Using the Cluster – Introduction

At RIT's Research Computing, we use a piece of software called SLURM to manage the many users we have contending for access to our limited physical resources. SLURM has a number of commands you may be unfamiliar with; the purpose of this document is to introduce to their basic usage.

Prerequisites:

- Getting a Research Computing Account – http://apply.rc.rit.edu
- Connecting to Research Computing Systems with SSH
- File Management - Getting your files to and from RC systems
- Using the bash shell and running software

Note: You might also want to check out these screencasts of the workshops we ran introducing how to use tropos: https://wiki.rit.edu/display/rc/Computing

Checking the status of the cluster

Once you've logged into the cluster headnode, ion.rc.rit.edu, you can check the status of the cluster by issuing the squeue command:

```
[abc1234@ion ~ []]$ squeue
JOBID PARTITION     NAME     USER  ST       TIME  NODES NODELIST(REASON)
65483      work test-3-5   abc1234  PD       0:00      1 (Priority)
65484      work test-4-1   abc1234  PD       0:00      1 (Priority)
65485      work test-4-2   abc1234  PD       0:00      1 (Priority)
65486      work test-4-3   abc1234  PD       0:00      1 (Priority)
65487      work test-4-4   abc1234  PD       0:00      1 (Priority)
65488      work test-4-5   abc1234  PD       0:00      1 (Priority)
65489      work test-5-1   abc1234  PD       0:00      1 (Priority)
65490      work test-5-2   abc1234  PD       0:00      1 (Priority)
65491      work test-5-3   abc1234  PD       0:00      1 (Priority)
65492      work test-5-4   abc1234  PD       0:00      1 (Priority)
65493      work test-5-5   abc1234  PD       0:00      1 (Priority)
65481      work test-3-3   abc1234  PD       0:00      1 (Resources)
65482      work test-3-4   abc1234  PD       0:00      1 (Priority)
65346      work     test   abc1234  PD       0:00      1 (Priority)
63992      work ON-IC-0_   xyz5678  PD       0:00      1 (Priority)
65469      work test-1-1   abc1234   R       0:10      1 h1
65470      work test-1-2   abc1234   R       0:10      1 h1
65471      work test-1-3   abc1234   R       0:10      1 h1
65472      work test-1-4   abc1234   R       0:10      1 h1
65473      work test-1-5   abc1234   R       0:10      1 h2
65474      work test-2-1   abc1234   R       0:10      1 h2
65475      work test-2-2   abc1234   R       0:10      1 h2
65476      work test-2-3   abc1234   R       0:10      1 h2
65477      work test-2-4   abc1234   R       0:10      1 h3
65478      work test-2-5   abc1234   R       0:10      1 h3
65479      work test-3-1   abc1234   R       0:10      1 h3
65480      work test-3-2   abc1234   R       0:10      1 h3
```

Here you can see that there are 27 jobs currently known by the SLURM scheduler.

The 12 jobs that have an *R* in the *ST* column are currently in the running state. They are executing on three different remote nodes, h1, h2, and h3. The rest of the jobs have a *PD* in the *ST* column meaning they are in a pending state. They are pending for different reasons – some do not have sufficient priority to be running yet whereas another is marked as requesting resources that are not yet available.

The *USER* column (perhaps obviously) indicates what user owns the submitted job. Here the abc1234 user owns most of the submitted jobs; another xyz5678 user owns one other that is waiting for access.

The *TIME* column indicates how long the jobs have been running.

Note: The squeue command is one command of many provided by the SLURM scheduler. You can break it down phonetically as *s-queue*. All SLURM commands begin with an *s* (for SLURM). The *queue* part means that this command will display the queue of jobs waiting for or currently consuming resources provisioned by the scheduler.

So the cluster at this point looks 'pretty full', meaning that there are jobs waiting in the queue to get access; the computing resources look fully occupied.

You may have noticed that there was a *PARTITION* column in the squeue output and that all the jobs listed were marked as under the *work* partition.

We'll use another command, the sinfo command, to get another look at the cluster's status and find out more about these partitions:

```
[abc1234@ion ~ []]$ sinfo
PARTITION AVAIL  TIMELIMIT  NODES  STATE NODELIST
debug        up       10:0      1   idle h[8]
work         up 14-00:00:0      7  alloc h[1-7]

premium      up 14-00:00:0      7  alloc h[1-7]
```

Here we see that the cluster is divided into three partitions: *debug*, *work*, and *premium*.

Th*e debug* partition has a small time limit by design of 10 minutes. It's purpose (perhaps obviously) is to handle jobs for debugging when you are first writing your scripts to submit work

The *work* partition is the main partition. The time limit for jobs is 14 days. You can see from the sinfo output above that all of the nodes in that partition are currently allocated.

The *premium* partition is the paid pseudo-partition to allow for the pre-emption of jobs by paid cluster users. Jobs submitted to the *premium* partitio n can pause jobs in the *work* partition to allow for drastically reduced cluster wait times.

Submitting your first job

Submitting jobs to the cluster requires you to have written a script that defines your workload and metadata about it. Lucky for you, we've written a handy example-creator called slurm-make-examples.sh. It will copy some examples into your home directory.

Run:

```
[abc1234@ion ~ []]$ slurm-make-examples.sh
** Placing examples in /home/abc1234/slurm-examples-2011-12-09
...
** Done copying to /home/abc1234/slurm-examples-2011-12-09
** Replacing all instances of 'USER' with 'abc1234'.
** Done replacing in /home/abc1234/slurm-examples-2011-12-09
```

Now take a look at your home directory and change into the newly created examples directory:

```
[abc1234@ion ~ []]$ ls -alh
drwx------ 106 abc1234 abc1234  36K Dec  9 16:36 .
drwxr-xr-x   5 root    root      0  Dec  9 16:28 ..
drwxrwx---   5 abc1234 abc1234 2.0K Dec  9 16:36 slurm-examples-2011-12-09

[abc1234@ion ~ []]$ cd slurm-examples-2011-12-09/

[abc1234@ion slurm-examples-2011-12-09 []]$ ls -alh
total 192K
drwxrwx---   5 abc1234 abc1234 2.0K Dec  9 16:36 .
drwx------ 106 abc1234 abc1234  36K Dec  9 16:36 ..
drwxr-x---   2 abc1234 abc1234 2.0K Dec  9 16:36 example-1-simple-jobs
drwxr-x---   2 abc1234 abc1234 2.0K Dec  9 16:36 example-2-basic-looping
drwxr-x---   2 abc1234 abc1234 2.0K Dec  9 16:36 example-3-job-dependency-and-dynamic-node-claiming
```

There are three examples there. Change directory into the first one and list its contents:

```
[abc1234@ion slurm-examples-2011-12-09 []]$ cd example-1-simple-jobs/

[abc1234@ion example-1-simple-jobs []]$ ls -alh
total 160K
drwxr-x--- 2 abc1234 abc1234 2.0K Dec  9 16:36 .
drwxrwx--- 5 abc1234 abc1234 2.0K Dec  9 16:36 ..
-rwxrwx--- 1 abc1234 abc1234 1.3K Dec  9 16:36 slurm-mpi.sh
-rwxrwx--- 1 abc1234 abc1234 1.2K Dec  9 16:36 slurm-single-core.sh
-rwxrwx--- 1 abc1234 abc1234 1.3K Dec  9 16:36 slurm-smp.sh
```

The file we're going to be working with first is slurm-single-core.sh. It is a *SLURM* job file that describes...

- Metadata about the job we're going to submit
- The payload of the job; the actual work we want to get done.

Let's take a look at it. Run the following command, less slurm-single-core.sh:

```
[abc1234@ion example-1-simple-jobs []]$ less slurm-single-core.sh
#!/bin/bash -l
# NOTE the -l flag!
#

# This is an example job file for a single core CPU bound program
# Note that all of the following statements below that begin
# with #SBATCH are actually commands to the SLURM scheduler.
# Please copy this file to your home directory and modify it
# to suit your needs.
#
# If you need any help, please email rc-help@rit.edu
#

# Name of the job - You'll probably want to customize this.
#SBATCH -J test

# Standard out and Standard Error output files
#SBATCH -o test.output
#SBATCH -e test.output

#SBATCH --mail-user abc1234@rit.edu

# notify on state change: BEGIN, END, FAIL or ALL
#SBATCH --mail-type=ALL

# Request 5 minutes run time MAX, anything over will be KILLED
#SBATCH -t 0:5:0

# Put the job in the "debug" partition and request one core
# "debug" is a limited partition.  You'll likely want to change
# it to "work" once you understand how this all works.
#SBATCH -p debug -n 1

# Job memory requirements in MB
#SBATCH --mem=300

#
# Your job script goes below this line.
#
echo "(${HOSTNAME}) sleeping for 1 minute to simulate work (ish)"
sleep 60
echo "(${HOSTNAME}) Ahhh, alarm clock!"
```

You'll see by the first line, #!/bin/bash, that this is a bash script. As you might already know, any line in a bash script that begins with a # is a comment and is therefore disregarded when the script is running.

However, in this context, any line that begins with #SBATCH is actually a meta-command to the *SLURM* scheduler that informs it how to prioritize, schedule, and place your job.

The last three lines are the 'payload' of the job. In this case it just prints out a statement, goes to sleep for 60 seconds (pretending to work) and then wakes up and prints one last statement. Very important scientific work, don't you agree?

Let's give this script a run. We'll submit it to the SLURM scheduler using the sbatch command, but we need one more piece of information before we do.

Research Computing divvies out resources to users by way of Qualities-of-Service (or QOSes). If you don't know what QOS your account is in, you can run the show-my-qos command. If things are still unclear, you can email rc-help@rit.edu to ask, but you are most likely in the rc or *free* QOS. For each grouping of users, we define two different priority-levels under which you can submit jobs.

Everyone is a member of the free QOS, which lacks core restrictions. You can find your premium QOSes and their core limits with:

```
[abc1234@ion example-1-simple-jobs []]$ show-my-qos

QOS Name                #Cores
------------------------------
foo                         10
bar                          6
```

Submit your job with the following command:

```
[abc1234@ion example-1-simple-jobs []]$ sbatch --qos=free slurm-single-core.sh
Submitted batch job 727
```

You can now check to see that your job is really running in the debug partition by running squeue:

```
[abc1234@ion example-1-simple-jobs []]$ squeue
JOBID PARTITION     NAME      USER  ST       TIME  NODES NODELIST(REASON)
  727     debug     test   abc1234   R       0:27      1 einstein
```

Now that our script is running, we should be able to see its output. Check for it with ls -alh:

```
[abc1234@ion example-1-simple-jobs []]$ ls -alh
total 192K
drwxr-x--- 2 abc1234 abc1234 2.0K Dec  9 17:08 .
drwxrwx--- 5 abc1234 abc1234 2.0K Dec  9 16:36 ..
-rwxrwx--- 1 abc1234 abc1234 1.3K Dec  9 16:36 slurm-mpi.sh
-rwxrwx--- 1 abc1234 abc1234 1.2K Dec  9 16:36 slurm-single-core.sh
-rwxrwx--- 1 abc1234 abc1234 1.3K Dec  9 16:36 slurm-smp.sh
-rw-rw---- 1 abc1234 abc1234   86 Dec  9 17:09 test.output
```

And check its contents with the cat command:

```
[abc1234@ion example-1-simple-jobs []]$ cat test.output
(einstein) sleeping for 1 minute to simulate work (ish)
(einstein) Ahhh, alarm clock!
```

Neat! This is the output that would normally be printed to the screen, printed instead to the contents of the output file we specified in our SLURM job script slurm-single-core.sh. Our code was executed on the remote compute node called einstein and its results were redirected over NFS back to us.

If you've been able to follow the above steps and successfully submit and monitor a job, you might want to check out *Using the Cluster – Advanced Usage*.