Simple Linux Utility for Resource Management (SLURM)

Overview

HPC cluster are managed by the Simple Linux Utility for Resource Management (SLURM). SLURM is an open-source tool that performs cluster management and job scheduling for Linux clusters. Jobs are submitted to the resource manager, which queues them until the system is ready to run them. SLURM selects which jobs to run, when to run them, and how to place them on the compute node, according to a predetermined site policy meant to balance competing user needs and to maximize efficient use of cluster resources. The resource manager divides a cluster into logical units called partitions by SLURM and generally known as *queues* in other queueing systems. Different partitions may contain different nodes, or they may overlap; they may also impose different resource limitations. The CoM HPC environment provides several partitions and there is no default; each job must request a partition. Our examples will use the “economy” partition for all jobs. To determine which other queues are available to your group, log in to the HPC System and type queues at a Linux command-line prompt.

SLURM architecture

SLURM has a controller process (called a daemon) on a head node and a worker daemon on each of the compute nodes. The controller is responsible for queueing jobs, monitoring the state of each node, and allocating resources. The worker daemon gathers information about its node and returns that information to the controller. When assigned a user job by the controller, the worker daemon initiates and manages the job. SLURM provides the interface between the user and the cluster. To submit a job to the cluster, you must request the appropriate resources and specify what you want to run with a SLURM Job Command File. SLURM performs three primary tasks:

1. It manages the queue(s) of jobs and settles contentions for resources;
2. It allocates a subset of nodes or cores for a set amount of time to a submitted job;
3. It provides a framework for starting and monitoring jobs on the subset of nodes/cores.

Batch job scripts are submitted to the SLURM Controller to be run on the cluster. A batch job script is simply a shell script containing*directive*s that specify the resource requirements (e.g. the number of cores, the maximum runtime, partition specification, etc.) that your job is requesting along with the set of commands required to execute your workflow on a subset of cluster compute nodes.  When the script is submitted to the resource manager, the controller reads the directives, ignoring the rest of the script, and uses them to determine the overall resource request.  It then assigns a priority to the job and places it into the queue.  Once the job is assigned to a worker, the job starts as an ordinary shell script on the "master" node, in which case the directives are treated as comments.  For this reason it is important to follow the format for directives exactly.

The remainder of this tutorial will focus on the SLURM command line interface. More detailed information about using SLURM can be found in the [official SLURM documentation](http://slurm.schedmd.com/documentation.html).

Displaying job status

The **squeue** command is used to obtain status information about all jobs submitted to all queues. Without any specified options, the **squeue** command provides a display which is similar to the following:

JOBID PARTITION NAME USER ST TIME NODES NODELIST(REASON)

----------------------------------------------------------------------------------

12345 serial myHello mst3k R 5:31:21 4 udc-ba33-4a, udc-ba33-4b, uds-ba35-22d, udc-ba39-16a

12346 economy bash mst3k R 2:44 1 udc-ba30-5

The fields of the display are clearly labeled, and most are self-explanatory. The TIME field indicates the elapsed walltime (hrs:min:secs) that the job has been running. Note that JOBID 12346 has the name bash, which indicates it is an interactive job. In that case, the TIMEfield provides the amount of walltime during which the interactive session has be open (and resources have been allocated). The ST field lists a code which indicates the state of the job. Commonly listed states include:

* PD PENDING: Job is waiting for resources;
* R RUNNING: Job has the allocated resources and is running;
* S SUSPENDED: Job has the allocated resources, but execution has been suspended.

A complete list of job state codes is available [here](https://computing.llnl.gov/linux/slurm/squeue.html).

Submitting a job

Job scripts are submitted with the sbatch command, e.g.:

% sbatch hello.slurm

The job identification number is returned when you submit the job, e.g.:

% sbatch hello.slurm

Submitted batch job 18341

Canceling a job

SLURM provides the **scancel** command for deleting jobs from the system using the job identification number:

% scancel 18341

If you did not note the job identification number (JOBID) when it was submitted, you can use **squeue** to retrieve it.

% squeue -u mst3k

JOBID PARTITION NAME USER ST TIME NODES NODELIST(REASON)

-------------------------------------------------------------------------------------

18341 serial myHello mst3k R 0:01 1 udc-ba30-5

For further information about the squeue command, type **man** squeue on the cluster front-end machine or see the [SLURM Documentation](http://http/slurm.schedmd.com/documentation.html).

Job accounting data

When submitting a job to the cluster for the first time, the walltime requirement should be overestimated to ensure that SLURM does not terminate the job prematurely. After the job completes, you can use **sacct** to get the total time that the job took. Without any specified options, the **sacct** command provides a display which is similar to the following:

JobID JobName Partition Account AllocCPUS State ExitCode

------------------------------------------------------------------------

18347 hello2.sl+ economy default 1 COMPLETED 0:0

18347 .batch batch default 1 COMPLETED 0:0

18348 hello2.sl+ economy default 1 COMPLETED 0:0

18348 .batch batch default 1 COMPLETED 0:0

18352 bash economy default 1 COMPLETED 0:0

18352 .0 python default 1 COMPLETED 0:0

18353 bash economy default 1 RUNNING 0:0

18353 .0 python default 1 COMPLETED 0:0

18353 .1 python default 1 COMPLETED 0:0

18353 .2 python default 1 COMPLETED 0:0

To include the total time, you will need to customize the output by using the format options. For example, the command

% sacct --format=jobID --format=jobname --format=Elapsed --format=state

yields the following display:

JobID JobName Elapsed State

------------------------------------------

18347 hello2.sl+ 00:54:59 COMPLETED

18347.batch batch 00:54:59 COMPLETED

18347.0 orted 00:54:59 COMPLETED

18348 hello2.sl+ 00:54:74 COMPLETED

18348.batch batch 00:54:74 COMPLETED

18352 bash 01:02:93 COMPLETED

18352.0 python 00:21:27 COMPLETED

18353 bash 02:01:05 RUNNING

18353.0 python 00:21:05 COMPLETED

18353.1 python 00:17:77 COMPLETED

18353.2 python 00:16:08 COMPLETED

The Elapsed time is given in hours, minutes, and seconds, with the default format of hh:mm:ss. The Elapsed time can be used as an estimate for the amount of time that you request in future runs; however, there can be differences in timing for a job that is run several times. In the above example, the job called python took 21 minutes, 27 seconds to run the first time (JobID 18352.0) and 16 minutes, 8 seconds the last time (JobID 18353.2). Because the same job can take varying amounts of time to run, it would be prudent to increase Elapsed time by 10% to 25% for future walltime requests. Requesting a little extra time will help to ensure that the time does not expire before a job completes.

Job scripts for parallel programs

Distributed memory jobs

If the executable is a parallel program using the Message Passing Interface (MPI), then it will require multiple processors of the cluster to run. This information is specified in the SLURM nodes resource requirement. The script mpiexec is used to invoke the parallel executable. This example is a SLURM job command file to run a parallel (MPI) job using the OpenMPI implementation:

#!/bin/bash

#SBATCH --nodes=2

#SBATCH --ntasks-per-node=20

#SBATCH --time=12:00:00

#SBATCH --output=output\_filename

#SBATCH --partition=parallel

module load openmpi/gcc

mpiexec ./parallel\_executable

In this example, the SLURM job file is requesting two nodes with four tasks per node (for a total of 8 processors).  Both OpenMPI and MVAPICH2 are able to obtain the number of processes and the host list from SLURM, so these are not specified.  In general, MPI jobs should use all of a node so we'd recommend ntasks-per-node=20 on the parallel partition, but some codes cannot be distributed in that manner so we are showing a more general example here.

SLURM can also place the job freely if the directives specify *only* the number of tasks.

#!/bin/bash

#SBATCH --ntasks=8

#SBATCH --time=12:00:00

#SBATCH --output=output\_filename

#SBATCH --partition=parallel

module load openmpi/gcc

mpiexec ./parallel\_executable

Threaded jobs (OpenMP or pthreads)

SLURM considers a task to correspond to a process.  This example is for OpenMP:

#!/bin/bash

#SBATCH --ntasks=1

#SBATCH --cpus-per-task=20

#SBATCH --time=12:00:00

#SBATCH --output=output\_filename

#SBATCH --partition=parallel

module load gcc

**export** OMP\_NUM\_THREADS=$SLURM\_CPUS\_PER\_TASK

./threaded\_executable

Hybrid

The following example runs a total of 32 MPI processes, 4 on each node, with each task using 5 cores for threading.  The total number of cores utilized is thus 160.

#!/bin/bash

#SBATCH --ntasks=32

#SBATCH --ntasks-per-node=4

#SBATCH --cpus-per-task=5

#SBATCH --time=12:00:00

#SBATCH --output=output\_filename

#SBATCH --partition=parallel

module load mvapich2/gcc

**export** OMP\_NUM\_THREADS=$SLURM\_CPUS\_PER\_TASK

mpiexec ./hybrid\_executable

Job Arrays

A large number of jobs can be submitted through one request if all the files used follow a strict pattern.  For example, if input files are named input\_1.dat, ... , input\_1000.dat, we could write a job script requesting the appropriate resources for a single one of these jobs with

#!/bin/bash

#SBATCH --ntasks=1

#SBATCH --time=1:00:00

#SBATCH --output=result\_%a.out

#SBATCH --partition=economy

/myprogram < input\_**${SLURM\_ARRAY\_TASK\_ID}**.dat

In the output file name, %a is the placeholder for the array ID.  We submit with

sbatch --array=1-1000 myjob.sh

The system automatically submits 1000 jobs, which will all appear under a single job ID with separate array IDs.

Specifying job dependencies

With the sbatch command, you can invoke options that prevent a job from starting until a previous job has finished. This constraint is especially useful when a job requires an output file from another job in order to perform its tasks. The --dependency option allows for the specification of additional job attributes. For example, suppose that we have two jobs where job\_2 must run after job\_1 has completed. Using the corresponding SLURM command files, we can submit the jobs as follows:

% sbatch job\_1.slurm

Submitted batch job 18375

% sbatch --dependency=afterok:18375 job\_2.slurm

Notice that the --dependency has its own condition, in this case afterok. We want job\_2 to start only after the job with id 18375 has completed successfully. The afterok condition specifies that dependency. Other commonly-used conditions include the following:

* after: The dependent job is started after the specified job\_id starts running;
* afterany: The dependent job is started after the specified job\_id terminates either successfully or with a failure;
* afternotok: The dependent job is started only if the specified job\_id terminates with a failure.

More options for arguments of the dependency condition are detailed in the manual pages for **sbatch** found [here](https://computing.llnl.gov/linux/slurm/sbatch.html) or by typing**man** sbatch at the Linux command prompt.

We also are able to see that a job dependency exists when we view the job status listing, although the explicit dependency is not stated, e.g.:

% squeue

JOBID PARTITION NAME USER ST TIME NODES NODELIST(REASON)

---------------------------------------------------------------------------------

18375 economy job\_2.sl mst3k PD 0:00 1 (Dependency)

18374 economy job\_1.sl ms53k R 0:09 1 udc-ba30-5

Submitting an interactive job

To submit an interactive job, first request the desired resource with the **salloc** command.

% salloc -p economy --ntasks=16

salloc: Pending job allocation 25394

salloc: job 25394 queued and waiting **for** resources

There may be a slight delay before the resources are available. There will be a notification when the resources have been allocated.

salloc: job 25394 has been allocated resources

salloc: Granted job allocation 25394

At that point, you may run the job.

% srun -n 16 python myPYprog.py

The allocation will launch a user's shell. When the job has completed, you will need to relinquish the resources by exiting the shell.

% **exit**

salloc: Relinquishing job allocation 25394

**Note:** The **salloc** command performs an SSH onto a separate host. However, the new shell may not be a login shell and would not inherit the environment of the shell in which salloc was run. Be aware that environment variables used by the application may need to be set manually. When an interactive job is executed, the standard input, output, and error streams of the job are displayed in the user's shell, where the job is launched. Because the job is interactive, thesruncommand will not return the command-line prompt until the job is finished.  In addition, an interactive SLURM job will not terminate until the user exits the terminal session.

A simpler method is to use the locally-written command ijob:

ijob

Usage: ijob [-c] [-p] [-J] [-w] [-t] [-m] [-A]

Arguments:

  -A: account to use (required: no default)  
  -p: partition to run job in (default: serial)  
  -c: number of CPU cores to request (required: no default)  
  -m: MB-amount of memory to request per-core (default 2000)  
  -J: job name (default: interactive)  
  -w: node name  
  -t: time limit (default: 4:00:00)

ijob is a wrapper around sallocand srun, with appropriate options to start a bash shell on the remote node.

The allocated node(s) will remain reserved as long as the terminal session is open, up to the walltime limit, so **it is extremely important that users exit their interactive sessions as soon as their work is done** so that their nodes are returned to the available pool of processors and the user is not charged for unused time.

Job submission policies

Please refer to our Usage Policies for more information. Researchers with extraordinary needs for the cluster, either in terms of extended compute time or number of nodes, [contact UVACSE](http://www.uvacse.virginia.edu/contact) to discuss making special arrangements to meet those needs.

Common SLURM options and environment variables

Options

Note that most SLURM options have two forms, a short (single-letter) form that is preceded by a single hyphen and followed by a space, and a longer form preceded by a double hyphen and followed by an equals sign.

* Number of nodes: -N <n>    or --nodes=<n>
* Number of cores per node: --ntasks-per-node=<n>
* Total number of tasks: -n <n> or--ntasks=<n>
* Total memory per node in megabytes (not needed in most cases): --mem=<M>
* Memory per core in megabytes (not needed in most cases): --mem-per-cpu=<M>
* Wallclock time: -t d-hh:mm:ssor--time=d-hh:mm:ss
* Partition requested: -p <part> or--partition=<part>
* Rename output file (the default is slurm-<jobid>.out and standard output and standard error are joined): -o <outfile> or--output=<outfile>
* Separate standard error and standard output and rename standard error: -e <errfile>or--error=<errfile>
* Account to be charged: -A <account> or--account=<account>

Environment variables

These are the most basic; there are many more.  By default SLURM changes to the directory from which the job was submitted, so the SLURM\_SUBMIT\_DIR environment variable is usually not needed.

* SLURM\_JOB\_ID
* SLURM\_SUBMIT\_DIR
* SLURM\_JOB\_PARTITION
* SLURM\_JOB\_NODELIST

Sample SLURM command scripts

In this section are a number of sample SLURM command files for different types of jobs.

Gaussian 03

This is a SLURM job command file to run a Gaussian 03 batch job. The Gaussian 03 program input is in the file gaussian.in and the output of the program will go to the file gaussian.out.

#!/bin/bash

#SBATCH --tasks=1

#SBATCH -t 160:00:00

#SBATCH -o gaussian.out

#SBATCH -p serial

#SBATCH -A mygroup

module load gaussian/g98

# Copy Gaussian input file to compute node scratch space

LS="/scratch/mst3k"

**cd** $LS

cp /home/mst3k/gaussian/gaussian.in .

# Define Gaussian scratch directory as compute node scratch space export

GAUSS\_SCRDIR=$LS

g03 < $LS/gaussian.in > $LS/gaussian.out

IMSL

This is a SLURM job command file to run a serial job that is compiled with the IMSL libraries.

#!/bin/bash

#SBATCH -n 1

#SBATCH -t 01:00:00

#SBATCH -o output\_filename

#SBATCH -p economy

#SBATCH -A mygroup

module load imsl

./myprogram

MATLAB

This example is for a serial (one core) Matlab job.

#!/bin/bash

#SBATCH -N 1

#SBATCH -n 1

#SBATCH -t 01:00:00

#SBATCH -o output\_filename

#SBATCH -p economy

#SBATCH -A mygroup

module load matlab

matlab -nojmv -nodisplay -nosplash -singleCompThread -r "Mymain(myvar1s);exit"

R

This is a SLURM job command file to run a serial R batch job.

#!/bin/bash

#SBATCH -n 1

#SBATCH -t 01:00:00

#SBATCH -o myRprog.out

#SBATCH -p parallel

#SBATCH -A mygroup

module load R/openmpi/3.1.1

Rscript myRprog.R

This is a SLURM job command file to run a parallel R batch job using the Rmpi or parallel packages.

#!/bin/bash

#SBATCH -n 2

#SBATCH --ntasks-per-node=3

#SBATCH -t 00:30:00

#SBATCH -o myRprog.out

#SBATCH -p parallel

#SBATCH -A mygroup

module load R/openmpi/3.1.1

mpirun Rscript myRprog.R