Slurm Quick Start Tutorial

Resource sharing on a supercomputer dedicated to technical and/or scientific computing is often organized by a piece of software called a resource manager or job scheduler. Users submit jobs, which are scheduled and allocated resources (CPU time, memory, etc.) by the resource manager.

Slurm is a resource manager and job scheduler designed to do just that, and much more. It was originally created by people at the Livermore Computing Center, and has grown into a full-fledge open-source software backed up by a large community, commercially supported by the original developers, and installed in many of the Top500 supercomputers.

Gathering information

Slurm offers many commands you can use to interact with the system. For instance, the `sinfo` command gives an overview of the resources offered by the cluster, while the `squeue` command shows to which jobs those resources are currently allocated.

By default, `sinfo` lists the partitions that are available. A **partition** is a set of **compute nodes** (computers dedicated to... computing) grouped logically. Typical examples include partitions dedicated to batch processing, debugging, post processing, or visualization.

```
# sinfo
PARTITION AVAIL  TIMELIMIT  NODES  STATE  NODELIST
debug*       up   infinite      1   idle  pubrec-cls-01
batch        up   infinite      1   comp* pubrec-bkup-01
batch        up   infinite      1   idle  pubrec-cls-01]
```

In the above example, we see two partitions, named *batch* and *debug*. The debug is the default partition as it is marked with an asterisk. The *batch* partition on the node `pubrec-bkup-01` are being used but the *batch* partition on the node `pubrec-cls-01` are idle.

The `sinfo` command also lists the time limit (column `TIMELIMIT`) to which jobs are subject. On every cluster, jobs are limited to a maximum run time, to allow job rotation and let every user a chance to see their job being started. Generally, the larger the cluster, the smaller the maximum allowed time.

Note

If the output of the `sinfo` command is organised differently from the above, it probably means default options are set through environment variables. Use `printenv|grep ^SINFO` to review them.

The command `sinfo` can output the information in a node-oriented fashion, with the argument `-N`

```
# sinfo -N -l

NODELIST  NODES PARTITION  STATE     CPUS  S:C:T MEMORY TMP_DISK WEIGHT AVAIL_FE
REASON

pubrec-bkup-01 1   batch    completing*  8   1:8:1   1      0    1   (null) none
pubrec-cls-01 1   batch       idle       8   1:8:1   1      0    1   (null) none
pubrec-cls-01 1   debug*       idle       8   1:8:1   1      0    1   (null) none
```

Note that with the `-l` argument, more information about the nodes is provided: number of CPUs, memory, temporary disk (also called scratch space), node weight (an internal parameter specifying preferences in nodes for allocations when there are multiple possibilities), features of the nodes (such as processor type for instance) and the reason, if applicable, for which a node is down.

You can actually specify precisely what information you would like `sinfo` to output by using its `--format` argument. For more details, have a look at the command manpage with `man sinfo`.

The `squeue` command shows the list of jobs which are currently running (they are in the *RUNNING* **state**, noted as 'R') or waiting for resources (noted as 'PD', short for *PENDING*).

```
# squeue
JOBID PARTITION NAME USER ST  TIME  NODES NODELIST(REASON)
12345     debug job1 dave  R   0:21     4 node[9-12]
12346     debug job2 dave PD   0:00     8 (Resources)
12348     debug job3 ed   PD   0:00     4 (Priority)
```

The above output show that is one job running, whose name is *job1* and whose **jobid** is 12345. The jobid is a unique identifier that is used by many Slurm commands when actions must be taken about one particular job. For instance, to cancel job *job1*, you would use `scancel 12345`. Time is the time the job has been running until now. Node is the number of nodes which are allocated to the job, while the `Nodelist` column lists the nodes which have been allocated for running jobs. For pending jobs, that column gives the reason why the job is pending. In the example, job 12346 is pending because requested resources (CPUs, or other) are not available in sufficient amounts, while job 12348 is waiting for job 12346, whose priority is higher, to run. Each job is indeed assigned a **priority** depending on several parameters whose details are explained in section Slurm priorities. Note that the priority for pending jobs can be obtained with the `sprio` command.

There are many switches you can use to filter the output by user `--user`, by partition `--partition` by state `--state` etc. As with the `sinfo` command, you can choose what you want `sprio` to output with the `--format` parameter.

Creating a job

Now the question is: How do you create a job?

A job consists in two parts: **resource requests** and **job steps**. Resource requests consist in a number of CPUs, computing expected duration, amounts of RAM or disk space, etc. Job steps describe **tasks** that must be done, software which must be run.

The typical way of creating a job is to write a **submission script**. A submission script is a shell script, e.g. a Bash script, whose comments, if they are prefixed with *SBATCH*, are understood by Slurm as parameters describing resource requests and other submissions options. You can get the complete list of parameters from the sbatch manpage `man sbatch`.

Important

The `SBATCH` directives must appear at the top of the submission file, before any other line except for the very first line which should be the shebang (e.g. `#!/bin/bash`).

The script itself is a job step. Other job steps are created with the `srun` command.

For instance, the following script, hypothetically named *submit.sh*,

```
#!/bin/bash
#
#SBATCH --job-name=test
#SBATCH --output=res.txt
#
#SBATCH --ntasks=1
#SBATCH --time=10:00
#SBATCH --mem-per-cpu=100

srun hostname
srun sleep 60
```

would request one CPU for 10 minutes, along with 100 MB of RAM, in the default queue. When started, the job would run a first job step `srun hostname`, which will launch the UNIX command `hostname` on the node on which the requested CPU was allocated. Then, a second job step will start the `sleep` command. Note that the `--job-name` parameter allows giving a meaningful name to the job and the `--output` parameter defines the file to which the output of the job must be sent.

Once the submission script is written properly, you need to **submit** it to slurm through the `sbatch` command, which, upon success, responds with the *jobid* attributed to the job. (The dollar sign below is the shell prompt)

```
$ sbatch submit.sh
sbatch: Submitted batch job 99999999
```

The job then enters the queue in the *PENDING* state. Once resources become available and the job has highest priority, an **allocation** is created for it and it goes to the RUNNING state. If the job completes correctly, it goes to the *COMPLETED* state, otherwise, it is set to the *FAILED* state.

Interestingly, you can get near-realtime information about your running program (memory consumption, etc.) with the `sstat` command, by introducing `sstat -j jobid`. You can select what you want `sstat` to output with the `--format` parameter. Refer to the manpage for more information `man sstat`.

Upon completion, the output file contains the result of the commands run in the script file. In the above example, you can see it with `cat res.txt` command.

This example illustrates a *serial* job which runs a single CPU on a single node. It does not take advantage of multi-processor nodes or the multiple compute nodes available with a cluster. The next sections explain how to create parallel jobs.

Interactive jobs

Slurm jobs are normally batch jobs in the sense that they are run unattended. If you want to have a direct view on your job, for tests or debugging, you have two options.

If you need simply to have an interactive Bash session on a compute node, with the same environment set as the batch jobs, run the following command:

```
srun --pty bash
```

Doing that, you are submitting a 1-CPU, default memory, default duration job that will return a Bash prompt when it starts.

If you need more flexibility, you will need to use the salloc command. The `salloc` command accepts the same parameters as `sbatch` as far as resource requirement are concerned. Once the allocation is granted, you can use `srun` the same way you would in a submission script.

Example: Run hostname in an interactive allocation:

```
$ salloc

salloc: Granted job allocation 147
```

blocks here until job runs

```
salloc: Granted job allocation 147

$ srun hostname

pubrec-cls-01
```

Run it again

```
$ srun hostname

pubrec-cls-01
```

Now exit the job and allocation

```
$ exit

exit

salloc: Relinquishing job allocation 147
```

Like `srun` in the first example, `salloc` defaults to asking for one node of the default queue charging the default account. Once the job runs and the prompt appears, any further commands are run within the job's allocated resources until exit is invoked.

Cancel a job, whether it is pending in the queue or running

```
$ scancel <job_ID>
```