What we're covering:

- Review Cluster Architecture
- Working with SLURM
- Simple job submission script
- How to submit job script
- `sinfo, salloc, squeue, scancel, sbatch, srun`
- Tips for scripting submit files
- Reporting Job Errors
Cluster Architecture

- Login Node - Server that acts as your interface to the cluster
- Scheduler - Server that schedules jobs
- Compute Nodes - Servers that run jobs
Cluster Architecture (Storage)

Login Nodes

Scheduler

SLURM

Compute Nodes

/home and /group

/lustre
Cluster Architecture (Storage)

Login Nodes

Scheduler

SLURM

Compute Nodes

/home and /group

Running Job Data goes here!

/lustre
Overview: Slurm is an open source, fault-tolerant, and highly scalable cluster management and job scheduling system for large and small Linux clusters.

- Slurm requires no kernel modifications for its operation and is relatively self-contained. As a cluster workload manager, Slurm has three key functions:
  - First, it allocates exclusive access to resources (compute nodes) for users for some duration of time so they can perform work.
  - Second, it provides a framework for starting, executing, and monitoring work (normally a parallel job) on the set of allocated nodes.
  - Finally, it arbitrates contention for resources by managing a queue of pending work.
Why use SLURM?

- SLURM allows jobs to be scheduled so a user does not have to wait until a node is free to begin work. SLURM does the waiting for you!
- SLURM allows resources to be prioritized for groups that purchase shares on Colonial One.
- SLURM enables efficient use of the cluster since it constantly monitors resources in use and schedules jobs on unallocated resources as they free up.
- SLURM runs a job based on a submit script.
- A submit script calls your job script to execute your calculations.
As depicted in Figure 1, Slurm consists of a **slurmd** daemon running on each compute node and a central **slurmctld** daemon running on a management node (with optional fail-over twin). The **slurmd** daemons provide fault-tolerant hierarchical communications. The user commands include: **sacct**, **salloc**, **sattach**, **sbatch**, **sbcast**, **scancel**, **scontrol**, **sinfo**, **smap**, **squeue**, and **srun**. All of the commands can run anywhere in the cluster.
The entities managed by these Slurm daemons, shown in Figure 2, include **nodes**, the compute resource in Slurm, **partitions**, which group nodes into logical (possibly overlapping) sets, **jobs**, or allocations of resources assigned to a user for a specified amount of time, and **job steps**, which are sets of (possibly parallel) tasks within a job. The partitions can be considered job queues, each of which has an assortment of constraints such as job size limit, job time limit, users permitted to use it, etc. Priority-ordered jobs are allocated nodes within a partition until the resources (nodes, processors, memory, etc.) within that partition are exhausted. Once a job is assigned a set of nodes, the user is able to initiate parallel work in the form of job steps in any configuration within the allocation. For instance, a single job step may be started that utilizes all nodes allocated to the job, or several job steps may independently use a portion of the allocation.
Common software is available to all users via the “module” system

Can I install my own software?

- Yes, provided it runs from your home/group directory
- Users cannot use a package manager (yum) to install software
- Colonial One users do not have root on Colonial One

Login to Colonial One using your GW NetID/email address & password

$ ssh <netid>@login.colonialone.gwu.edu
### SLURM Commands

**sinfo**
- lists node and partition information for the cluster
- useful for finding unallocated nodes

Example: type "sinfo" at the prompt

<table>
<thead>
<tr>
<th>PARTITION</th>
<th>AVAIL</th>
<th>TIMELIMIT</th>
<th>NODES</th>
<th>STATE</th>
<th>NODELIST</th>
</tr>
</thead>
<tbody>
<tr>
<td>defq*</td>
<td>up</td>
<td>14-00:00:0</td>
<td>128</td>
<td>alloc</td>
<td>node[033-160]</td>
</tr>
<tr>
<td>short</td>
<td>up</td>
<td>2-00:00:00</td>
<td>95</td>
<td>alloc</td>
<td>node[097-191]</td>
</tr>
<tr>
<td>128gb</td>
<td>up</td>
<td>14-00:00:0</td>
<td>24</td>
<td>alloc</td>
<td>node[041-191]</td>
</tr>
<tr>
<td>256gb</td>
<td>up</td>
<td>14-00:00:0</td>
<td>8</td>
<td>alloc</td>
<td>node[033-040]</td>
</tr>
<tr>
<td>2tb</td>
<td>up</td>
<td>14-00:00:0</td>
<td>1</td>
<td>alloc</td>
<td>node901</td>
</tr>
<tr>
<td>gpu</td>
<td>up</td>
<td>7-00:00:00</td>
<td>22</td>
<td>alloc</td>
<td>node[003-029,032]</td>
</tr>
<tr>
<td>gpu</td>
<td>up</td>
<td>7-00:00:00</td>
<td>10</td>
<td>idle</td>
<td>node[001-002,021-028]   ← Free nodes!</td>
</tr>
<tr>
<td>gpu-noecc</td>
<td>up</td>
<td>7-00:00:00</td>
<td>22</td>
<td>alloc</td>
<td>node[003-029,032]</td>
</tr>
<tr>
<td>ivygpu</td>
<td>up</td>
<td>7-00:00:00</td>
<td>21</td>
<td>idle</td>
<td>node[333-353]           ← Free nodes!</td>
</tr>
<tr>
<td>ivygpu-noecc</td>
<td>up</td>
<td>7-00:00:00</td>
<td>21</td>
<td>idle</td>
<td>node[333-353]           ← Free nodes!</td>
</tr>
<tr>
<td>allgpu-noecc</td>
<td>up</td>
<td>7-00:00:00</td>
<td>22</td>
<td>alloc</td>
<td>node[003-020,029-032]</td>
</tr>
<tr>
<td>allgpu-noecc</td>
<td>up</td>
<td>7-00:00:00</td>
<td>31</td>
<td>idle</td>
<td>node[001-002,021-028,333-353] ← Free nodes!</td>
</tr>
<tr>
<td>debug</td>
<td>up</td>
<td>4:00:00</td>
<td>2</td>
<td>alloc</td>
<td>node[991-992]</td>
</tr>
<tr>
<td>debug-cpu</td>
<td>up</td>
<td>4:00:00</td>
<td>1</td>
<td>alloc</td>
<td>node992</td>
</tr>
<tr>
<td>debug-gpu</td>
<td>up</td>
<td>4:00:00</td>
<td>1</td>
<td>alloc</td>
<td>node991</td>
</tr>
</tbody>
</table>
**SLURM Commands**

- **salloc** - Obtain a Slurm job allocation (a set of nodes), execute a command, and then release the allocation when the command is finished. You can use `salloc` to run interactive jobs:

  ```
salloc -N 1 -p ivygpu -t 5
  ```

- **srun** - Run a parallel job on cluster managed by Slurm. Use `srun` to identify your allocated nodes after running `salloc`:

  ```
srun hostname # or
squeue -u <netid>
ssh nodename
  ```
**SLURM Commands**

**squeue** - view information about jobs located in the Slurm scheduling queue.

List jobs for your user account:

`squeue -u <username>`

Estimate when a job will start:

`squeue -u username --start`

List by job status:

`squeue -u username -t RUNNING or PENDING`
Check Status of Job by User

[ user@login4 ~ ]$ squeue -u <username>

<table>
<thead>
<tr>
<th>JOBID</th>
<th>PARTITION</th>
<th>NAME</th>
<th>USER</th>
<th>ST</th>
<th>TIME</th>
<th>NODES</th>
<th>NODELIST(REASON)</th>
</tr>
</thead>
<tbody>
<tr>
<td>2325403</td>
<td>defq</td>
<td>job_OpenMP.sh</td>
<td>user</td>
<td>PD</td>
<td>0:00</td>
<td>1</td>
<td>(Resources)</td>
</tr>
</tbody>
</table>

Jobs typically pass through several states in the course of their execution. The typical states are PENDING, RUNNING, SUSPENDED, COMPLETING, and COMPLETED. An explanation of each state follows.

CA = CANCELLED
CD = COMPLETED
CG = COMPLETING
F  = FAILED
NF = NODE_FAIL
PD = PENDING
R  = RUNNING
S  = SUSPENDED
TO = TIMEOUT
SLURM Commands

**sbatch** - Submit a batch script to Slurm

```
sbatch <submit_script.sh>
```

**scancel** - used to signal or cancel jobs, job arrays or job steps.

```
scancel <jobid>
```

**sinfo** - view information about Slurm nodes and partitions.
● **salloc** - Obtain a Slurm job allocation (a set of nodes), execute a command, and then release the allocation when the command is finished.

● **squeue** - View information about jobs located in the Slurm scheduling queue.

● **scancel** - Used to signal jobs or job steps that are under the control of Slurm.

● **sbatch** - Submit a batch script to Slurm.

● **srun** - Run parallel jobs
This small Python program will list an mpi process number and the node it is running on. Copy the text into a document called "hello-mpi.py" and save it.

```python
#hello.py
from mpi4py import MPI
import socket

comm = MPI.COMM_WORLD
rank = comm.Get_rank()
size = MPI.COMM_WORLD.Get_size()
thishost = socket.gethostname()

print "hello world from process ", rank, " of ", size, " on ", thishost
```

<------ THIS IS ALL ONE LINE
How to submit job script

Load the current MPI module so we can run our program correctly:

    module load openmpi/1.8/gcc/4.9.2/cpu
    module load python/2.7.6

Check your home directory to make sure your compiled program is there:

    ls -la hello*
How to submit job script

Create a SLURM script using an editor such as vi or emacs using steps 1 through 3. The script (or file) can be called anything you want but should end in .sh (i.e. submit.sh).

**Step 1: Resource Specification**

```
[login4 ~]$ nano submit.sh

#!/bin/sh
#SBATCH --time 5:00
#SBATCH -o testing%j.out
#SBATCH -e testing%j.err
#SBATCH -p defq -N 1
#SBATCH --mail-user=<username>@gwu.edu
#SBATCH --mail-type=ALL
module load openmpi/1.8/gcc/4.9.2/cpu
module load python/2.7.6
mpirun -n 8 python /home/<username>/hello.py
```

**Step 2: Submit job**

```
module load slurm
[login4 ~]$ sbatch submit.sh
Submitted batch job 2325403
```
How to submit job script

SLURM will email you when you job has finished. Once the job is finished, SLURM will place a .out file and a .err file (if there are errors) in the directory with your submit script.

```bash
$> ls slurm*
slurm-2333627.out  slurm-2333631.out  slurm-2333633.out
slurm-2333629.out  slurm-2333632.out  slurm-2333636.out
```

The .out file is the result of your job. Cat or less the file to show you the results.

```bash
$> cat slurm-2333636.out
Process 1 on node991.cm.cluster out of 8
Process 3 on node991.cm.cluster out of 8
Process 7 on node991.cm.cluster out of 8
Process 5 on node991.cm.cluster out of 8
Process 4 on node991.cm.cluster out of 8
Process 6 on node991.cm.cluster out of 8
Process 0 on node991.cm.cluster out of 8
Process 2 on node991.cm.cluster out of 8
```
Reporting Job Errors

- Email to hpchelp@gwu.edu
- Include .err file
- Include submit script
- Include what modules you loaded