Transitioning to HPC cluster Prince (CGSB)

This page assumes you are familiar with Mercer cluster and its workload manager PBS.

Mercer, the cluster most of you are familiar with is in the process of being retired. Prince is the new HPC cluster, which is in production as of Spring 2017.

Introduction to Prince

![Diagram of Prince Cluster]

Compute Resources Mercer vs Prince

<table>
<thead>
<tr>
<th>Mercer Compute Nodes</th>
<th>Nodes</th>
<th>Cores /Node</th>
<th>CPU Type</th>
<th>Memory</th>
<th>Network</th>
<th>OS</th>
</tr>
</thead>
<tbody>
<tr>
<td>16</td>
<td>12</td>
<td>Westmere @2.67GHz</td>
<td>23 GB</td>
<td>QDR 40GB/8</td>
<td>CentOS 6.3</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>12</td>
<td>Westmere @2.67GHz</td>
<td>46 GB</td>
<td>QDR 40GB/8</td>
<td>CentOS 6.3</td>
<td></td>
</tr>
<tr>
<td>64</td>
<td>12</td>
<td>Westmere @2.67GHz</td>
<td>23 GB</td>
<td>QDR 40GB/8</td>
<td>CentOS 6.3</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>16</td>
<td>Sandy Bridge 2.0GHz (GPU)</td>
<td>126 GB</td>
<td>QDR 40GB/8</td>
<td>CentOS 6.3</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>20</td>
<td>Ivy Bridge @2.5GHz (GPU)</td>
<td>126 GB</td>
<td>FDR 56GB/8</td>
<td>CentOS 6.3</td>
<td></td>
</tr>
<tr>
<td>64</td>
<td>20</td>
<td>Ivy Bridge @3.0GHz</td>
<td>62 GB</td>
<td>FDR 56GB/8</td>
<td>CentOS 6.3</td>
<td></td>
</tr>
<tr>
<td>112</td>
<td>20</td>
<td>Ivy Bridge @3.0GHz</td>
<td>62 GB</td>
<td>QDR 40GB/8</td>
<td>CentOS 6.3</td>
<td></td>
</tr>
</tbody>
</table>
### Prince Compute Nodes

<table>
<thead>
<tr>
<th>Nodes</th>
<th>Cores /Node</th>
<th>CPU Type</th>
<th>Memory</th>
<th>Network</th>
<th>OS</th>
</tr>
</thead>
<tbody>
<tr>
<td>68</td>
<td>28</td>
<td>Broadwell @ 2.60GHz</td>
<td>125 GB</td>
<td>EDR</td>
<td>CentOS 7.2</td>
</tr>
<tr>
<td>32</td>
<td>28</td>
<td>Broadwell @ 2.60GHz</td>
<td>250 GB</td>
<td>EDR</td>
<td></td>
</tr>
<tr>
<td>32</td>
<td>20</td>
<td>Haswell @ 2.60GHz</td>
<td>62 GB</td>
<td>FDR</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>28</td>
<td>Broadwell @ 2.60GHz (GPU)</td>
<td>250 GB</td>
<td>EDR</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>28</td>
<td>Broadwell @ 2.60GHz (GPU)</td>
<td>126 GB</td>
<td>EDR</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>20</td>
<td>Ivy Bridge @ 2.5 GHz (GPU)</td>
<td>126 GB</td>
<td>FDR</td>
<td></td>
</tr>
<tr>
<td>64</td>
<td>20</td>
<td>Ivy Bridge @3.0 GHz</td>
<td>62 GB</td>
<td>FDR</td>
<td></td>
</tr>
<tr>
<td>112</td>
<td>20</td>
<td>Ivy Bridge @ 3.0 GHz</td>
<td>62 GB</td>
<td>QDR</td>
<td></td>
</tr>
<tr>
<td>48</td>
<td>20</td>
<td>Ivy Bridge @ 3.0 GHz</td>
<td>189 GB</td>
<td>QDR</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>20</td>
<td>Haswell @ 3.1 GHz</td>
<td>505 GB</td>
<td>FDR</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>20</td>
<td>Ivy Bridge @ 3.0 GHz</td>
<td>1490 GB</td>
<td>FDR</td>
<td></td>
</tr>
<tr>
<td>383</td>
<td>8716</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Filesystems on Prince

<table>
<thead>
<tr>
<th>Space</th>
<th>Environment variable</th>
<th>Purpose</th>
<th>Flushed?</th>
<th>Allocation/User</th>
</tr>
</thead>
<tbody>
<tr>
<td>/archive</td>
<td>$ARCHIVE</td>
<td>Long-term storage</td>
<td>NO</td>
<td>2 TB / 700 TB</td>
</tr>
<tr>
<td>/scratch</td>
<td>$SCRATCH</td>
<td>File staging - frequent writing and reading</td>
<td>YES. Files unused for 60 days are deleted</td>
<td>5 TB / 1.1 PB</td>
</tr>
<tr>
<td>/share/apps</td>
<td>N/A</td>
<td>Software directory</td>
<td>NO</td>
<td>N/A</td>
</tr>
<tr>
<td>/home</td>
<td>$HOME</td>
<td>Small files, code</td>
<td>NO</td>
<td>20 GB / 43 TB</td>
</tr>
<tr>
<td>/beegfs</td>
<td>$BEEGFS</td>
<td>File staging - frequent writing and reading</td>
<td>YES. Policy TBD</td>
<td>2 TB / 500 TB</td>
</tr>
</tbody>
</table>

*Fields in green indicate resources available on both Prince and Mercer*

For more details on Prince's hardware configuration, please click the following link: "Cluster - Prince".

### Connecting to Prince

Similarly to Mercer, if you are on NYU Network, you can connect to Prince login node prince.hpc.nyu.edu directly. If you are outside of campus, you first need to log in to bastion host hpc.nyu.edu/hpc2.nyu.edu and after that you will be able to connect to login node.
ITSs-Air-3:~ johd$ ssh prince.hpc.nyu.edu
johd@prince.hpc.nyu.edu's password:
[johd@log-0 ~]$ 

ITSs-Air-3:~ johd$ ssh johd@hpc.nyu.edu
 ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
WARNING: UNAUTHORIZED PERSONS DO NOT PROCEED  ~~~~~~~~~~~~~
This computer system is operated by New York University (NYU) and may be
accessed only by authorized users. Authorized users are granted specific,
limited privileges in their use of the system. The data and programs
in this system may not be accessed, copied, modified, or disclosed without
prior approval of NYU. Access and use, or causing access and use, of this
computer system by anyone other than as permitted by NYU are strictly pro-
hibited by NYU and by law and may subject an unauthorized user, including
unauthorized employees, to criminal and civil penalties as well as NYU-
initiated disciplinary proceedings. The use of this system is routinely
monitored and recorded, and anyone accessing this system consents to such
monitoring and recording. Questions regarding this access policy should be
directed (by e-mail) to askits@nyu.edu or (by phone) to 212-998-3333.
Questions on other topics should be directed to COMMENT (by email) or to
212-998-3333 by phone.
 ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
johd@hpc.nyu.edu's password:

[johd@hpc ~]$ ssh prince.hpc.nyu.edu
johd@prince.hpc.nyu.edu's password:
Last login: Sat Jan 14 11:45:42 2017 from hpc2.es.its.nyu.edu
[johd@log-1 ~]$ 

Slurm Job Scheduler

Prince cluster is using a new job Scheduler SLURM (Simple Linux Utility for Resource Management). Slurm is an open source, fault-tolerant, and very efficient cluster management system.

Basic Terminology

- **Partition** - a logical group of compute nodes. Can be considered a job queue, which has a set of constraints such as job size limit, job time limit, users permitted to run jobs on it, etc. Partitions may or may not overlap.

- **Job** - a unit of execution, started by user from a command line with sbatch, srun and salloc.

- **Step** - task within a job, created with srun command. If srun was invoked from a command line, this interactive job will have one step. If srun is called from batch script, each srun will be a new step in a job. They will run sequentially, although each step can start multiple parallel tasks.

- **Task** - a single process in a job. If job has steps, each step will have at least one task.

- **Array** - a set of similar jobs submitted with the same initial requirement automatically and almost simultaneously.

Translation from PBS to Slurm

<table>
<thead>
<tr>
<th>User Commands</th>
<th>PBS/Torque</th>
<th>Slurm</th>
</tr>
</thead>
<tbody>
<tr>
<td>Job submission</td>
<td>qsub [script_file]</td>
<td>sbatch [script_file]</td>
</tr>
<tr>
<td>Job deletion</td>
<td>qdel [job_id]</td>
<td>scancel [job_id]</td>
</tr>
<tr>
<td>Job status (by job)</td>
<td>qstat [job_id]</td>
<td>squeue [job_id]</td>
</tr>
<tr>
<td>--------------------------</td>
<td>---------------</td>
<td>----------------</td>
</tr>
<tr>
<td>Job status (by user)</td>
<td>qstat -u [user_name]</td>
<td>squeue -u [user_name]</td>
</tr>
<tr>
<td>Queue list</td>
<td>qstat -Q</td>
<td>squeue</td>
</tr>
<tr>
<td>Node list</td>
<td>pbsnodes -l all</td>
<td>sinfo -N OR scontrol show nodes</td>
</tr>
<tr>
<td>Cluster status</td>
<td>qstat -a</td>
<td>squeue</td>
</tr>
<tr>
<td>Graphical cluster status</td>
<td>pbsstop</td>
<td>slurmtop</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Environment</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Starting directory on compute node</td>
<td>User's home directory</td>
<td>the working (submit) directory</td>
</tr>
<tr>
<td>Job ID</td>
<td>$PBS_JOBID</td>
<td>$SLURM_JOBID</td>
</tr>
<tr>
<td>Submit Directory</td>
<td>$PBS_O_WORKDIR</td>
<td>$SLURM_SUBMIT_DIR</td>
</tr>
<tr>
<td>Submit Host</td>
<td>$PBS_O_HOST</td>
<td>$SLURM_SUBMIT_HOST</td>
</tr>
<tr>
<td>Node List</td>
<td>$PBS_NODEFILE</td>
<td>$SLURM_JOB_NODELIST</td>
</tr>
<tr>
<td>Job Array Index</td>
<td>$PBS_ARRAYID</td>
<td>$SLURM_ARRAY_TASK_ID</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Job Specification</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Script directive</td>
<td>#PBS</td>
<td>#SBATCH</td>
</tr>
<tr>
<td>Job Name</td>
<td>-N [name]</td>
<td>--job-name=[name]</td>
</tr>
<tr>
<td>Node Count</td>
<td>-l nodes=[count]</td>
<td>-N [min[-max]]</td>
</tr>
<tr>
<td>CPU Count</td>
<td>-l ppn=[count] OR -l mppwidth=[PE_count]</td>
<td>-n [count]</td>
</tr>
<tr>
<td>Memory Size</td>
<td>-l mem=[MB] OR --mem=[mem][M</td>
<td>G</td>
</tr>
<tr>
<td>Wall Clock Limit</td>
<td>-l walltime=[hh:mm:ss]</td>
<td>-t [min] OR -t [days-hh:mm:ss]</td>
</tr>
<tr>
<td>Standard Output File</td>
<td>-o [file_name]</td>
<td>-o [file_name]</td>
</tr>
<tr>
<td>Standard Error File</td>
<td>-e [file_name]</td>
<td>-e [file_name]</td>
</tr>
<tr>
<td>Combine stdout/err</td>
<td>-j oe (both to stdout) OR -j eo (both to stderr)</td>
<td>(use -o without -e)</td>
</tr>
<tr>
<td>Tasks Per Node</td>
<td>-l mppnppn [PEs_per_node]</td>
<td>--tasks-per-node=[count]</td>
</tr>
<tr>
<td>CPUs Per Task</td>
<td>N/A</td>
<td>--cpus-per-task=[count]</td>
</tr>
</tbody>
</table>
Submit jobs - [sbatch]

Batch job submission can be accomplished with the command sbatch. Like in Torque qsub, we create a bash script to describe our job requirements: what resources we need, what software we want to run, how much memory and how many CPUs we need, and where to send job standard output, error etc. After a job is submitted, Slurm will find the suitable resources, schedule and drive the job execution, and report outcome back to the user. The user can then return to look at the output files.

Converting a PBS batch script to SLURM batch script is generally a straightforward process. Below is the comparison between different options specified in Slurm batch script and in PBS batch script:

```bash
#!/bin/sh
#
PBS --M $USER@nyu.edu
PBS -m e
PBS -N JobName
PBS -o JobName=$PBS_JOBID.out
PBS -e JobName=$PBS_JOBID.err
PBS -l walltime=01:00:00
PBS -l nodes=1:ppn=1
PBS -l mem=1GB

/bin/hostname
/bin/pwd

module purge
module load r/intel/3.3.1
Rscript --e 'V <- c(1,3,5,7); print(V)'

#!/bin/sh
#
SBATCH --mail-user=$USER@nyu.edu
SBATCH --mail-type=END
SBATCH --job-name=JobName
SBATCH --output=JobName-%j.out
SBATCH --error=JobName-%j.err
SBATCH --time=01:00:00
SBATCH --nodes=1
SBATCH --ntasks=1
SBATCH --mem=1GB

/bin/hostname
/bin/pwd
sleep 15
module purge
module load r/intel/3.3.2
srun Rscript --e 'V <- c(1,3,5,7); print(V)'
You can find a full list of options for job specification if you view sbatch man page:

```
$ man sbatch
```

**Monitor cluster/job status - [sinfo, squeue,sstat,sacct]**

The `sinfo` command gives information about the cluster status, by default listing all the partitions. Partitions group computing nodes into logical sets, which serves various functionalities such as interactivity, visualization and batch processing.

```
$ sinfo
PARTITION  AVAIL  TIMELIMIT  NODES  STATE  NODELIST
  c01_25*    up    00:00:00    4    mix  c13-[01-04]
  c01_25*    up    00:00:00    95    idle  c01-[01-04],c02-[01-04],c03-[01-04],c04-[01-04],c05-[01-04],c06-[01-04],c07-[01-04]...
  c26        up    00:00:00    16    idle  c26-[01-16]
  c27        up    00:00:00    16    idle  c27-[01-16]
  gpu        up    00:00:00    2    mix  gpu-[01-02]
  gpu        up    00:00:00    7    idle  gpu-[03-09]
```

`sinfo` by default prints information aggregated by partition and node state. As shown above, there are 4 partitions namely c01_25, c26, c27 and gpu. The partition marked with an asterisk is the default one. Except there are two lines with the node state 'mix', which means some CPU cores occupied, all other nodes are idle. Some of the common node states are listed in the table below:

<table>
<thead>
<tr>
<th>Node State</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>allocated</td>
<td>The node has been allocated to one or more jobs</td>
</tr>
<tr>
<td>completing</td>
<td>All jobs associated with this node are in the process of COMPLETING</td>
</tr>
<tr>
<td>down</td>
<td>The node is unavailable for use</td>
</tr>
<tr>
<td>drained</td>
<td>The node is unavailable for use per system administrator request</td>
</tr>
<tr>
<td>idle</td>
<td>The node is not allocated to any jobs and is available for use</td>
</tr>
<tr>
<td>mixed</td>
<td>The node has some of its CPUs ALLOCATED while others are IDLE</td>
</tr>
<tr>
<td>reserved</td>
<td>The node is in an advanced reservation and not generally available</td>
</tr>
</tbody>
</table>

The `squeue` command lists jobs which are in a state of either running, or waiting or completing etc. It can also display jobs owned by a specific user or with specific job ID.

```
$ squeue
JOBID  PARTITION  NAME       USER    ST    TIME  NODES  NODELIST(REASON)
  9874    c01_25 model_ev  johd   R     17:00   4 c13-[01-04]
  9868    gpu relases - xh814  R    17:45:45   1 gpu-01
  9869    gpu amberGPU       xh814  R   1:30:19   1 gpu-01
  9873    gpu pemed_1        johd   R    17:08   1 gpu-02
```

```
$ squeue -u johd
JOBID  PARTITION  NAME       USER    ST    TIME  NODES  NODELIST(REASON)
  9874    c01_25 model_ev  johd   R     22:19   4 c13-[01-04]
```

```
$ squeue -j 9877
JOBID  PARTITION  NAME       USER    ST    TIME  NODES  NODELIST(REASON)
  9877    gpu pemed_1        johd   R      0:10   1 gpu-02
```
Most of the columns in the output of the `squeue` command are self-explanatory.

The column "ST" in the middle is the job status, which can be:

- **PD** - pending: waiting for resource allocation
- **S** - suspended
- **R** - running
- **F** - failed: non-zero exit code or other failures
- **CD** - completed: all processes terminated with zero exit code
- **CG** - completing: in the completing process, some processes may still be alive

The column "NODELIST(REASON)" in the end is job status due to the reason(s), which can be:

- JobHeldUser: (obviously)
- Priority: higher priority jobs exist
- Resources: waiting for resources to become available
- BeginTime: start time not reached yet
- Dependency: wait for a depended job to finish
- QOSMaxCpuPerUserLimit: number of CPU core limit reached

Run the command `sstat` to display various information of running job/step. Run the command `sacct` to check historical/accounting information of jobs and job steps in the Slurm log or database. There is a `--helpformat` option in these two commands to help checking what output columns are available.

```bash
$ sstat -j 23221 -o JobID,Nodelist,Pids,MaxRSS,AveRSS,MaxVMSize
JobID             Nodelist                 Pids     MaxRSS     AveRSS  MaxVMSize
------------ -------------------- -------------------- ---------- ---------- ----------
23221.0                    c03-04               158503   3462088K   2681124K   8357328K

$ sacct -j 7050 --format JobID,jobname,NTasks,nodelist,MaxRSS,MaxVMSize,AveRSS,AveVMSize
JobID    JobName   NTasks        NodeList     MaxRSS  MaxVMSize     AveRSS  AveVMSize
------------ ---------- -------- --------------- ---------- ---------- ---------- ----------
7050         mpiexec-t+              c17-[01-03]    0      4316K          0      4316K
7050.extern      extern        3     c17-[01-03]          0      4316K          0      4316K
7050.0            orted        2     c17-[02-03]    141016K    370880K    140024K    370868K
```

**Cancel a job - [scancel]**

Things can go wrong, or in an unexpected way. Should you decide to terminate a job before it finishes, `scancel` is the tool to help.

```bash
$ squeue -j 9877
 JOBID PARTITION NAME     USER ST    TIME NODES NODELIST(REASON)
9877    gpu  pemed_1  johd  R    9:04    1    gpu-02

$ scancel 9877
```

**Running Interactive Jobs**

You can run interactive job with Slurm using `srun` command:
Running array jobs

Using job array you may submit many similar jobs with almost identical job requirement. This makes life easier for users and for the scheduler system. Job array can only be used in batch jobs. Usually the only requirement difference among jobs in a job array is the input file or files.

The example below describes running an array of read alignments jobs with bowtie. First, we create a batch script:

```bash
#!/bin/bash
#SBATCH --array=0-2
#SBATCH --output=out_alignreads.%A_%a
#SBATCH --error=err_alignreads.%A_%a
#SBATCH --job-name=b2_alignreads
#SBATCH --nodes=1
#SBATCH --cpus-per-task=3
#SBATCH --mem=10G
#SBATCH --time=01:00:00

module purge
module load bowtie2/intel/2.2.9

FILES=($(ls *.fastq))

INPUT=${FILES[$SLURM_ARRAY_TASK_ID]}
OUTPUT=${INPUT}.sam

srun bowtie2 --phred64 -x Athaliana -p $SLURM_CPUS_PER_TASK -U $INPUT -S $OUTPUT
```

And submit it with sbatch command:

```
$ sbatch array.sh
```