Slurm Tutorial

Table of Contents

1. Overview
2. What is a Workload Manager
3. Basic Concepts
   1. Jobs
   2. Queues and queue limits
4. Basic Functions
   1. Building a Job Script
   2. Submitting Jobs
   3. Monitoring Jobs
   4. Job Status and Job Codes
   5. Holding and Releasing Jobs
   6. Canceling Jobs
   7. Changing Job Parameters
   8. Output Files
   9. When Will Your Job Start?
   10. Job Expiration
5. Displaying Configuration and Accounting Information
6. Parallel Jobs and the srun Command
7. Running Multiple Jobs from a Single Job Script
8. Running on Serial Clusters
9. Batch Commands Summary
10. References and More Information

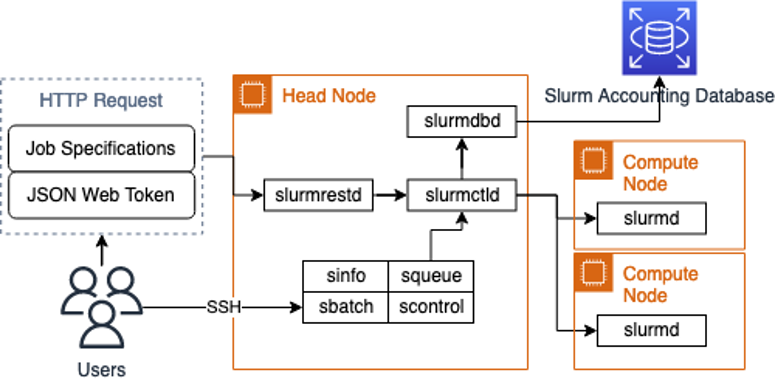
Overview

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 and/or non-exclusive access to resources (compute nodes) to users for some duration of time so they can perform work. Finally, it arbitrates contention for resources by managing a queue of pending work. Optional plugins can be used for [accounting](https://slurm.schedmd.com/accounting.html), [advanced reservation](https://slurm.schedmd.com/reservations.html), [gang scheduling](https://slurm.schedmd.com/gang_scheduling.html) (time sharing for parallel jobs), backfill scheduling, [topology optimized resource selection](https://slurm.schedmd.com/topology.html), [resource limits](https://slurm.schedmd.com/resource_limits.html) by user or bank account, and sophisticated [multifactor job prioritization](https://slurm.schedmd.com/priority_multifactor.html) algorithms.

What is a Workload Manager?

* HPC clusters computations are performed on compute nodes. Programs called resource managers, workload managers, or job schedulers are used to allocate processors and memory on compute nodes for users’ jobs. Slurm Workload Manager is utilized for this purpose.
* Tasks commonly performed by the Slurm Workload Manager:
  + Provide users a way to specify and submit work as "jobs"
  + Prioritize, schedule and run jobs
  + Provide a means for users to monitor, modify and interact with jobs
  + Manage, allocate and provide access to available machine resources
  + Manage pending work in job queues
  + Monitor and troubleshoot jobs and machine resources
  + Provide accounting and reporting facilities for jobs and machine resources
  + Efficiently balance work over machine resources; minimize wasted resources

Generalized architecture and workflow of the Slurm Workload Manager:



|  |  |
| --- | --- |
| * + Logs into cluster   + Creates job script and submits it to workload manager   + Monitors and interacts with job via workload manager   + Tracks job and cluster information   + Sends jobs to compute node daemons for actual execution | * + Typically runs on a separate server as multiple processes   + Receives job submissions, commands, queries from user   + Matches job requirements to available machine resources   + Evaluates, prioritizes and queues jobs |

Slurm

Slurm

* Slurm is an open-source cluster management and job scheduling system for Linux clusters.
* Used on many of the world's TOP500 supercomputers.
* [SchedMD](https://schedmd.com/) is the primary source for Slurm downloads and documentation. SchedMD also offers development and support services for Slurm.
* Some history:
  + Slurm began development as a collaborative effort primarily by Lawrence Livermore National Laboratory (LLNL), Linux NetworX, Hewlett-Packard and Groupe Bull as a free software resource manager in 2001.
  + Acronym originally stood for Simple Linux Utility for Resource Management.
* Documentation:
  + SchedMD website: [schedmd.com](https://schedmd.com/) - see the Documentation section.

Basic Concepts

Jobs

**Simple Definition**

* To a user, a job can be simply described as a request for compute resources needed to perform computational work.
* Jobs typically specify what resources are needed, such as type of machine, number of machines, job duration, amount of memory required, account to charge, etc.
* Jobs are submitted to the Workload Manager by means of a job script. **Slightly More Complex Definition**
* A job contains the following components:
  + **Consumable resources**
  + **Resource and job constraints**
  + **Execution environment**
  + **Credentials**
* **Consumable resources**: Any object which can be utilized ( i.e., consumed and thus made unavailable to another job) by, or dedicated to a job is considered to be a resource. Common examples of resources are a node's physical memory, CPUs or local disk. Network adapters, if dedicated, may also be considered a consumable resource.
* **Resource and job constraints**: A set of conditions which must be fulfilled for the job to start. For example:
  + Type of node/machine
  + Number of processors
  + Speed of processor
  + Partition
  + Features, such as disk, memory, adapter, etc.
  + When the job may run
  + Starting job relative to a particular event (i.e., start after job X successfully completes)
* **Execution environment**: A description of the environment in which the executable is launched. This environment may include attributes such as the following:
  + An executable
  + Command line args
  + Input file
  + Output file
  + Local user id
  + Local group id
  + Process resource limits
* **Credentials**: With workload managers, credential-based policies and limits are often established. At submit time, jobs are associated with several credentials which subject the job to various polices and grant it various types of access. For example, querying a job shows that it possesses the following credentials:

Creds: user:user1 group:submitters account:va class:prod qos:normal

* + user: automatically assigned as your login userid.
  + group: automatically assigned as your login group.
  + account: automatically assigned as your default bank.
  + qos: "Quality of Service". Provides for assigning special services. The default QoS of "normal" is assigned. Other options include "expedite" and "standby".

Queues and Queue Limits

Queues:

* The majority of nodes on a production system are designated as compute nodes.
* **Note**: Login nodes are a shared, limited resource not intended for production work. They are not associated with any queue.

|  |  |
| --- | --- |
|  |  |

Why in the World Won't My Job Run?

* A user submits a job requesting 16 nodes when 50 nodes are shown as available/idle. However, the job sits in the queue and doesn't run. Why?
* Aside from any "user error" related reasons, there are several other, sometimes complicated, reasons.
* Probably the most important reason is the underlying mechanism used by the batch system to determine when/if a job should run.
* The Workload Manager has been programmed to use a "Fair Share with Half-Life Decay of Usage" algorithm for determining a job's eligibility to run.

Other Considerations

* Jobs may not start if they request node and/or and time resources which exceed queue limits:
  + How long a job may run
  + How many nodes a job may use

Number of jobs that may run simultaneously per user

Building a Job Script

The Basics

* Users submit jobs to the Workload Manager for scheduling by means of a job script.
* A job script is a plain text file that you create with your favorite editor.
* Job scripts can include any/all the following:
  + Commands, directives and syntax specific to a given batch system
  + Shell scripting
  + References to environment variables
  + Names of executable(s) to run
  + Comment lines and white space
* Simple Slurm job control scripts appear below:

Options:

* There are a wide variety of options that can be used in your job script. Some of the more common/useful options are shown below.

Usage Notes

* All #SBATCH lines must come before shell script commands.
* Uppercase vs. lowercase:
  + Always use uppercase for the #SBATCH tokens. Otherwise, the token will (usually) be ignored with no error message resulting in the default setting.
  + The parameters specified by both tokens are case sensitive
* Batch scheduler syntax is parsed upon job submission. Shell scripting is parsed at runtime. Therefore, it is entirely possible to successfully submit a job that has shell script errors that won't fail until the job actually runs.
* Do not submit binary executables directly (without a script) as they will fail.
* The srun command is required to launch parallel jobs.
* Include your preferred shell as the first in your batch script. Otherwise, your job will be rejected. For example:

#!/bin/bash

Submitting Jobs

Job Submission Commands

* The sbatch command is used to submit your job script to the Workload Manager. Upon successful submission, the job's ID is returned and it is spooled for execution.
* These commands accept the same options as the #SBATCH / #MSUB tokens in a batch script.
* Examples:

|  |
| --- |
| % sbatch yourjobscript  Submitted batch job 12345  % sbatch -p pdebug -A physics yourjobscript  Submitted batch job 12345 |

Usage Notes

Use sbatch to submit job scripts with #SBATCH syntax

* After you submit your job script, changes to the contents of the script file will have no effect on your job because it has already been spooled to system file space.
* Users may submit and queue as many jobs as they like, up to a reasonable configuration defined limit.
* The default directory is where you submit your job from. If you need to be in another directory, then you will need to explicitly cd to, or set the working directory with an #SBATCH option.

Environment Variables

* Most of your usual login environment variables are exported to your job's runtime environment.
* There are #SBATCH options that allow you to explicitly specify environment variables to export, in case they are not exported by default.
* Slurm provides several environment variables that allow you to specify/query #SBATCH options and other job behavior.

Passing Arguments to Your Job

* Workload Managers do not provide a convenient way to pass arguments to your job.

Examples:

This works:

% setenv NODES 2

% sbatch -N $NODES yourscript

% cat yourscript

#!/bin/bash

srun -N NODES hostname

Sample output:

result01

result02

This does not work:

% setenv NODES 2

% sbatch yourscript

% cat yourscript

#!/bin/bash

#SBATCH -N $NODES

srun -N $NODES hostname

Sample output:

sbatch: error: ‘$NODES” is not a valid node count

sbatch: error: invalid node count `$NODES'

|  |  |
| --- | --- |
|  |  |

Monitoring Jobs

Multiple Choices

* There are several different job monitoring commands.
* The more commonly used job monitoring commands are summarized in the table below, with example output following.

|  |  |
| --- | --- |
| [squeue](https://hpc.llnl.gov/banks-jobs/running-jobs/slurm#squeue) | Displays one line of information per job by default. Numerous options. |
| [scontrol show job *jobid*](https://hpc.llnl.gov/banks-jobs/running-jobs/slurm#scontrolshowjob) | Provides detailed information about a specific job. |
| [sprio -l](https://hpc.llnl.gov/banks-jobs/running-jobs/slurm#sprio) | Displays a list of queued jobs, their priority, and the primary factors used to calculate job priority. |
| [sinfo](https://hpc.llnl.gov/banks-jobs/running-jobs/slurm#sinfo) | Displays state information about a cluster's queues and nodes |

squeue

* Shows one line of information per job, both running and queued.
* Numerous options for additional/customized output
* Common/useful options:
  + -j shows information for a specified job only
  + -l shows additional job information
  + -o provides customized output - see man page
  + -u shows jobs for a specified user only
* Examples below (some output omitted to fit screen):

|  |
| --- |
| % squeue |

showq

* Shows one line of information per job, both running and queued by default.
* Common/useful options:
  + -r shows only running jobs
  + -i shows only idle jobs
  + -b shows only blocked jobs
  + -c shows recently completed jobs
  + -u shows jobs for a specified user only
* Examples below (some output omitted to fit screen):

|  |
| --- |
| % showq |

scontrol show job

* Similar to checkjob
* Command can not be used with completed jobs.
* Example below:

|  |
| --- |
| % scontrol show job 123456 |

sprio -l

* Displays a list of queued jobs, their priority, and the primary factors used to calculate job priority.
* These commands show identical output.
* To sort the job list by priority, use the command:

% sprio -l | sort -r -k 4,4

* Useful for determining where your jobs are queued relative to other jobs. Highest priority jobs are at the top of the list.
* Man pages:
  + sprio
* Example below (some output omitted to fit screen):

|  |
| --- |
| % sprio -l |

sinfo

* Displays state information about a cluster's queues and nodes
* Numerous options for additional/customized output

|  |  |
| --- | --- |
| BatchHold SystemHold UserHold JobHeldUser | Job is idle and is not eligible to run due to a user, admin, or batch system hold. |
| Canceling CA  CANCELLED | Job is cancelled or in the process of being cancelled. |
| Completed CD  COMPLETED CG  COMPLETING | Job is in the process of, or has completed running. |
| Deferred | Job can not be run for one reason or another, however it will continue to evaluate the job periodically for run eligibility. |
| Depend Dependency | Job can not run because it has a dependency. |
| F  FAILED | Job terminated with non-zero exit code or other failure condition. |
| NF  NODE\_FAIL | Job terminated due to failure of one or more allocated nodes. |
| Idle | Job is queued and eligible to run but is not yet executing. |
| Migrated | This is a transitional state that indicates that the job is in being handed off to the native Slurm resource manager on a specific machine in preparation for running. |
| NotQueued | Indicates a system problem in most cases. |
| PD  PENDING | Job is awaiting resource allocation. |
| Priority | One or more higher priority jobs exist for this partition. |
| Removed | Job has run to its requested walltime successfully but has been canceled by the scheduler or resource manager due to exceeding its walltime or violating another policy; includes jobs canceled by users or administrators either before or after a job has started. |
| Resources | The job is waiting for resources to become available. |
| Running R  RUNNING | Job is currently executing the user application. |
| Staging | The job has been submitted to the batch system for it to run but the batch system has not confirmed yet that the job is actually running. |
| Starting | The batch system has attempted to start the job and the job is currently performing pre-start tasks which may including provisioning resources, staging data, executing system pre-launch scripts, etc. |
| Suspended S  SUSPENDED | Job was running but has been suspended by the scheduler or an admin. The user application is still in place on the allocated compute resources, but it is not executing. |
| TimeLimit TO  TIMEOUT | Job terminated upon reaching its time limit. |
| Vacated | Job canceled after partial execution due to a system failure. |
|  |  |

* Common/useful options:
  + -s summarizes queue information
* Examples below:

|  |
| --- |
| % sinfo |

Job States and Status Codes

* Job state and status codes usually appear in the output of job monitoring commands. Most are self-explanatory.
* For details, consult the man page for the relevant command.
* The table below describes commonly observed job state and status codes.

Basic Functions

Holding and Releasing Jobs

Holding Jobs:

* Users can place their jobs in a "user hold" state several ways:

|  |  |
| --- | --- |
| Job script | #SBATCH -H |
| Command line | sbatch -H *jobscript* |
| Command line (queued job) | scontrol hold *jobid* |

* Jobs placed in a user hold state will be shown as such in the output of the various job monitoring commands.
* Running jobs cannot be placed on hold.
* Note that jobs can be placed on system hold status by the workload manager or by system administrators. Not covered here.
* Examples (Slurm):

Placing a job on hold at submission time:

% sbatch -H myjob

Submitted batch job 12345

% squeue -j 12345

JOBID PARTITION NAME USER ST TIME NODES NODELIST(REASON)

12345 pbatch T4.CMD joeuser PD 0:00 2 (JobHeldUser)

Placing a job on hold after it was submitted:

% sbatch myjob

Submitted batch job 12345

% squeue -j 12345

JOBID PARTITION NAME USER ST TIME NODES NODELIST(REASON)

12345 pbatch T3.CMD joeuser PD 0:00 128 (Priority)

% scontrol hold 12345

% squeue -j 12345

Releasing Jobs:

|  |
| --- |
| To release a queued job from a user hold statescontrol release *jobid* |

* Example (Slurm):
* % squeue -j 12345
* % scontrol release 12345
* % squeue -j 12345
* To cancel either running or queued jobs:

|  |
| --- |
| scancel *jobid* |

* Both scancel and canceljob can be used to cancel multiple jobs at the same time.
* The scancel command has a number of options to specify criteria for job cancellation.

Changing Job Parameters

* Only a few job parameters can be changed after a job is submitted. These parameters include:
  + job dependency (change to "none")
  + queue
  + job name
  + number of nodes (decrease only)
  + user priority
  + wall clock limit
  + parallel file system dependency
  + account
  + qos
* These parameters can only be changed for queued (non-running) jobs.
* Examples:

|  |  |
| --- | --- |
| Syntax | scontrol update job=*jobid* *parameter*=*value* |
| dependency (set to "none") | scontrol update job=651070 depend=none |
| Queue | scontrol update job=651070 partition=pdebug |
| job name | scontrol update job=651070 name=alphaScan |
| nodes (decrease) | scontrol update job=651070 numnodes=4 |
| user priority | scontrol update job=651070 prio=100 |
| wall clock limit | scontrol update job=651070 timelimit=2:00:00 |
| parallel file system | scontrol update job=651070 gres=ignore |
| Account | scontrol update job=651070 account=physics |
| Qos | scontrol update job=651070 qos=standby |

Setting Up Dependent Jobs

* If a job depends upon the completion of one or more other jobs, you can specify this several ways:

|  |  |
| --- | --- |
| **Where** | **Slurm** |
| Command line | sbatch -d 52628 jobscript  sbatch -d after:52628:52629:52630 jobscript  sbatch -d afterany:52628:52629:52630 jobscript |
| Job script (for this job) | #SBATCH -d 52628  #SBATCH -d after:52628:52629:52630  #SBATCH -d afterany:52628:52629:52630 |
| Job script (for next job) | # $SLURM\_JOBID is current job's id  # Submit next job dependent on this job  sbatch jobscript -d $SLURM\_JOBID |

* Checking/verifying job dependency - use the checkjob *jobid* command.
* % checkjob 123456

sreport

* Reports usage information for a cluster, bank, individual, date range, and more.
* Example: show usage by user (in hours) for the alliance bank on the cluster cab between the dates shown.
* % sreport -t hours cluster AccountUtilizationByUser accounts=alliance cluster=cab start=2/1/12 end=3/1/1

Output Files

Defaults

* The batch output file is named slurm-*jobid*.out
* stdout and stderr are combined into the same batch output file.
* Will be written to the directory where you issued the sbatch or msub command.
* The name of a job has no effect on the name of the output file.
* If an output file with the same name already exists, new output will append to it.

Assigning Unique Output File Names

* Use the -o and -e options to uniquely name your output files - either on the command line or within your job script.
* Use %j to include the jobid, and %N to include the node name in the output file.
* Examples:

|  |
| --- |
| #SBATCH -o /g/g11/joeuser/myjob.out  #SBATCH -o myjob.out.%j  #SBATCH -o myjob.out.%N  #SBATCH -o myjob.out.%j.%N |
| sbatch -e /g/g11/joeuser/myjob.err jobscript  sbatch -o $HOME/proj12/myjob.out jobscript  sbatch -o myjob.out.%j jobscript |

Caveats:

* #SBATCH / #MSUB tokens in a job script do not interpret the ~ (tilde) character or $VARIABLE in file names. These will be interpreted however, by the command line sbatch and msub commands.
* Erroneous file paths are not checked or reported upon.

When Your Job Will Start

* One of the most frequently asked questions is "When will my job start?".
* Because job scheduling is dynamic, and can change at any moment, picking an exact time is often impossible.
* There are a couple ways that you can get an estimate on when your job will start, based on the current situation.
* Sometimes the squeue --start command can be used to get an estimate for job start times (and sometimes it can't). For example:
* % squeue --start
* You can also view the position of your job in the queue relative to other jobs. Either of the commands below will give you a list of idle jobs sorted by priority - highest priority is at the top of the list.
* sprio -l | sort -r -k 4,4
* For example (some output deleted to fit screen):
* % sprio -l | sort -r -k 4,4

Determining When Your Job's Time is About to Expire

* Determining when your job's time is about to expire is useful for cleaning up, writing checkpoint files or other data, and exiting gracefully.
* One common way to accomplish this is to request the batch system to send your job a signal shortly before its time expires.
* Another common way is to query/poll the batch system to determine how much time remains.

Running in Standby Mode

* Jobs can have a Quality of Service (QOS) of:
  + normal - usual case; default
  + exempt - overrides normal policies, limits; requires authorization
  + expedite - highest priority; requires authorization
  + standby - lowest priority
* Standby designates a quality of service (QOS) credential that permits jobs to be preempted/terminated if their resources are needed by non-standby jobs.
* Typically employed by users who wish to take advantage of available cycles on a machine but need to yield to other users with higher priority work. For example:
  + User A has low (or no) priority on a cluster but wishes to take advantage of free cycles.
  + User B has higher priority for work on the cluster
  + User A submits jobs to the cluster with a QOS of standby
  + User A jobs will run to completion if User B doesn't submit jobs with a non-standby (normal) QOS

|  |  |
| --- | --- |
| Sprio | Priority information |
| Sacct  sacct -a | Lists current jobs and their associated accounts |
| scontrol show job  scontrol show job *jobid* | Detailed job information |
| scontrol show node  scontrol show node *nodename* | Detailed node configuration information |
| scontrol show partition  scontrol show partition *partitionname* | Detailed queue configuration information |
| scontrol show config | Detailed Slurm configuration information |
| scontrol show version | Display Slurm version |
| sshare -l | Displays shares, usage and fairshare information |

* + If User B submits jobs that need the nodes being used by User A jobs, then
* Job terminations are immediate and without a warning/signal being sent.
* Running in standby QOS can be set upon job submission or after the job is queued, but before it begins to run:

|  |  |
| --- | --- |
| Job script | #SBATCH --qos=standby |
| Command line (when submitted) | sbatch --qos=standby |
| Command line (queued job) | scontrol update job=*jobid* qos=standby |

Displaying Configuration and Accounting Information

What's Available?

* Several commands can be used to display system configuration and user accounting information.
* Most of this information is for system managers, though it can prove useful to users as well.
* Note that some commands may not be available for all users and/or may be reserved for system managers.
* The table below summarizes some of the more common/useful commands.
* See the relevant man pages for details:
  + scontrol
  + sacct
  + sprio
  + sshare

Parallel Jobs and the srun Command

Srun Command

* The Slurm srun command is required to launch *parallel* jobs - both batch and interactive.
* It should also be used to launch *serial* jobs in the pdebug and other interactive queues.
* Syntax:

[srun](https://hpc.llnl.gov/sites/default/files/srun.txt)   [option list]   [executable]   [args]

Note that srun options must precede your executable.

* Interactive use example, from the login node command line. Specifies 2 nodes (-N), 72 tasks (-n) and the interactive pdebug partition (-p):
* % srun -N2 -n72 -ppdebug myexe
* Batch use example requesting 16 nodes and 576 tasks (assumes nodes have 36 cores):

|  |
| --- |
| #!/bin/tcsh  #SBATCH -N 16  #SBATCH -t 2:00:00  #SBATCH -p pbatch  # Run info and srun job launch  cd /p/lustre1/joeuser/par\_solve  srun -n576 a.out  echo 'Done' |
| % sbatch myjobscript |

* Primary differences between batch and interactive usage:

|  |  |  |
| --- | --- | --- |
| Where used: | From login node command line | In batch script |
| Scheduling: | If there are available interactive nodes, job will run immediately. Otherwise, it will queue up (fifo) and wait until there are enough free nodes to run it. | The batch scheduler handles when to run your job regardless of the number of nodes available. |

* More Examples:

|  |  |
| --- | --- |
| srun -n64 -ppdebug my\_app | 64 process job run interactively in pdebug partition |
| srun -N64 -n512 my\_threaded\_app | 512 process job using 64 nodes. Assumes pbatch partition. |
| srun -N4 -n16 -c4 my\_threaded\_app | 4 node, 16 process job with 4 cores (threads) per process. Assumes pbatch partition. |
| srun -N8 my\_app | 8 node job with a default value of one task per node (8 tasks). Assumes pbatch partition. |
| srun -n128 -o my\_app.out my\_app | 128 process job that redirects stdout to file my\_app.out. Assumes pbatch partition. |
| srun -n32 -ppdebug -i my.inp my\_app | 32 process interactive job; each process accepts input from a file called my.inp instead of stdin |

Task Distribution and Binding for Batch Jobs:

* The default is for the scheduler to distribute tasks as evenly as possible across the allocated nodes.
* Examples: if 4 nodes (with 16 cores each) are requested by a batch job using:
* #SBATCH -N 4

#MSUB -l nodes=4

then the behavior of srun -N and -n flags will be as follows:

* Additionally, tasks are bound to specific cores to promote better cache utilization.
* Threads associated with a task are likewise bound to the same cores.

srun Options

* srun is a powerful command with @100 options affecting a wide range of job parameters.
* For example:
  + Accounting
  + Number and placement of processes/threads
  + Process/thread binding
  + Job resource requirements; dependencies
  + Mail notification options
  + Input, output options
  + Time limits
  + Checkpoint, restart options
  + and much more....
* Some srun options may be set via @60 Slurm environment variables. For example, SLURM\_NNODES behaves like the -N option.

Parallel Output

* Please use a parallel file system for parallel I/O.
* It's a good idea to launch parallel jobs from a parallel file system even if they aren't doing much I/O. A core dump on a parallel job can hang a non-parallel file system easily. Core dumps are written to the directory where you launched your job.
* It is not uncommon for an entire file system to "hang" because a user inadvertently directs output from a large parallel job to an NFS mounted file system.
* NFS file systems include your home directory and /usr/workspace directories.

Running Multiple Jobs from a Single Job Script

Concept:

* It is possible to run more than one job from a single job script.
* Combining multiple jobs into a single job script means there is only one wait in the queue for the entire job group.

Sequential:

* If one job is dependent upon the completion of a previous job, then this method can be used.
* When you submit your job script, be sure to specify enough wall clock time to cover all the included jobs.
* Individual jobs can vary in the number of nodes used, provided none of them exceed the number of nodes allocated to your encompassing job script.
* Example below. Assumes 36 cores per node: 16 nodes \* 36 cores = 576 tasks max.

|  |
| --- |
| #!/bin/tcsh  #SBATCH -N 16  #SBATCH -t 12:00:00  srun -n576 myjob1  srun -n576 myjob2  srun -N16 -n288 myjob3  srun -N12 -n432 myjob4 |

Simultaneous

* This method can be used if there are no dependencies between jobs.
* When you submit your job script, be sure to specify enough nodes to cover all of the included jobs.
* You can vary the number of nodes used by individual jobs as long as the aggregate number of nodes doesn't exceed the number of nodes allocated to your encompassing job script.
* Important to remember:
  + Put each individual job "in the background" using an ampersand - otherwise they will run sequentially.
  + Include a wait statement to ensure the job script doesn't terminate prematurely.
  + With your srun commands, be sure to explicitly specify how many nodes each job requires - or else the scheduler will think each job has access to all nodes, with possible complications.
* Example 1: every job uses the same number of nodes/tasks.  
  Assumes 36 cores per node: 16 nodes \* 36 cores = 576 tasks max.

|  |
| --- |
| #!/bin/tcsh  #SBATCH -N 16  #SBATCH -t 12:00:00  srun -N4 -n144 myjob1 &  srun -N4 -n144 myjob2 &  srun -N4 -n144 myjob3 &  srun -N4 -n144 myjob4 &  wait |

* Example 2: jobs differ in the number of nodes/tasks used.  
  Assumes 36 cores per node: 16 nodes \* 36 cores = 576 tasks max.

|  |
| --- |
| #!/bin/tcsh  #SBATCH -N 16  #SBATCH -t 12:00:00  srun -N4 -n144 myjob1 &  srun -N2 -n72 myjob2 &  srun -N8 -n8 myjob3 &  srun -N2 -n16 myjob4 &  wait |

Batch Commands Summary

|  |  |
| --- | --- |
| [sacct -j](https://hpc.llnl.gov/sites/default/files/sacct.txt) | Display information about a running job, including multiple job steps |
| [sbatch](https://hpc.llnl.gov/banks-jobs/running-jobs/slurm#Submit) | Submit a job script to the batch system. Many options. |
| [scancel](https://hpc.llnl.gov/banks-jobs/running-jobs/slurm#Cancel) | Cancel a running or queued job |
| [scancel --signal](https://hpc.llnl.gov/banks-jobs/running-jobs/slurm#TimeExpired) | Signal a running job |
| [scontrol hold](https://hpc.llnl.gov/banks-jobs/running-jobs/slurm#Hold/Release) | Place a queued job on user hold |
| [scontrol release](https://hpc.llnl.gov/banks-jobs/running-jobs/slurm#Hold/Release) | Release a user held job |
| [scontrol show job](https://hpc.llnl.gov/sites/default/files/scontrol.txt) | Display detailed job information |
| [scontrol show partition](https://hpc.llnl.gov/sites/default/files/scontrol.txt) | Display detailed queue information |
| [scontrol update](https://hpc.llnl.gov/banks-jobs/running-jobs/slurm#ChangingParameters) | Change a job's parameters |
| [sinfo](https://hpc.llnl.gov/banks-jobs/running-jobs/slurm#sinfo) | Display a concise summary of queues and running jobs |
| [sprio](https://hpc.llnl.gov/banks-jobs/running-jobs/slurm#sprio) | Display a list of queued jobs, their priority, and the primary factors used to calculate job priority |
| [squeue](https://hpc.llnl.gov/banks-jobs/running-jobs/slurm#squeue) | Display running jobs |
| [sreport](https://hpc.llnl.gov/banks-jobs/running-jobs/slurm#BanksUsage) | Report usage information for a cluster, bank, individual, date range, and more |
| [srun](https://hpc.llnl.gov/banks-jobs/running-jobs/slurm#ParallelJobs) | Launch a parallel job from within a job script or interactively |
| [sshare](https://hpc.llnl.gov/sites/default/files/sshare.txt) | Display bank/account allocations, usage statistics, and fair-share information |
| [sview](https://hpc.llnl.gov/banks-jobs/running-jobs/slurm#sview) | Graphically display a map of jobs, nodes they are running on, and additional detailed job information |

* For convenience, the table below summarizes several useful batch system commands discussed in this tutorial.

References and More Information

* Slurm information from SchedMD: <http://www.schedmd.com/>
* Job Examples from Livermore Computing: HPVC at LLNL: <https://hpc.llnl.gov/>
* HPC Definitions from Iowa State University: <https://www.hpc.iastate.edu/>
* Slurm Documentation Cheat sheet from: <https://slurm.schedmd.com/rosetta.pdf>
* Task Examples from University of Copenhagen: <https://hpc.ku.dk/documentation/>