1
cd ${PBS_O_WORKDIR:-"."}

mpirun ./myprogram
```

5.4 Deleting a job

PBS provides "qdel" command for deleting jobs. Users can delete only your own job.

Format:

```
$ qdel <job ID>
```

e.g.

```
$ qdel 51
$ qdel 1234[].server
```

job ID can be confirmed by "qstat" command.

Users can forcefully delete an unfinished job using "-W force" option with qdel command.

```
$ qdel -W force <job ID>
```

5.5 Displaying job status

The "qstat" command is for watching the job status. There are 3 types in the statuses which are on S column.

(a) Job status: A job queued in queue "ctest"

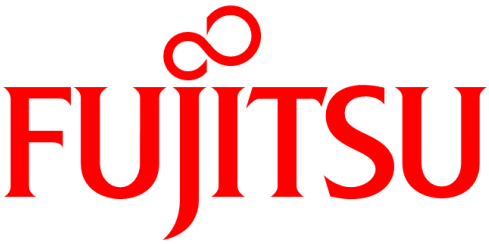
```
$ qstat -u user01
Job id                Name          User         Time Use S Queue
------------------------------------------------------------------------------
-----------------------------------
12.localhost           example01      user01             0 Q ctest
```

(b) Job status: Running

```
$ qstat -u user01
Job id                Name          User         Time Use S Queue
------------------------------------------------------------------------------
-----------------------------------
12.localhost           example01      user01             0 R ctest
```

(c) Job status: Completed

```
$ qstat -u user01
Job id                Name          User         Time Use S Queue
------------------------------------------------------------------------------
-----------------------------------
12.localhost           example01      user01        00:00:55 C ctest
```



shaping tomorrow with you