Requesting resources via Slurm

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
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
  - 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

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 60GB, 124GB and 252GB 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).
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 Prince 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. When jobs are done, run `sacct` to check actual usages (run `sacct -j <jobID> -o` would show much more information):

```
$ sacct -j 6891 -o JobID,JobName,ExitCode,NNodes,NCPUS,MaxRSS,Elapsed
JobID   JobName ExitCode NNodes NCPUS MaxRSS    Elapsed
------- ------- -------- -------- -------- ------- -------
6891    mpiexec-t+ 0:0  3   84       00:41:21
6891.batch batch 0:0 1 28 647076K   00:41:21
6891.extern extern 0:0 3   84       0 00:41:21
6891.0   orted 0:0 2 14 639040K   00:41:19
```

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.
From which you can deduce that the job took 42 minutes of wall clock time and about 647MB of memory. So a sensible resource request for the next job is (see [RESOURCES](#) for more about the options):

```
$ sbatch -t1:00:00 --mem=1000 ... testme.s
```

### Requesting resources

Slurm 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 your job will actually need, use these #SBATCH option:

**Options to request compute resources:**

- `--nodes, --ntasks-per-node, --cpus-per-task`
  Number of nodes, number of tasks per node, and number of CPU core per task required. Default is, one node, one task per node, one CPU core per task, so is 1 CPU core on 1 node

- `--mem, memory required per node in MegaBytes`
  --mem-per-cpu, memory required per CPU in MegaBytes

- `--time`
  Maximum run time of a job. Time formats include "minutes", "minutes:seconds", "hours:minutes:seconds", "days-hours", "days-hours:minutes:seconds".

A job submitted without requesting any specific resources will get the default resource limits for the default partition.

#### Requesting GPUs

To request GPU nodes:

- `--gres=gpu:1`
  1 node with 1 core and 1 GPU card
- `--gres=gpu:2 -c2`
  1 node with 2 cores and 2 GPU cards
- `--gres=gpu:k80:3 -c3`
  1 node with 3 cores and 3 GPU cards, specifically the type of Tesla K80 cards. Note that it is always best to request at least as many CPU cores as GPUs

The available GPU node configurations are shown [here](#).

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

- `CUDA_VISIBLE_DEVICES`
- `GPU_DEVICE_ORDINAL`

To your application, it will look like you have GPU 0.1... (up to as many GPUs as you requested). So if for example, there are two jobs from different users: the first one requesting 1 GPU card, the second 3 GPU cards, and they happen landing on the same node gpu-08:
The 1st job

$ env | grep DEV
CUDA_VISIBLE_DEVICES=0
GPU_DEVICE_ORDINAL=0

$ nvidia-smi -L
GPU 0: Tesla K80 (UUID: GPU-c448f648-2759-5ec0-0fde-0b72a857cd09)

The 2nd job

$ env | grep DEV
CUDA_VISIBLE_DEVICES=0,1,2
GPU_DEVICE_ORDINAL=0,1,2

$ nvidia-smi -L
GPU 0: Tesla K80 (UUID: GPU-6e023c21-2644-aec5-5aa9-78a0874bda48)
GPU 1: Tesla K80 (UUID: GPU-f2735f79-615d-bf15-00eb-54b976b944b8)
GPU 2: Tesla K80 (UUID: GPU-b9bf5b2d-d7d5-85aa-75a5-68d0b532a4e)

The above boxes shown that there is an overlapped value of '0' between the two jobs for the two environment variables. But the GPU cards assigned are actually different for the jobs, as we should trust UUIDs the nvidia-smi command display to us.

Exercise

Try submitting a simple job. Here's an example script:

```bash
#!/bin/bash
#SBATCH -N1
#SBATCH -n1
#SBATCH -t5:00
#SBATCH --mem=500
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"
```