Tutorial 2:
HPC at NYU

Accessing software with Environment Modules

Job scripts and how to reserve resources

Introduction to job scheduling
Submitting a job with qsub
Requesting resources

Requesting resources

Using compute nodes interactively
Key Points

- `qsub -l resource=amount` to specify what a job needs
- CPUS (nodes=, proc=), memory (mem=) and time (walltime=) are the most important resources.
- The job script is executed on only the first node

At a minimum, the scheduler needs to know:

- How many CPU cores you need, and whether they must be on the same node
  - If you don't know, the answer is probably "1 core". If the program supports parallel, it probably supports multithreading - multiple cores on a **single** node. To use multiple nodes, a program generally needs MPI, so you should only request multiple nodes if you...
are sure the program can use them.

- How much memory you need
  - NYU has nodes with 23GB, 46GB, 62GB, 90GB, 120GB and 189GB available to jobs. (The remaining memory is needed by the operating system)
- How long the job is expected to take
  - NYU HPC users can request up to 168 hours (1 week) for a single job. But priority is given to jobs requesting less time

**CPUs - nodes and cores**

HPC is associated with parallel processes - but it’s not magic! To use multiple CPUs, the program must have been written with either threading (eg OpenMP) or message passing (MPI).

A multithreaded job (OpenMP) can run on two CPUs in the same node, and share the same memory.

If in doubt:
- try with 1 core first
- check the documentation of the software, and next try with multiple cores on 1 node
- when using multiple nodes, check whether the job is actually running on all of the nodes. Contact us for help with this.

**How much do I need?**

The HPC cluster is not magic, its CPUs are only as fast as any other contemporary CPU. In fact, some nodes on Mercer are a few years old, and may be slower than your desktop (see Clusters July 2017 for a table of node types and when we installed them).

The performance of the HPC cluster comes from its *scale*. Most nodes in the cluster have 12 or 20 cores, 48GB up to 192GB of RAM, access to a large fast parallel filesystem and there is a 40Gb/s dedicated network link between any two nodes in each of the main groups. And there are thousands of nodes.

So although the resources your job needs depend very much on your job, and there is no simple rule for estimating requirements, you can make some initial guesses based on why you need the HPC cluster:

- **My desktop has not enough RAM**
  You should request at least as much RAM as your desktop possesses. The time required will probably be similar to the time required on your desktop. Be aware though that many problems scale with O(n^2) or more, so doubling the number of data points might require 4x the RAM and 8x the compute time.
- **Each run takes 4 hours on my 32GB desktop, and I have 1000 experiments to run**
Each experiment will probably take 32GB of memory and 4 hours on the HPC cluster too - but you can submit 1000 of them at once and a few hundred might run simultaneously.

For a few one-off jobs, you can safely request much more than you need. But use these qsub/PBS options:

```bash
#PBS -mae  # send an email if the job aborts (a) and when it ends (e)
#PBS -M $USER@nyu.edu  # send the email to this address
```

In the email you will see something like:

```
PBS Job Id: 3205348
Job Name: testme.q
Exec host: compute-1-10.local/7
Execution terminated
Exit_status=0
resources_used.cput=00:32:01
resources_used.mem=3052mb
resources_used.vmem=25644mb
resources_used.walltime=00:42:03
Error_Path: login-0-2.local:/home/ab123/testing/testme.q.e3205348
Output_Path: login-0-2.local:/home/ab123/testing/testme.q.o3205348
```

From which you can deduce that the job took 42 minutes of wallclock time and about 3GB of memory. So a sensible resource request for the next job is (see RESOURCES for more about the options):

```
qsub -l walltime=1:00:00 -l mem=4GB testme.q
```

### Requesting resources

Moab schedules jobs based on the resources they expect to need. The default at NYU is 1 CPU core and 2GB of memory for 1 hour. To specify what you job will actually need, use the `-l` directive or option:

**Options to request compute resources:**

- `-l walltime=walltime`
  - Maximum wallclock time the job will need. Default is 1 hour. Walltime is specified in seconds or as `hh:mm:ss`.
- `-l mem=memory`
  - Maximum memory per node the job will need. Default depends on queue, normally 2GB for serial jobs and the full node for parallel jobs. Memory should be specified with units, eg `500MB` or `8GB`.
- `-l nodes=ppn=num`
  - Number of nodes and number of processors per node required. Default is 1 node and 1 processor per node. The `:ppn=num` can be omitted, in which case (at NYU HPC) you will get full nodes. When using multiple nodes the job script will be executed on the first allocated node.
- `-q queue`
  - Submit to a specific queue. If not specified, Torque will choose a queue based on the resources requested.

A job submitted without requesting a specific queue or resources will go to the default serial queue (`s48 on Mercer`) with the default resource limits for that queue.

Requesting the resources you need, as accurately as possible, allows your job to be started at the earliest opportunity as well as helping the system to schedule work efficiently to everyone’s benefit.
Resources can be requested with multiple \texttt{-l} options, or as a comma-separated list of options. Both of the following examples are correct:

\begin{verbatim}
$ qsub \texttt{-l} \texttt{walltime=2:00} \texttt{-l} \texttt{mem=4GB} \texttt{myjob.q}

$ qsub \texttt{-l} \texttt{walltime=2:00,mem=4GB} \texttt{myjob.q}
\end{verbatim}

Most nodes on Mercer have 48GB or 64GB memory. Requesting a large portion of the memory on a node will cause Moab to reserve an entire node for your job even if you only request 1 CPU, since there will be insufficient remaining memory to run other jobs.

A small amount of memory on each node is needed by the operating system, so for example on a 64GB node, only about 62GB is available to jobs. A job requesting 64GB of memory will therefore be too big for a 64GB node, and Moab will schedule it on a 96GB or 192GB node instead. We have fewer nodes with so much memory, so the job is likely to spend longer waiting in the queue. \textbf{Tip: try requesting 62GB instead.}

The serial queues on NYU HPC clusters are limited to a single node, but allow multiple processors on that node to be used. Therefore, parallel jobs using only one node, such as OpenMP or multithreaded jobs can be submitted to a serial queue.

When using more than one node, the \texttt{job script is executed only on the first node}. To make use of the other nodes you must use MPI or \texttt{pbsdsh}.

\section*{Requesting GPUs}

To request GPU nodes:

\begin{itemize}
\item \texttt{-l nodes=1:ppn=1:gpus=1}
  1 node with 1 core and 1 GPU
\item \texttt{-l nodes=1:ppn=1:gpus=1:titan}
  1 node with 1 core and 1 GPU, specifically a Titan Black GPU
\item \texttt{-l nodes=1:ppn=1:gpus=1:k80}
  1 node with 1 core and 1 GPU, specifically an Nvidia K80 GPU
\item \texttt{-l nodes=1:ppn=4:gpus=4:titan}
  1 node with 4 Titan GPUs. Note that we request \texttt{ppn=4} too, it is always best to request at least as many CPU cores are GPUs
\end{itemize}

The available GPU node configurations are shown \texttt{here}.  

When you request GPUs, the system will set two environment variables - we strongly recommend you \textbf{do not change} these:

\begin{itemize}
\item \texttt{CUDA_VISIBLE_DEVICES} has a comma-separated list of the device IDs this job is allowed to use (eg "2,3"). The CUDA library within the application will use this to prevent multiple GPU jobs on the same node from interfering with each other
\item \texttt{CUDA_DEVICES} has a zero-based sequence of the "logical device IDs" for your job (eg "0 1"). So, if your application expects a list of GPU IDs starting at zero, and you have been allocated GPU numbers 2 and 3, then you can pass \texttt{CUDA_DEVICES} to your application and it will see 2 devices, named "0" and "1", which happen to correspond (via \texttt{CUDA_VISIBLE_DEVICES}) to the GPUs whose physical IDs are "2" and "3"
\end{itemize}

To your application, it will look like you have GPU 0,1,... (up to as many GPUs as you requested). So if for example, you request 2 GPUs, and are allocated GPU 2 and GPU 3, you will have:

\begin{verbatim}
# echo $CUDA_VISIBLE_DEVICES
2,3
# echo $CUDA_DEVICES
0,1
\end{verbatim}

Now if your application calls "cudaSetDevice(0)", you will use \textit{the GPU that appears as device 0, but is actually device 2}. And a call to "cudaSetDevice(3)", will return an error, because as far as the application can see, the node only has 2 GPUs, numbered 0 and 1.

\section*{Exercise}
Try submitting a simple job. Here's an example script:

```
#!/bin/bash
#PBS -l nodes=1:ppn=1
#PBS -l walltime=5:00
#PBS -l mem=500mb
mkdir -p $SCRATCH/my_run_dir
cd $SCRATCH/my_run_dir
echo "running a job on node $(hostname) in directory $(pwd)"
sleep 120
echo "finishing"
```