Submitting Jobs to the Cluster

Overview

All user processes must be run as cluster jobs in the cluster job queuing system, slurm.

Slurm provides an easy and transparent way to streamline calculations on the cluster and make cluster use more efficient. Slurm offers several user friendly features:

- It has the ability to run parallel calculations by allocating a set of nodes specifically for that task.
- Users running serial jobs with minimal input/output can start the jobs directly from the commandline without resorting to batch files.
- Slurm has the ability to handle batch jobs as well.
- Slurm allows users to run interactive commandline or X11 jobs.

Node Status

sinfo

sinfo reports the state of partitions and nodes managed by Slurm. Example output:

```
$ sinfo

PARTITION        AVAIL  TIMELIMIT  NODES  STATE NODELIST
all*                up   infinite      9  down* rb2u[3,5-12]
all*                up   infinite      5   idle dn[1-2],rb2u[1-2,4]
xeon-6136-256G      up   infinite      2   idle dn[1-2]
xeon-e5-2620-32G    up   infinite      9  down* rb2u[3,5-12]
xeon-e5-2620-32G    up   infinite      3   idle rb2u[1-2,4]
```

In the above, the partition "all" contains all the nodes. There are also partitions for the differing specifications of nodes. Nodes will be listed both in "all" and their individual spec class partition.

The above eample shows that nodes dn1 and dn2 are idle – up and no jobs are running. Nodes rb2u3, rb2u5, rb2u6, rb2u7.... through rb2u12 are all down. If a node is allocated to a job, the status will be "alloc" . If a node is set to run its current jobs and allow no more jobs in preparation of downtime, its status will be set to "drain".

sview

sview is a graphical user interface to get state information for nodes (and jobs).

Job Status

sview

sview is a graphical user interface to get job state information on nodes.

squeue

squeue reports the state of jobs or job steps. By default, it reports the running jobs in priority order and then the pending jobs in priority order.

```
$ squeue

JOBID PARTITION  NAME  USER ST  TIME NODES NODELIST(REASON)
65646     batch  chem  mike  R 24:19     2 adev[7-8]
65647     batch   bio  joan  R  0:09     1 adev14
65648     batch  math  phil PD  0:00     6 (Resources)
```

Each calculation is given a JOBID. This can be used to cancel the job if necessary. The PARTITION field references the node class spec partitions as mentioned above in the "sinfo" documentation. The NAME field gives the name of the program being used for the calculation. The NODELIST field shows which node each calculation is running on. And the NODES field shows the number of nodes in use for that job.

Starting a Job

There are multiple ways to essentially accomplish the same thing. A quick overview:

sbatch and *salloc* allocate resources to the job, while *srun* launches parallel tasks across those resources. When invoked within a job allocation, srun will launch parallel tasks across some or all of the allocated resources. In that case, srun inherits by default the pertinent options of the sbatch or salloc which it runs under. You can then (usually) provide srun different options which will override what it receives by default. Each invocation of srun within a job is known as a job step.

srun can also be invoked outside of a job allocation. In that case, srun requests resources, and when those resources are granted, launches tasks across those resources as a single job and job step.

Load any environment modules **before the srun/sbatch/salloc commands.** These commands will copy your environment as it is at job submission time.

srun

You can start a calculation/job directly from the commandprompt by using srun. This command submits jobs to the slurm job submission system and can also be used to start the same command on multiple nodes. 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 such as memory and disk space.

sbatch

sbatch is used to submit a job script for later execution. The script will typically contain one or more *srun* commands to launch parallel tasks.

salloc

salloc is used to allocate resources for a job in real time. Typically this is used to allocate resources and spawn a shell. The shell is then used to execute srun commands to launch parallel tasks.

srun Examples

```
$ srun -N4 /bin/hostname
rb2u4
dn1
rb2u1
rb2u2
```

In the example above, we use srun to start the command *hostname* on 4 nodes in the cluster. The option -N4 tells slurm to run the job on four nodes of its choice. And we see the output printed by the hostname command of each node that was used.

Piping Data

With many calculations it is important to pipe data in (<) from an input file and pipe date out (>) to an output file. The program may also have command line options as well:

```
$program [options] < input.dat > output.dat
```

One of the nice features of srun is that it preserves this ability to redirect input and output. Just remember that any options directly after srun such as –N will be used by srun. However, any options or piping commands after your program name will be used by the program only.

Dealing with Batch Files

In many cases, you will want to run your calculation on the scratch space for a particular node. This will prevent your calculation from writing to the NSF mounted /home directory and insure that you are not wasting time due to unnecessary data transfer between nodes. However, the srun command doesn't know which node you want to run on or in which directory your calculation will need to run. In these cases, it is essential to write a batch file that will guide your calculation along. The essential steps to include in your batch file are (this example uses a naming convention that this is run #2 on day May 24):

1. The first line of your script file must indicate that this is a script:

   ```
   #!/bin/bash
   ```

2. Create an unique directory in your scratch space for this calculation (substitute your user name is the commands below):

   ```
   mkdir /scratch/user_name/may24_run2/
   ```

3. Report back to a file in your home directory as to which node the calculation is running on and the time it started. This will help you track down the results when it is finished.

```
/usr/bin/hostname > /home/user_name/may24_run2.log
/usr/bin/date >> /home/user_name/may24_run2.log
```

4. Now that we have created the directory, we need to move all the necessary input files over from the home directory.

```
cp /home/user_name/input_store/input.dat /scratch/user_name/may24_run2/
```

5. Hop into the directory we created

```
cd /scratch/user_name/may24_run2
```

6. Start the calculation

```
/home/user_name/bin/cool_program.x < input.dat > may24.out.run
```

7. Report back when the calculation is finished. Leave some info in our home directory log file.

```
echo "Job Done" >> /home/user_name/may24_run2.log
/bin/date >> /home/user_name/may24_run2.log
```

8. Copy results back back to a directory in your home directory (make sure you have previously created the "output_store" folder in your home directory)

```
cp /scratch/user_name/may24_run2/may24.out.run /home/user_name/output_store/
```

9. Clean up the scratch space on the node so as to not fill up the disk

```
rm -rf /scratch/user_name/may24_run2
```

Ensure that your batch file containing the above commands is executable:

```
$ chmod u+x batch_file.run
```

Use srun to submit the job:

```
$ srun batch_file.run
```

You will see a new job listed when you type *squeue* .

Running Parallel Calculations

Often we need to run parallel calculations that take advantage of several of the nodes on the cluster. This can be done using the slurm job submission with a few small modifications. Instead of running the calculation directly with srun, we are only going to use srun to reserve the nodes we need for the calculation. Let's say we want to run a parallel calculation on 4 nodes.

First, we allocate the nodes for the calculation:

```
$ srun -n 4 -N 4 --pty bash
```

The "-n 4" says we will be running 4 tasks. The "-N 4" asks to allocate 4 nodes. And the "–pty bash" says to give us an interactive shell.

After typing the above command, you will see a bash shell prompt on one of the compute nodes. You can now run your command in one of several ways, including the OpenHPC "prun" command (which will execute the "mpirun" command – we can see what prun does by passing the "-v" option to it):

```
$ prun -v a.out
[prun] Master compute host = dn1
[prun] Resource manager = slurm
[prun] Setting env variable: OMPI_MCA_mca_base_component_show_load_errors=0
[prun] Setting env variable: PMIX_MCA_mca_base_component_show_load_errors=0
[prun] Setting env variable: OMPI_MCA_ras=^tm
[prun] Setting env variable: OMPI_MCA_ess=^tm
[prun] Setting env variable: OMPI_MCA_plm=^tm
[prun] Setting env variable: OMPI_MCA_io=romio314
[prun] Launch cmd = mpirun a.out (family=openmpi3)

 Hello, world (4 procs total)
    --> Process #   0 of   4 is alive. -> dn1
    --> Process #   1 of   4 is alive. -> rb2u1
    --> Process #   2 of   4 is alive. -> rb2u2
    --> Process #   3 of   4 is alive. -> rb2u4
```

Interactive Shells

You can allocate an interactive shell on a node for running calculations by hand.

Load any environment modules **before the srun command.**

To just allocate a bash shell:

```
$ srun -n1 --pty bash
```

To allocate a shell with X11 (Option 1 for X11 forwarding) forwarding so that you can use the rappture GUI:

```
$ module load rappture
$ srun -n1 --x11 --pty bash
```

To allocate a shell with X11 (Option 2 for X11 forwarding, which may work better than option1 depending on the software and is slightly more complicated to use):

```
# With this method, you end up in two subshells underneath your main nanolab login:
# nanolab main login -> salloc subshell on nanolab -> subshell via ssh -Y on the allocated node

# Do not load modules until you have connected to the allocated node, as exemplified below:

$ salloc -n1

# You will see output such as:

salloc: Granted job allocation 18820
salloc: Waiting for resource configuration
salloc: Nodes <nodename> are ready for job

$ ssh -Y <nodename_from_above>

# Now load your module
$ module load rappture

# Now run your software

# If you forget the -Y option to ssh, you will see an error such as:
xterm: Can't open display:
xterm: DISPLAY is not set

# When exiting, you will have to exit an extra time.
# First, out of the node that was allocated to you.
# Then out of the salloc command, which will print output such as
salloc: Relinquishing job allocation 18820

# Then a third time from nanolab itself
```

If you did not connect to the head node with X11 forwarding enabled and are using option 1, you will see the following error:

```
$ srun -n1 --x11 --pty bash
srun: error: No DISPLAY variable set, cannot setup x11 forwarding.
```

sbatch Examples

Load any environment modules **before using the sbatch command to submit your job.**

With sbatch, your script file will contain special #SBATCH commands detailing your job requirements. Here is an example:

```
#!/bin/bash

#SBATCH -J test           # job name
#SBATCH -o job.%j.out     # Name of standard output file (%j expands to %jobId)
#SBATCH -N 2              # Number of nodes requested
#SBATCH -n 16             # Total number of mpi tasks requested
#SBATCH -t 01:30:00       # Run tim e(hh:mm:ss)

# Launch MPI-based executable
prun ./a.out
```

You would then submit the above batch execution script with the sbatch command:

```
$ sbatch job.mpi
Submitted batch job 339
```

Stopping a Job

scancel

scancel is used to cancel a pending or running job or job step. To do this we need the JOB ID for the calculation and the command scancel. The JOB ID can be determined using the squeue command described above. To cancel the job with ID=84, just type:

```
$ scancel 84
```

If you rerun squeue you will see that the job is gone.