Monitoring batch jobs at Prince

Quick Links

HPC Home
Getting an account
Gentle Introduction to using HPC
Getting started on Prince
Prince How-to Articles
Logging in
  Windows
  Mac / Linux
Clusters and Storage
  Prince (HPC)
  Dumbo (Hadoop)
  Brooklyn (OpenStack)
  Dalma (NYU Abu Dhabi)
Transferring data to/from the clusters
Transferring data to/from Prince cluster using Globus
Submitting jobs with sbatch
Available software
Licensed Software
Available on the HPC Cluster
Building
Software
Slurm Tutorial
Tutorials
FAQs
Scratch Area
Cleanup
Programming
for Biologists
Acknowledge
Statement
Research Gallery
HPC People
HPC Policies
(Tip: click "<<" at bottom left to close Confluence sidebar)
Running jobs on the Prince Cluster

Accessing the Prince Cluster
- From Windows workstation
- From Mac workstation

Software and Environment Module
- Job script and resource request

Module Job script and resource request
- Introduction to job scheduling
- Submitting jobs with `sbatch`
- Requesting resources
- Using computing nodes interactively

Monitoring batch jobs
- Monitoring batch jobs - `squeue`
- What is running and where? `slurmtop`

Canceling your jobs

Compiling your own software

Putting all pieces together
- An Amber example
- A R example

Summary

Where is my job in the queue - or on the system - and why?

The simplest queuing algorithm is "first come, first served". The queuing of jobs on the HPC cluster is a little more sophisticated as we pursue several goals:

- Minimal queuing times, especially for short jobs. Nobody wants to spend 4 hours in the queue for a 1-hour job.
- Efficient use of the available resources. If there is a job ready which can use hardware that would otherwise be idle, run it, even if it's not next in the queue.
- Fair use of resources. If you've made heavy use of the cluster recently, jobs belonging to a user who has had less CPU time will get higher priority.
  At NYU "recently" means "the last 24 hours", so users with large workloads are not excessively penalized.

![Should you need more resources than the fair share allocations because of critical deadlines such as a grant application, a publication deadline, or class use, please email hpc@nyu.edu to make special arrangements.]

- Special consideration for HPC Stakeholders. NYU HPC uses a "condo" model in which we manage HPC resources owned by specific schools and departments, in exchange for allowing the rest of the NYU HPC community to use those resources when the owners are not.
Slurm supports these goals by calculating a priority for each submitted job and placing the job in the queue according to its priority. The schedule of which job will run where and when is built from the job queues. When a job finishes earlier than scheduled (due to an overestimated walltime request), Slurm attempts to fill the newly-available space by scanning the queue for the first job which will fit without delaying an already-scheduled, higher priority job. In this way low-priority jobs with smaller resource requirements can jump ahead and be run early.

You can take advantage of this by requesting CPU, walltime and memory resources as accurately as possible. Be careful not to request too little though, or your job may exceed the request and be killed.

### Monitoring jobs with Slurm

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sinfo</td>
<td>Report the status of partitions, nodes and queues etc.</td>
</tr>
<tr>
<td>squeue</td>
<td>Report job and job step status in the scheduling queue</td>
</tr>
<tr>
<td>sacct</td>
<td>Report accounting information by individual job and job step</td>
</tr>
<tr>
<td>sstat</td>
<td>Report accounting information about currently running jobs and job steps</td>
</tr>
<tr>
<td>scontrol</td>
<td>Administrator tool to view and/or update system, job, step, partition or reservation status</td>
</tr>
</tbody>
</table>

#### To see the status of a single job - or a list of specific jobs - pass the Job IDs to squeue, as in the following example:

```
$ squeue -j 7037,7039
```

<table>
<thead>
<tr>
<th>JOBID</th>
<th>PARTITION</th>
<th>NAME</th>
<th>USER</th>
<th>ST</th>
<th>TIME</th>
<th>NODES</th>
<th>NODELIST(REASON)</th>
</tr>
</thead>
<tbody>
<tr>
<td>7037</td>
<td>gpu</td>
<td>wrap</td>
<td>ab33</td>
<td>R</td>
<td>3:25</td>
<td>1</td>
<td>gpu-01</td>
</tr>
<tr>
<td>7039</td>
<td>c01_25</td>
<td>dathg</td>
<td>ab33</td>
<td>R</td>
<td>1:30</td>
<td>1</td>
<td>c17-01</td>
</tr>
<tr>
<td>7043</td>
<td>c01_25</td>
<td>wscr</td>
<td>ed35</td>
<td>PD</td>
<td>0:00</td>
<td>1</td>
<td>(ReqNodeNotAvail, UnavailableNodes:c23-03)</td>
</tr>
<tr>
<td>7042</td>
<td>c01_25</td>
<td>nmad</td>
<td>fd48</td>
<td>PD</td>
<td>0:00</td>
<td>1</td>
<td>(JobHeldUser)</td>
</tr>
</tbody>
</table>

Most of the fields in the output 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
- **ReqNodeNotAvail**: requested node maybe down, in use, reserved for other jobs
- **BeginTime**: start time not reached yet
- **Dependency**: wait for a depended job to finish

Other, less common job flags are described in the manual (`man squeue`).

To list all jobs owned by you or any specific friend, use:

```
$ squeue -u NetID
```

For example:

```
$ squeue -u $USER
```

<table>
<thead>
<tr>
<th>JOBID</th>
<th>PARTITION</th>
<th>NAME</th>
<th>USER</th>
<th>ST</th>
<th>TIME</th>
<th>NODES</th>
<th>NODELIST(REASON)</th>
</tr>
</thead>
<tbody>
<tr>
<td>7046</td>
<td>c01_25</td>
<td>data_cru</td>
<td>johd</td>
<td>CG</td>
<td>0:00</td>
<td>1</td>
<td>c13-01</td>
</tr>
<tr>
<td>7047</td>
<td>c01_25</td>
<td>data_cru</td>
<td>johd</td>
<td>R</td>
<td>0:01</td>
<td>4</td>
<td>c13-[02-04], c14-01</td>
</tr>
<tr>
<td>7045</td>
<td>c01_25</td>
<td>data_cru</td>
<td>johd</td>
<td>R</td>
<td>2:01</td>
<td>1</td>
<td>c17-01</td>
</tr>
</tbody>
</table>

In Unix, the shell sets an environment variable USER to your username (at NYU this is your NetID). In the example above this environment variable is used instead of explicitly typing my NetID.
While a job with steps is running, `sstat` can show its memory usage and a lot other information (run `sstat --helpformat` to see what other quantities available), e.g.

```
$ sstat -j 7051 --format JobID,NTasks,nodeList,MaxRSS,MaxVMSize,AveRSS,AveVMSize

<table>
<thead>
<tr>
<th>JobID</th>
<th>NTasks</th>
<th>NodeList</th>
<th>MaxRSS</th>
<th>Max VMSize</th>
<th>AveRSS</th>
<th>Ave VMSize</th>
</tr>
</thead>
<tbody>
<tr>
<td>7051.0</td>
<td>2</td>
<td>c17-[02-03]</td>
<td>141068K</td>
<td>370880K</td>
<td>139004K</td>
<td>370868K</td>
</tr>
</tbody>
</table>
```

If your job seems stuck in the queue, `scontrol` can give useful hints about why:

`scontrol` is mainly for administrators to view and/or update system, job, step, partition or reservation status. Users can also list detailed job and step information, e.g.

```
$ scontrol show job -dd 7051
JobId=7051 JobName=mpiexec-test
UserId=johd(2761180) GroupId=johd(1002761180) MCS_label=N/A
Priority=108683 Nice=0 Account=users QOS=normal
JobState=RUNNING Reason=None Dependency=(null)
Requeue=0 Restarts=0 BatchFlag=1 Reboot=0 ExitCode=0:0
DerivedExitCode=0:0
RunTime=00:00:03 TimeLimit=12:00:00 TimeMin=N/A
SubmitTime=2017-01-11T15:40:02 EligibleTime=2017-01-11T15:40:02
StartTIme=2017-01-11T15:40:02 EndTime=2017-01-12T03:40:03 Deadline=N/A
PreemptTime=None SuspendTime=None SecsPreSuspend=0
Partition=c01_25 AllocNode:Sid=log-0:153794
ReqNodeList=(null) ExcNodeList=(null)
NodeList=c17-[01-03]
BatchHost=c17-01
NumNodes=3 NumCPUs=84 NumTasks=12 CPUs/Task=7 ReqB:S:C:T=0:0:*:*
TRES=cpu=84,mem=360G,node=3
Socks/Node=* NtasksPerNode:S:C=4:0:*:* CoreSpec=*
  Nodes=c17-[01-03] CPU_IDs=0-27 Mem=122880
MinCPUsNode=28 MinMemoryNode=120G MinTmpDiskNode=0
Features=(null) Gres=(null) Reservation=(null)
OverSubscribe=OK Contiguous=0 Licenses=(null) Network=(null)
Command=/home/johd/test_slurm/openmpi/run-openmpi.sh
WorkDir=/home/johd/test_slurm/openmpi
StdErr=/home/johd/test_slurm/openmpi/slurm-7051.out
StdIn=/dev/null
StdOut=/home/johd/test_slurm/openmpi/slurm-7051.out
BatchScript=

#!/bin/bash
#
#SBATCH --job-name=mpiexec-test
#SBATCH --nodes=3
#SBATCH --tasks-per-node=4
#SBATCH --cpus-per-task=7
#SBATCH --time=12:00:00
#SBATCH --mem=120GB

module purge
module load openmpi/intel/2.0.1

cd /home/johd/test_slurm/openmpi
mpiexec .pi

Power=
```

**Exercise**

Request an interactive batch session on Prince cluster. When it starts (or immediately, in another window), use `squeue` to see it in the queue. Try `scontrol show job -dd <job_id>` too.