Copy of Running jobs - Scripts

This page is retained from an earlier version of the HPC wiki only for reference, and the equivalent up-to-date page is at Running jobs on the NYU HPC clusters.

(Too long, break it into pages)

things to cover here:

- note: scripting a job and scheduling a job are two slightly different topics. Scripting should have links to includes of where and how to move data about and how to load the required modules. Scheduling should cover interactive jobs too.

A third topic which relates to both is "checkpoint-restart" for long jobs - especially, if there is significant risk that your job will run out of time or resources.

And monitoring jobs.

- overview of the idea of batch scheduling
- what queues are available on the clusters and what are they for
  - (link to later section of "how to read pbstop" with diagram of what resources are available to what queues)
  - qstat -q
- what resource you need to specify and how to do so
- special jobs: increasing walltime beyond 48 hours
  - (talk to us first)

checkpoint-restart story: (anonymised version): imagine this scenario: you have a simulation to run, but don't know exactly how long it will take. Extrapolating from a shorter run suggests about 20 hours, so you request 24 hours (to be safe). 24 hours later your job is killed, unfinished. So you submit again, this time requesting 48 hours (really make sure). The job finishes after 25 hours.

Requesting too little time is worse than requesting too much time, but there are times when you cannot know in advance how long a job needs. Checkpointing allows your job to pick up from where it left off. (also good for extra long jobs)

Scheduling first, then scripting, then monitoring, then checkpoint-restart

Working on the HPC clusters is not the same as working at a desktop workstation: in order to provide high performance computing to many users simultaneously, computational work must be packaged into a job - a script specifying what resources the job will need and the commands necessary to perform the work - and submitted to the system to be run without further input from the user. The system then schedules and runs the job on a dedicated portion of the cluster. (Note that there is a way TODO link to work interactively within this model, for work such as debugging which cannot be scripted).

Following is an overview of scheduling. You can also jump directly to how to write (TODO link), submit (with qsub) (TODO link) and monitor (with qstat, pbstop and other tools) (TODO link) jobs.

Job Scheduling

A CPU can only do one thing at a time. When you run two programs at a time on your single-CPU workstation, the operating system uses "timeslicing" to give the illusion of running both at once: it runs one of the programs for a moment, then pauses that program to run the other one for a moment, and so on. If you are reading and editing documents, as most workstations are used for, this works fine - most of the time the computer is waiting for you to hit the next key anyway.

Running a simulation, however, uses the CPU heavily. When timeslicing between two simulations on a single-CPU workstation, you can expect each simulation to take roughly twice as long as if it had dedicated use of the CPU. This is obviously counter to the goals of high performance.
computing, so HPC clusters use *batch job scheduling* to ensure that each job has dedicated access to the resources it needs.

The principle behind batch job scheduling is:

- Each simulation is prepared (by the user) as a *job*, (batch job) which is a script that sets up and runs the simulation without interactive input from the user.
- Each job needs a certain set of resources for a certain amount of time. The user knows these needs in advance and can specify them in the job script.
  - "Resources" includes a number of CPUs and an amount of memory. Some jobs might also need a specific type of CPU or a specific software license.
- The scheduler plans when and on which compute nodes to run each job.

The diagram below illustrates how jobs can be allocated to certain parts of the cluster at certain times. When a new job is submitted, the scheduler looks for a place and time to run the job, always aiming to start the job as soon as possible and to make the most efficient use of the full resources of the cluster:

On the NYU clusters, *Torque* and *Moab* manage the running and scheduling of jobs. As a user you will interact mostly with Torque, which accepts and runs job scripts and manages and monitors the cluster's compute resources. Moab does the heavy thinking: the planning of which job should be run where and when.

Jobs needing fewer resources are easier to schedule - in the diagram above, a job requiring just 1 CPU for 1 hour could be inserted into the gap on Node 1 CPU 4. Smaller jobs are also more likely to receive priority when being scheduled. Therefore **avoid requesting vastly more CPUs, memory or walltime than you actually need**. Note that a small overestimate, such as 10%-20%, is wise, lest your job run out of time and be killed before it finishes, but requesting several times what you need will result in longer queueing time for your job and less efficient system utilization for everybody.

**Login and Compute Nodes**

Whilst the bulk of an HPC cluster's workload is running simulations, there is also a range tasks requiring user interaction, such as:
• Writing and compiling code
• Debugging code (but more on this later) (TODO link to interactive jobs and debugging)
• Preparing, submitting and monitoring batch jobs
• Analyzing and visualizing simulation results
• Moving data to, from and within the cluster

Such tasks do not have the performance requirements that simulations do, but also must not be allowed to interfere with running jobs. To achieve this the cluster is divided into two primary components:

• a small number of “login” nodes, for interactive activities
• a large number of “compute” nodes, for running jobs

The diagram below illustrates this division. Note also that different filesystems may be visible to the login nodes as to the compute nodes, specifically at NYU /archive is not visible to the compute nodes.

![](image)

---

**Important**

Do not run computationally-heavy or long-running jobs on the login nodes! Not only will you have poor performance, the heavy resource usage of such jobs impacts others ability to use the login nodes for their intended purposes. If you need to run a job interactively (for example, when debugging), please do so through an interactive batch session (TODO link)

---

**Job Queues**

Not all jobs can be run at once - the cluster is finite! - so when jobs are submitted they are placed into a queue. When a “space” becomes available in the schedule Moab looks down the queue for the first job that will fit into the space.

Jobs are not necessarily placed at the end of the queue - Moab uses the priority (discussed [here](#)) to determine where in the queue a job should be placed.

At NYU HPC, part of the cluster is reserved for short (<12 hour) jobs - so short jobs are likely to start sooner

There is more than one queue. Each queue is configured for different types of jobs and has resource limits and priorities set accordingly. If you do not specify a queue to submit to, Torque will use the resources requested to select a queue for you. Frequently this is the best option, however in some circumstances you are better off explicitly specifying a queue.

You can see the list of queues with the command "qstat -q", and you can see more detail about a specific queue with "qstat -Qf queue-name".

The following example shows the queues available on Bowery, with some more detail about each queue in the table below. The output shows:

• The name of each queue
• The maximum memory, CPU time, Wallclock time and number of nodes that a job in each queue can use
• The number of currently queued and currently running jobs in each queue
• The queue job limits and state (these columns are of interest mostly to the system administrators)
$ qstat -q
server: crunch.local

<table>
<thead>
<tr>
<th>Queue</th>
<th>Memory</th>
<th>CPU Time</th>
<th>Walltime</th>
<th>Node</th>
<th>Run</th>
<th>Que</th>
<th>Lm</th>
<th>State</th>
</tr>
</thead>
<tbody>
<tr>
<td>s48</td>
<td>46gb</td>
<td></td>
<td>48:00:00</td>
<td>1 606 550</td>
<td>--</td>
<td>E R</td>
<td></td>
<td></td>
</tr>
<tr>
<td>bigmem</td>
<td>244gb</td>
<td></td>
<td>48:00:00</td>
<td>8 9 0</td>
<td>--</td>
<td>E R</td>
<td></td>
<td></td>
</tr>
<tr>
<td>p12</td>
<td></td>
<td></td>
<td>12:00:00</td>
<td>13 7</td>
<td>--</td>
<td>E R</td>
<td></td>
<td></td>
</tr>
<tr>
<td>cgsb-s</td>
<td></td>
<td></td>
<td>1000:00:</td>
<td>1 17 6</td>
<td>--</td>
<td>E R</td>
<td></td>
<td></td>
</tr>
<tr>
<td>route</td>
<td></td>
<td></td>
<td></td>
<td>0 0</td>
<td>--</td>
<td>E R</td>
<td></td>
<td></td>
</tr>
<tr>
<td>p48</td>
<td></td>
<td></td>
<td>48:00:00</td>
<td>2 0</td>
<td>--</td>
<td>E R</td>
<td></td>
<td></td>
</tr>
<tr>
<td>interactive</td>
<td></td>
<td></td>
<td>04:00:00</td>
<td>2 5</td>
<td>0</td>
<td>E R</td>
<td></td>
<td></td>
</tr>
<tr>
<td>sysadm</td>
<td></td>
<td></td>
<td></td>
<td>0 0</td>
<td>--</td>
<td>E R</td>
<td></td>
<td></td>
</tr>
<tr>
<td>cuda</td>
<td></td>
<td></td>
<td>48:00:00</td>
<td>4 0</td>
<td>0</td>
<td>E R</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

652 563

Unable to render (include) The included page could not be found.

Writing job scripts

Submitting a Job