Working on the openmind computing cluster

BCS computational tutorial series, February 16 2017
Objectives

1. Learn the basics of how to use Openmind, a computing cluster managed using SLURM (Simple Linux Utility for Resource Management).
   a. See some example workflows.
   b. Learn best practices for effective usage.
2. More generally, see some strategies for using remote computing resources

Along the way...

- Learn to use the terminal
- Introduction to shell scripting
- Learn some basics of Linux operating systems
What is Openmind?

- A set of managed computers (called nodes) accessible through a single portal (openmind.mit.edu), enabling users to perform computations which are infeasible or too time consuming for a personal workstation/laptop.
- Different types of nodes with different numbers of CPUs and GPUs:
  - ~20 cores (up to 24) or (40 - 48 hyperthreaded)
  - most have 256 GB RAM, some 512GB, and one with 1 TB
  - 0 or several GPUs
- Lots of disk space
  - 483 TB total, all directories available from any node through Lustre filesystem
  - 10 TB quota per group (/om), plus 5 GB per user (/home)
- Fiber link to MIT
- Possibilities for expansion - subject to space constraints!
- Yearly fee of $2500 per lab/group
Openmind architecture

Login Node (openmind7.mit.edu) - Interactive use, code development.

Compute Nodes – Run “Batch Jobs” on, both single and multi-processor.

URLs like node001.mit.edu, node001.mit.edu ... node032.mit.edu ...
Getting an account

Submit a new issue to request an account:

Title: Account request

Message box: Attn @s-b: "mit_username beta_testers mit_email" PI: pi_name

For example, Attn @s-b: "ereming beta_testers ereming@mit.edu" PI: Mehrdad Jazayeri

As part of getting an account, you will also be subscribed to the openmind-announce mailing list at MIT. However, to get appropriate notifications from GitHub@MIT you will need to click on your settings and choose your primary email address.
How openmind works: Slurm

Simple Linux Utility for Resource Management. From slurm.schedmd.com:

Slurm is an open-source workload manager designed for Linux clusters of all sizes. It provides three key functions:

1. It allocates exclusive and/or non-exclusive access to resources (computer nodes) to users for some duration of time so they can perform work.
2. It provides a framework for starting, executing, and monitoring work (typically a parallel job) on a set of allocated nodes.
3. It arbitrates contention for resources by managing a queue of pending work.
Connect to Openmind: necessary software

Terminal emulator:

- Windows - MobaXterm (Windows 10 Bash shell?)
  - http://mobaxterm.mobatek.net/
- Macintosh – Built-in Terminal application
- Linux – Terminal application

SSH Client – To log in to the head node at openmind.mit.edu

File Transfer – Transferring files between openmind and your local machine.

X Forwarding – Display graphics for those programs with a GUI interface (such as MATLAB) or that otherwise display images.
The Linux shell

(Terminal, OSX)  (MobaXterm, Windows)
Connect to Openmind

From your ssh/terminal application on your tutorial workstation or your laptop or on a machine at home:

$ ssh -X <username>@openmind.mit.edu

-X enables X11 forwarding.

You will be prompted for your MIT password (Also can setup ssh keys)
Queue a task with srun

The srun command creates a “job allocation” and runs the associated command.

Example:

$ srun hostname

Output:

srun: job 6431128 queued and waiting for resources
srun: job 6431128 has been allocated resources	node040

Not really how SLRUM intended to be used: have to wait for resources!
Batch jobs

Batch jobs are jobs which are executed in the background once resources are available.

Typically each job will be defined by two or more scripts:

- The submission (sbatch) script which asks for a resource allocation and launches the job.
- The job script which defines the actual task (i.e. Matlab, Python, Freesurfer, etc.)

```
$ sbatch my_job_script.sh
```

This command will return a string informing the user of the Job ID number.
Requesting resources

Resources are what SLURM manages. These are CPUs, Memory (RAM), and sometimes GPUs. Specify using options.

Form is ‘-<single letter flag><optional value>’ or ‘--<long form flag>=<optional value>’

Example:

$ sbatch -n1

or

“=“ only used in conjunction with long-form flags. No spaces!

$ sbatch --ntasks=1

Some options only have a long form flag.
Requesting resources

- `-c, --cpus-per-task=<ncpus>`: advise the Slurm controller that ensuing job steps will require `ncpus` number of processors per task. Without this option, the controller will just try to allocate one processor per task.
- `--mem=<MB>`: specify the real memory required per node in MegaBytes. Use this or `--mem-per-cpu`.
- `--mem-per-cpu=<MB>`: minimum memory required per allocated CPU in MegaBytes. Use this or `--mem`.
- `-n, --ntasks=<number>`: request resources for a certain number of tasks
  - Note: the
- `-N, --nodes=<minnodes[-maxnodes]>`: request a minimum and maximum number of nodes to be allocated to the job.
- `-t, --time=<time>`: set a limit on the total run time of the job allocation. E.g. "hours:minutes:seconds" or "days-hours:minutes:seconds"

Quick example

Create script:

$vim testScript.sh

vim text editor basics: http://www.oregonwebradio.net/back up_fedora/tutorials/vim_li/quickstart.html

Script:

```bash
#!/bin/bash

#SBATCH --ntasks=1
#SBATCH --time=10:00
#SBATCH --mem-per-cpu=100
hostname
hostname

requests one CPU and 100 MB of RAM, in the default queue for 10 minutes. When started, the job would run a first command hostname, then when this is completed will run the second command, which is also hostname.
```

Now submit:

$sbatch testScript.sh

Print output file:

$ cat slurm-<jobid>.out

Output file

node048
node048
High throughput computing

The whole point! Several options, including:

- Parallel applications (e.g. parfor loops in Matlab).
- Use shell scripts (loops) to submit multiple job scripts.
- Use single scripts to perform multiple parallel tasks.
- Job arrays.

Each has advantages and disadvantages.
Parallel tasks with srun in batch scripts

“srun is used to submit a job for execution or initiate job steps in real time. srun has a wide variety of options to specify resource requirements, including: minimum and maximum node count, processor count, specific nodes to use or not use, and specific node characteristics (so much memory, disk space, certain required features, etc.). A job can contain multiple job steps executing sequentially or in parallel on independent or shared nodes within the job's node allocation.”

If srun is used to launch non-MPI tasks in a script where either -n or -N are larger than one, then the script will be launched -n or -N times. -n and -N values must be compatible; SLURM will adjust -N.

Using srun is not necessary for scripts which have only one task (i.e. run this Matlab script).
Parallel example

Create script:

```
$ vim testScriptParallel.sh
```

Script:

```
#!/bin/bash

#SBATCH --nodes=2
#SBATCH --ntasks=2
#SBATCH --time=10:00
#SBATCH --mem-per-cpu=100

srun hostname
srun hostname
```

requests one CPU and 100 MB of RAM for two steps, in the default queue for 10 minutes. When started, the job would run a first job step `hostname` on each of two nodes, then when this is completed will run the second job step again on the same two nodes. We could have set `--nodes=1` if we wanted both tasks on the same node, or not set it at all if we didn’t care.

Now submit:

```
$ sbatch testScript.sh
```

Load output file:

```
$ cat slurm-<jobid>.out
```

Output file:

```
node002
node003
node002
node003
```
Job arrays

Create script:

```
$ vim testScriptArray.sh
```

Script:

```
#!/bin/bash
#
#SBATCH --nodes=1
#SBATCH --ntasks=1
#SBATCH --time=10:00
#SBATCH --mem-per-cpu=100
hostname
echo ${SLURM_ARRAY_TASK_ID}
```

Here the job script includes an additional environmental variable SLURM_ARRAY_TASK_ID which can be passed to your application.

Now submit:

```
$ sbatch --array=1-2 testScriptArray.sh
```

Load output file:

```
$ vim slurm-<jobid>.out
```

Output files:

- node040 1
- node044 2
Interactive Jobs

Interactive jobs are useful for debugging. To start an interactive job with a command line terminal:

```
$ srun -p om_interactive --pty bash
```

"om_interactive" is an argument to the "-p" (partition) option.

To use a GUI on openmind, use ssh X window tunneling.

```
$ srun --x11 --gres=gpu:1 --mem=8000 --pty bash
$ module add mit/matlab/2016b
$ matlab
```

"add" is a sub-command of "module." Try typing:

```
$ module --help.
```
Other useful sbatch options

- **-a, --array=<indexes>:** submit a job array, multiple jobs to be executed with identical parameters. An environment variable will be created called SLURM_ARRAY_TASK_ID. This can be passed to your function in the script call command. This will request the amount of resources you’ve asked for multiplied by the number of elements in the index.

- **-d, --dependency=<dependency_list>:** defer the start of this job until the specified dependencies have been satisfied completed.
  - Unfortunately, ‘dependency_list’ must be a jobid, which are only generated at time of job submission, making scripting trickier.

- **--mail-type=<type>:** Notify user by email when certain event types occur. Valid type values are NONE, BEGIN, END, FAIL, REQUEUE, ALL. Use with --mail-user.

- **--mail-user=<user/email>:** User to receive email notification of state changes as defined by --mail-type.

- **-p, --partition=<partition_names>:** request a specific partition for the resource allocation.

- **--qos=<qos>:** request a quality of service for the job.

- **-w, --nodelist=<node name list>:** Request a specific list of nodes.

- **-x, --exclude=<node name list>:** explicitly exclude certain nodes.
Simple best practices!

● Don't run anything on the login node.

● Be as accurate as possible when specifying resources (time, memory, cpus) for jobs.

● Make jobs granular to aid in scheduling.

● Keep job counts in a reasonable range. Use parallel jobs and job arrays.
Cluster upgrades

- Located in Holyoke, MA at the Massachusetts Green High Performance Computing Center (MGHPCC), a collaboration between 5 major universities and the Commonwealth of Massachusetts.
- Fair bit of room for expansion, but please ask early
- Contribute nodes or GPUs
  - provides additional priority for your group
  - maintenance done by openmind
  - contact before purchase to determine suitability