Purdue University - ITaP Research Computing

Gladys Andino  
Dan Dietz  
Jieyu Gao  
Lev Gorenstein  
Erik Gough  
Stephen Harrell  
Randy Herban  
Steve Kelley  
Boyu Zhang  
Xiao Zhu  
Eric Adams

rcac-help@purdue.edu

April 2, 2021

Slides available:
www.rcac.purdue.edu/training/clusters101/
Vocabulary

Connecting

■ Windows
■ Mac
■ Linux

Modules

Types of Work
■ Serial vs Parallel
■ Parallel Computations

Good Citizenship

Jobs, Queues, and SLURM
■ Jobs
■ Queues
■ SLURM
■ Life of a Job
A few additional acknowledgments to the many people who have helped make this workshop possible.

- The material in this workshop was prepared by the Purdue University ITaP Research Computing team.
Vocabulary

Connecting
Modules
Types of Work
Good Citizenship
Jobs, Queues, and SLURM
A cluster is a cluster is a cluster?

- Hardware (compute nodes + interconnect + storage)
- Software (OS + compilers + libraries + apps + queue manager)
- Infrastructure (front-ends + power + cooling + data center + staff)
Vocabulary

Node:

- Network card
- Memory
- Processors
- Memory
- HDD
- Power supply
Vocabulary

- CPU:
  - Core
  - Cache
  - Controllers, I/O, etc.
Processor Core: individual compute unit ("slot") on the chip

- You see:
  2 physical processors, 10 cores ea.

- Queuing system sees:
  20 logical processors

- From now on, we will be mostly concerned with cores (logical processors), not physical chips
  ("I ran on 5 CPUs" == "on 5 processors" == "on 5 cores")
Front-end vs compute node:

“Front-end”
- shared by many
- e.g. scholar-fe01

“Back end”
- e.g. scholar-a003

Internet

ssh

batch job

queue

File system ( $HOME, $RCAC_SCRATCH)

Running Jobs: The goal is getting to the compute nodes
Vocabulary

IBM SYSTEM 360 (1 Computer, 1 Processor):
Community clusters:

**HPC (Halstead, Brown, Bell):** Multiple cores or nodes, probably MPI. Benefit from high-performance network and parallel filesystem. The vast majority of campus - 80% of all work!

**GPU Accelerated (Gilbreth):** Utilizes Nvidia P100 and V100 GPUs for acceleration. Useful for Machine Learning, AI, Computational Chemistry, etc.

**Scholar:** Special case for teaching. Mostly MPI at first glance, but also highly tweaked for interactive use (tasks on front-ends, Jupyter notebooks, Rstudio, etc). Also couple GPUs and mini-Hadoop.
Need help with ITaP Research Computing resources?

Check our User Guides and documentation:
www.rcac.purdue.edu/knowledge

You can always send us an email:
rcac-help@purdue.edu

Come to our coffee hour consultations:
www.rcac.purdue.edu/coffee
Connecting

- Windows
- Mac
- Linux
Many clients are available for Windows:

- We will use the PuTTY SSH client
- Download PuTTY, no install required
- [http://www.chiark.greenend.org.uk/~sgtatham/putty/download.html](http://www.chiark.greenend.org.uk/~sgtatham/putty/download.html) (or Google search *putty*)
- Download putty.exe for Intel x86 to your desktop
Host Name for Scholar is scholar.rcac.purdue.edu
Connecting

Mac

Connect using:

```
ssh myusername@scholar.rcac.purdue.edu
```
Connecting

Linux

Linux also has a built in terminal client, similar to Mac:

```
ssh myusername@scholar.rcac.purdue.edu
```
Vocabulary
Connecting

Modules

Types of Work

Good Citizenship

Jobs, Queues, and SLURM
All ITaP Research Computing clusters use a module system

- Module system provides for the dynamic modification of a user’s environment
- Module commands allow you to add applications and libraries to your environment
- This allows us to simultaneously and safely provide several versions of the same software
- You can set up the environment for running applications using simple commands:

  ```
  $ module load gcc  
  $ module load openmpi
  ```

- All ITaP Research Computing clusters have a default programming environment (compiler and MPI library) loaded for you when you log in.
Show all modules currently loaded in my environment:

```
$ module list

Currently Loaded Modules:
  1) intel/17.0.1.132  2) impi/2017.1.132  3) rcac
```

Load a new module:

```
$ module load matlab
$ module list

Currently Loaded Modules:
  1) intel/17.0.1.132  2) impi/2017.1.132  3) rcac
  4) matlab/R2017a
```
Shows you all software modules currently available to you:

```
$ module avail
```

Filters software modules currently available to you by a search term. In this case, show all Anaconda Python versions available:

```
$ module avail anaconda
```

```
--------------------- Core Applications ---------------------
 anaconda/5.0.0-py27  anaconda/5.1.0-py27 (D)
 anaconda/5.0.0-py36  anaconda/5.1.0-py36
```
All ITaP Research Computing clusters use an hierarchical software configuration. Modules are organized by the programming environment (compiler and libraries) that they were built with and depend on. You will only see (with `avail`) modules compatible with the programming environment currently loaded.

As you swap in other programming environments (compiler and libraries), you may find additional software available to you. Any libraries or software dependent on the programming environment will be automatically swapped with compatible versions as you change environments.
We can look for software not immediately available to us - that is, things that may not show up in `avail` output:

```
$ module avail openmpi
$ module spider openmpi

openmpi:

  Versions:
    openmpi/2.1.0
```

For detailed information about a specific "openmpi" module (including how to load the modules) use the module’s full name. For example:

```
$ module spider openmpi/2.1.0
```

This tells that openmpi is installed on the cluster, but we must further investigate how to load it.
Running `spider` command for specific version of openmpi:

```
$ module spider openmpi/2.1.0

--------------------------------------------------------------
openmpi: openmpi/2.1.0
--------------------------------------------------------------
You will need to load all module(s) on any one of the lines below before the "openmpi/2.1.0" module is available to load.
gcc/4.8.5
gcc/6.3.0
```

This tells us we must swap into a new programming environment to load openmpi:

```
$ module load gcc/4.8.5 openmpi/2.1.0
```
Unload a currently loaded module:

```
$ module unload matlab
$ module list
```

Currently Loaded Modules:
1) gcc/4.8.5  2) openmpi/2.1.0  3) rcac

Unload and purge all currently loaded modules:

```
$ module purge
$ module list
No modules loaded
```
What exactly does module load do?

It will change the environmental variables defined for that module. Typically these are $PATH$ and $LD_LIBRARY_PATH$. Also a few extra module-specific environmental variables could be set.

```
$ module show vmd
-----------------------------------------------
/opt/modulefiles/core/vmd/1.9.3.lua:
-----------------------------------------------
whatis("invoke VMD 1.9.3")
setenv("VMD_HOME", "/apps/cent7/vmd/vmd-1.9.3")
prefix_path("PATH", "/apps/cent7/vmd/vmd-1.9.3/bin")
prefix_path("LD_LIBRARY_PATH", "/apps/cent7/vmd/vmd-1.9.3/lib")
setenv("LIBGL_ALWAYS_INDIRECT", "y")
```
A few tips from us:

- Don’t load modules in `.bashrc/.profile/.login/.cshrc`
- Don’t load more than needed for current task
- Do `module purge` when in doubt
- Do check with `module show` what a given module does
- Do check with `module list` what’s actually loaded
- Do script and automate your builds - even if it’s just a 3-line snippet! Reproducibility is good!
Modules

Exercises:

1. List currently loaded modules
2. List available modules
3. Search available modules for matlab
4. Show what one of the matlab modules does
5. Load the openmpi module (hint: spider)
6. Purge all modules
7. List currently loaded modules
Exercises:

1. List currently loaded modules

   $ module list

   Currently Loaded Modules:
   1) intel/17.0.1.132  2) impi/2017.1.132  3) xalt (S)  4) rcac

2. List available modules

   $ module avail

   ...

3. Search available modules for matlab

   $ module avail matlab

   ---------------- Core Applications ----------------
   matlab/R2017a (D)  matlab/R2018a
4. Show what one of the matlab modules does

```bash
$ module show matlab
```

5. Load the openmpi module (hint: spider)

```bash
$ module spider openmpi
$ module spider openmpi/2.1.0
$ module load gcc/4.8.5 openmpi/2.1.0
Lmod is automatically replacing "intel/17.0.1.132" with "gcc /4.8.5".
Lmod is automatically replacing "impi/2017.1.132" with "openmpi /2.1.0".
```

6. Purge all modules

```bash
$ module purge
```

7. List currently loaded modules

```bash
$ module list
No modules loaded
```
Types of Work

- Serial vs Parallel
- Parallel Computations
Types of Work
Serial vs Parallel

More than one mower!
Mow the lawn faster ...
... or mow a bigger lawn
Lawn mowers = cores
Types of Work
Serial vs Parallel

Vocabulary
Connecting Modules
Types of Work
Serial vs Parallel
Parallel Computations
Good Citizenship
Jobs, Queues, and SLURM

More than one mower!
Mow the lawn faster ...
... or mow a bigger lawn
Lawn mowers = cores
Parallel computations come in a few flavors:

- **MPI** - Message Passing Interface, multiple nodes
- **OpenMP** - confine yourself to one node
- **Hybrid** - mix MPI and OpenMP
- **Accelerators/coprocessors** - GPU, Intel Xeon Phi
- **Parallel R, Matlab, Hadoop and Spark**
Good Citizenship

Vocabulary
Connecting
Modules
Types of Work

Good Citizenship

Jobs, Queues, and SLURM
Good Citizenship

Front-end vs compute node:

“Front-end” shared by many

“Back end”

Running Jobs: The goal is getting to the compute nodes

File system ( $HOME, $RCAC_SCRATCH)

Most common

Internet

ssh

Front-end

e.g. scholar-fe01

batch job

queue

Compute Nodes

e.g. scholar-a003

...
Good Citizenship

What does this mean?

- Do not do major science on the front end
- Front-ends are very limited resources and you’ll annoy the system administrators and other users
- Watchdog will monitor and quickly terminate (or severely throttle) any inappropriate resource-intensive process
- Instead: submit a job. We’ll get to that
What can I do on the front-ends?

- Building/compiling software
- Managing files: editing, transferring, tar, gzip, hsi
- Submitting, monitoring, and managing batch jobs
- Launching interactive jobs
- Modest post-processing and analysis

Scholar is somewhat fortified for interactive use, and you could run some things on its front-ends... but for anything heavy you’d still be much better off submitting a job.
Jobs, Queues, and SLURM

- Jobs
- Queues
- SLURM
- Life of a Job
What is a job?

- A job is simply a set of tasks to be performed by a cluster
- A script to instruct the cluster precisely what to do to complete your work
- Self contained to be executed without any interaction
- Submit and forget!
Remember, cluster front-end nodes are shared resources for

- Creating, submitting, and monitoring jobs
- File transfers
- Preparing inputs
- Editing and compiling code
- Small-scale testing

May be used by 50+ people simultaneously

- Check out the who command
- Jobs should have no interference by other people
- Want jobs carefully arranged on compute nodes
Jobs, Queues, and SLURM

Jobs

Jobs are submitted to the cluster:

• Cluster executes jobs on back-end compute nodes
• Jobs are carefully scheduled and arranged on the compute nodes
Jobs need to specify the resources they require:

- Three basic units:
  - Number of nodes
  - Number of cores
  - Time

- Memory

- Other resources

Cluster will allocate requested resources once they are available. Job starts once resources are allocated.
Jobs, Queues, and SLURM

Jobs

The most basic unit you can request is nodes:

- Serial applications probably only need one node
- Parallel applications may need 10s or 100s of nodes

Job should also request how many cores per node it needs:

- Cluster allows your job to use only the cores you request
- Often your job should request all of the cores in a node
- Remember, we want to avoid other people!
Nodes are physically different pieces of hardware. A job with 20 nodes and 20 cores each, is **not** 400 threads

- 20 individual computers!
- Program needs to be written with MPI to tie all the nodes together
Nature of the job: how much inter-process communications?
1 core on 20 nodes, or 20 cores on 1 node?

- Communication inside a node is faster than between nodes
- Collisions with other people’s jobs (20 nodes \times 19 other jobs = 380 jobs)
- Most performant, most predictable, least vulnerable, least interfering

Tight packing preferred. Most ITaP clusters enforce exclusive nodes unless explicitly specified for single node jobs.
Jobs, Queues, and SLURM

Jobs

The third basic unit jobs can request is walltime. The cluster won’t let your job run forever!

Cluster needs to have a time limit for your job:

- Helps the cluster efficiently schedule you on the nodes
- Make request accurate with some safety buffer
- Tune your walltime request through testing and experimenting
Jobs, Queues, and SLURM

Jobs

Provide a memory estimate for your jobs! Slurm controls memory just like nodes and cores.

Slurm will assign a default value if you do not.

This default will be proportional to the number of cores you requested on the node. If you exceed the memory requested per node, even the default, your job will be killed!
This job asked for the default one core on one node and the default memory slice. On a 20-core node with 64GB per node, if this job exceeds 3GB, it will be killed:

$ sbatch myjob.sub

This job will be able to use up to 60GB of memory per node:

$ sbatch --mem=60G myjob.sub
Memory across multiple compute nodes is distributed. Memory is discrete chunks of memory in different address spaces.

Can’t just “add more nodes” to get more memory:

- 20 nodes with 16 GB memory each is **not** 320 GB of memory
- 20 discrete chunks of 16 GB memory
- Can’t allocate 320 GB of memory to a single process
- Need MPI to tie memory together
Jobs, Queues, and SLURM

Jobs

SLURM is pretty good about isolating jobs from each other (even when they share a node). But if in doubt (or notice effects of contention), simply request an entire node’s worth of cores.
Jobs, Queues, and SLURM

Jobs

There are a number of other resources you can request in a job:

- Accelerators (GPUs)
- Specific types of nodes
- Certain licenses: MATLAB and its Toolboxes
Cluster can’t run all jobs at once so sometimes you must wait

- Jobs are submitted into queues
- Jobs wait until cluster and queue has free resources
- Queues set constraints on jobs that can be submitted
  - Max nodes
  - Max walltime
- Sets initial priorities
- Our clusters have several queue types
Jobs, Queues, and SLURM

Queues

Owner queues:

- Named for your PI, lab, or group
- Number of nodes set by what your PI buys into
- Allows for longest jobs - 336 hours
- Jobs should start in 4 hours or less (assuming your queue isn’t full of other jobs in the queue)

Queues are **not** tied to specific nodes in the cluster. They allow you to use the number of nodes somewhere in the cluster - whichever nodes are free, functioning, and of the type the queue can access.
Standby queue:

- Uses idle nodes from owner queues
- Everyone on cluster gets access
- Limited to 4 hours (owners must be able get quick access from their queue)
- No limit on number of nodes (probably won’t get whole cluster though)
- Lowest priority jobs, no promises on turnaround time (can be minutes or days)
- If you can run under 4 hours, go for it!
Debug queue:

- Allows for running small jobs for testing and debugging
- Up to 2 nodes for 30 minutes at a time
- Highest priority
- Starting time <1 minute
What is SLURM?

- The Simple Linux Utility for Resource Management (SLURM) is a system providing job scheduling and job management on compute clusters. With SLURM, a user requests resources and submits a job to a queue. The system will then take jobs from queues, allocate the necessary nodes, and execute them.
What is SLURM, really? It is the resource manager for the cluster:

- Job submission
- Job status
- Schedules jobs
- Executes jobs
- Manages queues
- Manages nodes
You prepare a job submission script
You submit job script into SLURM. The job is placed into a queue.
Jobs, Queues, and SLURM

Life of a Job

Scheduler iteratively asks for new jobs, status of old jobs, and status of nodes. Each cycle can last several minutes.
Jobs, Queues, and SLURM

Life of a Job

Scheduler considers jobs, acts as a handler. Tells SLURM to start jobs one at a time. Goes back and does this over and over.
SLURM checks health of the node. Send job script to node(s) and executes it.
Jobs, Queues, and SLURM

Life of a Job

SLURM watches for the job script to complete on nodes. Collects job output files from nodes and sends files to you.