Quick Start User Guide

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. Second, it provides a framework for starting, executing, and monitoring work (normally a parallel job) on the set of allocated nodes. Finally, it arbitrates contention for resources by managing a queue of pending work.

Architecture §

As depicted in Figure 1, Slurm consists of a **slurmd** daemon running on each compute node and a central **slurmctld** daemon running on a management node (with optional fail-over twin). The **slurmd** daemons provide fault-tolerant hierarchical communications. The user commands include: **sacct**, **sacctmgr**, **salloc**, **sattach**, **sbatch**, **sbcast**, **scancel**, **scontrol**, **scrontab**, **sdiag**, **sh5util**, **sinfo**, **sprio**, **squeue**, **sreport**, **srun**, **sshare**, **sstat**, **strigger** and **sview**. All of the commands can run anywhere in the cluster.

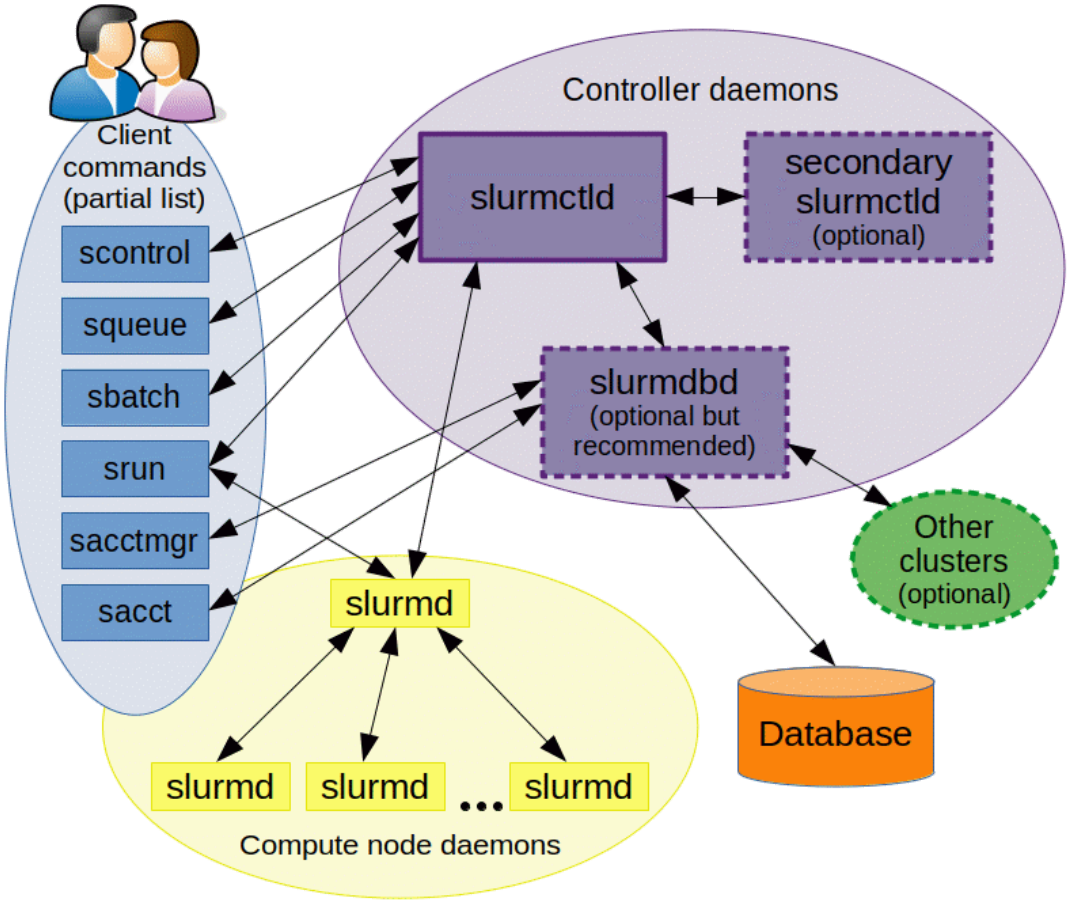

Figure 1. Slurm components

The entities managed by these Slurm daemons, shown in Figure 2, include **nodes**, the compute resource in Slurm, **partitions**, which group nodes into logical (possibly overlapping) sets, **jobs**, or allocations of resources assigned to a user for a specified amount of time, and **job steps**, which are sets of (possibly parallel) tasks within a job. The partitions can be considered job queues, each of which has an assortment of constraints such as job size limit, job time limit, users permitted to use it, etc. Priority-ordered jobs are allocated nodes within a partition until the resources (nodes, processors, memory, etc.) within that partition are exhausted. Once a job is assigned a set of nodes, the user is able to initiate parallel work in the form of job steps

in any configuration within the allocation. For instance, a single job step may be started that utilizes all nodes allocated to the job, or several job steps may independently use a portion of the allocation.

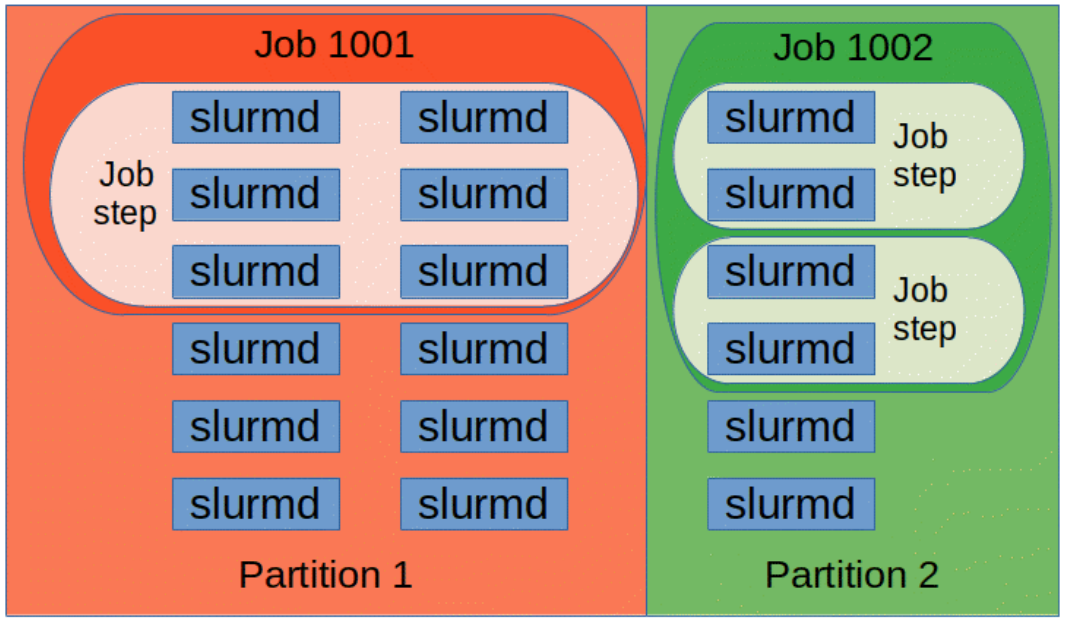

Figure 2. Slurm entities

Commands §

Man pages exist for all Slurm daemons, commands, and API functions. The command option `--help` also provides a brief summary of options. Note that the command options are all case sensitive.

`sacct` is used to report job or job step accounting information about active or completed jobs.

`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.

`sattach` is used to attach standard input, output, and error plus signal capabilities to a currently running job or job step. One can attach to and detach from jobs multiple times.

`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.

`sbcast` is used to transfer a file from local disk to local disk on the nodes allocated to a job. This can be used to effectively use diskless compute nodes or provide improved performance relative to a shared file system.

`scancel` is used to cancel a pending or running job or job step. It can also be used to send an arbitrary signal to all processes associated with a running job or job step.

`scontrol` is the administrative tool used to view and/or modify Slurm state. Note that many `scontrol` commands can only be executed as user root.

`sinfo` reports the state of partitions and nodes managed by Slurm. It has a wide variety of filtering, sorting, and formatting options.

`sprio` is used to display a detailed view of the components affecting a job's priority.

`squeue` reports the state of jobs or job steps. It has a wide variety of filtering, sorting, and formatting options. By default, it reports the running jobs in priority order and then the pending jobs in priority order.

`srun` is used to submit a job for execution or initiate job steps in real time. `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 (so much memory, disk space, certain required features, etc.). A job can contain multiple job steps executing sequentially or in parallel on independent or shared resources within the job's node allocation.

`sshare` displays detailed information about fairshare usage on the cluster. Note that this is only viable when using the priority/multifactor plugin.

`sstat` is used to get information about the resources utilized by a running job or job step.

`strigger` is used to set, get or view event triggers. Event triggers include things such as nodes going down or jobs approaching their time limit.

`sview` is a graphical user interface to get and update state information for jobs, partitions, and nodes managed by Slurm.

Examples §

First we determine what partitions exist on the system, what nodes they include, and general system state. This information is provided by the `sinfo` command. In the example below we find there are two partitions: *debug* and *batch*. The * following the name *debug* indicates this is the default partition for submitted jobs. We see that both partitions are in an *UP* state. Some configurations may include partitions for larger jobs that are *DOWN* except on weekends or at night. The information about each partition may be split over more than one line so that nodes in different states can be identified. In this case, the two nodes *adev[1-2]* are *down*. The * following the state *down* indicate the nodes are not responding. Note the use of a concise expression for node name specification with a common prefix *adev* and numeric ranges or specific numbers identified. This

format allows for very large clusters to be easily managed. The `sinfo` command has many options to easily let you view the information of interest to you in whatever format you prefer. See the man page for more information.

```
adev0: sinfo
PARTITION AVAIL  TIMELIMIT NODES  STATE NODELIST
debug*       up      30:00     2  down* adev[1-2]
debug*       up      30:00     3   idle adev[3-5]
batch        up      30:00     3  down* adev[6,13,15]
batch        up      30:00     3  alloc adev[7-8,14]
batch        up      30:00     4   idle adev[9-12]
```

Next we determine what jobs exist on the system using the `squeue` command. The *ST* field is job state. Two jobs are in a running state (*R* is an abbreviation for *Running*) while one job is in a pending state (*PD* is an abbreviation for *Pending*). The *TIME* field shows how long the jobs have run for using the format *days-hours:minutes:seconds*. The *NODELIST(REASON)* field indicates where the job is running or the reason it is still pending. Typical reasons for pending jobs are *Resources* (waiting for resources to become available) and *Priority* (queued behind a higher priority job). The `squeue` command has many options to easily let you view the information of interest to you in whatever format you prefer. See the man page for more information.

```
adev0: 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)
```

The `scontrol` command can be used to report more detailed information about nodes, partitions, jobs, job steps, and configuration. It can also be used by system administrators to make configuration changes. A couple of examples are shown below. See the man page for more information.

```
adev0: scontrol show partition
PartitionName=debug TotalNodes=5 TotalCPUs=40 RootOnly=NO
   Default=YES OverSubscribe=FORCE:4 PriorityTier=1 State=UP
   MaxTime=00:30:00 Hidden=NO
   MinNodes=1 MaxNodes=26 DisableRootJobs=NO AllowGroups=ALL
   Nodes=adev[1-5] NodeIndices=0-4

PartitionName=batch TotalNodes=10 TotalCPUs=80 RootOnly=NO
   Default=NO OverSubscribe=FORCE:4 PriorityTier=1 State=UP
   MaxTime=16:00:00 Hidden=NO
   MinNodes=1 MaxNodes=26 DisableRootJobs=NO AllowGroups=ALL
   Nodes=adev[6-15] NodeIndices=5-14


adev0: scontrol show node adev1
NodeName=adev1 State=DOWN* CPUs=8 AllocCPUs=0
   RealMemory=4000 TmpDisk=0
   Sockets=2 Cores=4 Threads=1 Weight=1 Features=intel
   Reason=Not responding [slurm@06/02-14:01:24]

65648     batch  math  phil PD  0:00     6 (Resources)
adev0: scontrol show job
JobId=65672 UserId=phil(5136) GroupId=phil(5136)
   Name=math
   Priority=4294901603 Partition=batch BatchFlag=1
   AllocNode:Sid=adev0:16726 TimeLimit=00:10:00 ExitCode=0:0
```

```
   StartTime=06/02-15:27:11 EndTime=06/02-15:37:11
   JobState=PENDING NodeList=(null) NodeListIndices=
   NumCPUs=24 ReqNodes=1 ReqS:C:T=1-65535:1-65535:1-65535
   OverSubscribe=1 Contiguous=0 CPUs/task=0 Licenses=(null)
   MinCPUs=1 MinSockets=1 MinCores=1 MinThreads=1
   MinMemory=0 MinTmpDisk=0 Features=(null)
   Dependency=(null) Account=(null) Requeue=1
   Reason=None Network=(null)
   ReqNodeList=(null) ReqNodeListIndices=
   ExcNodeList=(null) ExcNodeListIndices=
   SubmitTime=06/02-15:27:11 SuspendTime=None PreSusTime=0
   Command=/home/phil/math
   WorkDir=/home/phil
```

It is possible to create a resource allocation and launch the tasks for a job step in a single command line using the `srun` command. Depending upon the MPI implementation used, MPI jobs may also be launched in this manner. See the MPI section for more MPI-specific information. In this example we execute `/bin/hostname` on three nodes (*-N3*) and include task numbers on the output (*-l*). The default partition will be used. One task per node will be used by default. Note that the `srun` command has many options available to control what resource are allocated and how tasks are distributed across those resources.

```
adev0: srun -N3 -l /bin/hostname
0: adev3
1: adev4
2: adev5
```

This variation on the previous example executes `/bin/hostname` in four tasks (*-n4*). One processor per task will be used by default (note that we don't specify a node count).

```
adev0: srun -n4 -l /bin/hostname
0: adev3
1: adev3
2: adev3
3: adev3
```

One common mode of operation is to submit a script for later execution. In this example the script name is *my.script* and we explicitly use the nodes adev9 and adev10 (*-w "adev[9-10]"*, note the use of a node range expression). We also explicitly state that the subsequent job steps will spawn four tasks each, which will ensure that our allocation contains at least four processors (one processor per task to be launched). The output will appear in the file my.stdout ("-o my.stdout"). This script contains a timelimit for the job embedded within itself. Other options can be supplied as desired by using a prefix of "#SBATCH" followed by the option at the beginning of the script (before any commands to be executed in the script). Options supplied on the command line would override any options specified within the script. Note that my.script contains the command `/bin/hostname` that executed on the first node in the allocation (where the script runs) plus two job steps initiated using the `srun` command and executed sequentially.

```
adev0: cat my.script
#!/bin/sh
#SBATCH --time=1
/bin/hostname
srun -l /bin/hostname
srun -l /bin/pwd

adev0: sbatch -n4 -w "adev[9-10]" -o my.stdout my.script
sbatch: Submitted batch job 469

adev0: cat my.stdout
```

```
adev9
0: adev9
1: adev9
2: adev10
3: adev10
0: /home/jette
1: /home/jette
2: /home/jette
3: /home/jette
```

The final mode of operation is to create a resource allocation and spawn job steps within that allocation. The `salloc` command is used to create a resource allocation and typically start a shell within that allocation. One or more job steps would typically be executed within that allocation using the `srun` command to launch the tasks (depending upon the type of MPI being used, the launch mechanism may differ, see MPI details below). Finally the shell created by `salloc` would be terminated using the *exit* command. Slurm does not automatically migrate executable or data files to the nodes allocated to a job. Either the files must exists on local disk or in some global file system (e.g. NFS or Lustre). We provide the tool `sbcast` to transfer files to local storage on allocated nodes using Slurm's hierarchical communications. In this example we use `sbcast` to transfer the executable program *a.out* to */tmp/joe.a.out* on local storage of the allocated nodes. After executing the program, we delete it from local storage

```
tux0: salloc -N1024 bash
$ sbcast a.out /tmp/joe.a.out
Granted job allocation 471
$ srun /tmp/joe.a.out
Result is 3.14159
$ srun rm /tmp/joe.a.out
```

```
$ exit
salloc: Relinquishing job allocation 471
```

In this example, we submit a batch job, get its status, and cancel it.

```
adev0: sbatch test
srun: jobid 473 submitted

adev0: squeue
JOBID PARTITION NAME USER ST TIME  NODES NODELIST(REASON)
  473 batch     test jill R  00:00 1     adev9

adev0: scancel 473

adev0: squeue
JOBID PARTITION NAME USER ST TIME  NODES NODELIST(REASON)
```

Best Practices, Large Job Counts §

Consider putting related work into a single Slurm job with multiple job steps both for performance reasons and ease of management. Each Slurm job can contain a multitude of job steps and the overhead in Slurm for managing job steps is much lower than that of individual jobs.

Job arrays are an efficient mechanism of managing a collection of batch jobs with identical resource requirements. Most Slurm commands can manage job arrays either as individual elements (tasks) or as a single entity (e.g. delete an entire job array in a single command).

MPI §

MPI use depends upon the type of MPI being used. There are three fundamentally different modes of operation used by these various MPI implementations.

1. Slurm directly launches the tasks and performs initialization of communications through the PMI2 or PMIx APIs. (Supported by most modern MPI implementations.)
2. Slurm creates a resource allocation for the job and then mpirun launches tasks using Slurm's infrastructure (older versions of OpenMPI).
3. Slurm creates a resource allocation for the job and then mpirun launches tasks using some mechanism other than Slurm, such as SSH or RSH. These tasks are initiated outside of Slurm's monitoring or control. Slurm's epilog should be configured to purge these tasks when the job's allocation is relinquished. The use of pam_slurm_adopt is also strongly recommended.

Links to instructions for using several varieties of MPI with Slurm are provided below.

- Intel MPI
- MPICH2
- MVAPICH2
- Open MPI

Last modified 29 June 2021

slurm®
workload manager

Job Submission

salloc - Obtain a job allocation.
sbatch - Submit a batch script for later execution.
srun - Obtain a job allocation (as needed) and execute an application.

--array=<indexes> (e.g. "--array=1-10")	Job array specification. (sbatch command only)
--account=<name>	Account to be charged for resources used.
--begin=<time> (e.g. "--begin=18:00:00")	Initiate job after specified time.
--clusters=<name>	Cluster(s) to run the job. (sbatch command only)
--constraint=<features>	Required node features.
--cpus-per-task=<count>	Number of CPUs required per task.
--dependency=<state:jobid>	Defer job until specified jobs reach specified state.
--error=<filename>	File in which to store job error messages.
--exclude=<names>	Specific host names to exclude from job allocation.
--exclusive[=user]	Allocated nodes can not be shared with other jobs/users.
--export=<name[=value]>	Export identified environment variables.
--gres=<name[:count]>	Generic resources required per node.
--input=<name>	File from which to read job input data.
--job-name=<name>	Job name.
--label	Prepend task ID to output. (srun command only)
--licenses=<name[:count]>	License resources required for entire job.
--mem=<MB>	Memory required per node.
--mem-per-cpu=<MB>	Memory required per allocated CPU.
-N<minnodes[-maxnodes]>	Node count required for the job.
-n<count>	Number of tasks to be launched.
--nodelist=<names>	Specific host names to include in job allocation.
--output=<name>	File in which to store job output.
--partition=<names>	Partition/queue in which to run the job.
--qos=<name>	Quality Of Service.
--signal=[B:]<num>[@time]	Signal job when approaching time limit.
--time=<time>	Wall clock time limit.
--wrap=<command_string>	Wrap specified command in a simple "sh" shell. (sbatch command only)

Accounting

sacct - Display accounting data.

--allusers	Displays all users jobs.
--accounts=<name>	Displays jobs with specified accounts.
--endtime=<time>	End of reporting period.
--format=<spec>	Format output.
--name=<jobname>	Display jobs that have any of these name(s).
--partition=<names>	Comma separated list of partitions to select jobs and job steps from.
--state=<state_list>	Display jobs with specified states.
--starttime=<time>	Start of reporting period.

SchedMD
Slurm Support and Development

sacctmgr - View and modify account information.
Options:

--immediate	Commit changes immediately.
--parseable	Output delimited by '|'

Commands:

add <ENTITY> <SPECS> create <ENTITY> <SPECS>	Add an entity. Identical to the **create** command.
delete <*ENTITY*> where <*SPECS*>	Delete the specified entities.
list <ENTITY> [<SPECS>]	Display information about the specific entity.
modify <*ENTITY*> where <*SPECS*> set <*SPECS*>	Modify an entity.

Entities:

account	Account associated with job.
cluster	*ClusterName* parameter in the *slurm.conf*.
qos	Quality of Service.
user	User name in system.

Job Management

sbcast - Transfer file to a job's compute nodes.

sbcast [options] SOURCE DESTINATION

--force	Replace previously existing file.
--preserve	Preserve modification times, access times, and access permissions.

scancel - Signal jobs, job arrays, and/or job steps.

--account=<name>	Operate only on jobs charging the specified account.
--name=<name>	Operate only on jobs with specified name.
--partition=<names>	Operate only on jobs in the specified partition/queue.
--qos=<name>	Operate only on jobs using the specified quality of service.

--reservation=<name>	Operate only on jobs using the specified reservation.
--state=<names>	Operate only on jobs in the specified state.
--user=<name>	Operate only on jobs from the specified user.
--nodelist=<names>	Operate only on jobs using the specified compute nodes.

squeue - View information about jobs.

--account=<name>	View only jobs with specified accounts.
--clusters=<name>	View jobs on specified clusters.
--format=<spec> (e.g. "--format=%i %j")	Output format to display. Specify fields, size, order, etc.
--jobs<job_id_list>	Comma separated list of job IDs to display.
--name=<name>	View only jobs with specified names.
--partition=<names>	View only jobs in specified partitions.
--priority	Sort jobs by priority.
--qos=<name>	View only jobs with specified Qualities Of Service.
--start	Report the expected start time and resources to be allocated for pending jobs in order of increasing start time.
--state=<names>	View only jobs with specified states.
--users=<names>	View only jobs for specified users.

sinfo - View information about nodes and partitions.

--all	Display information about all partitions.
--dead	If set, only report state information for non-responding (dead) nodes.
--format=<spec>	Output format to display.
--iterate=<seconds>	Print the state at specified interval.
--long	Print more detailed information.
--Node	Print information in a node-oriented format.
--partition=<names>	View only specified partitions.
--reservation	Display information about advanced reservations.
-R	Display reasons nodes are in the down, drained, fail or failing state.
--state=<names>	View only nodes specified states.

scontrol - Used view and modify configuration and state.
Also see the **sview** graphical user interface version.

--details	Make show command print more details.
--oneliner	Print information on one line.

Commands:

create *SPECIFICATION*	Create a new partition or .
delete *SPECIFICATION*	Delete the entry with the specified SPECIFICATION
reconfigure	All Slurm daemons will re-read the configuration file.
requeue JOB_LIST	Requeue a running, suspended or completed batch job.
show ENTITY ID	Display the state of the specified entity with the specified identification
update *SPECIFICATION*	Update job, step, node, partition, or reservation configuration per the supplied specification.

Environment Variables

SLURM_ARRAY_JOB_ID	Set to the job ID if part of a job array.
SLURM_ARRAY_TASK_ID	Set to the task ID if part of a job array.
SLURM_CLUSTER_NAME	Name of the cluster executing the job.
SLURM_CPUS_PER_TASK	Number of CPUs requested per task.
SLURM_JOB_ACCOUNT	Account name.
SLURM_JOB_ID	Job ID.
SLURM_JOB_NAME	Job Name.
SLURM_JOB_NODELIST	Names of nodes allocated to job.
SLURM_JOB_NUM_NODES	Number of nodes allocated to job.
SLURM_JOB_PARTITION	Partition/queue running the job.
SLURM_JOB_UID	User ID of the job's owner.
SLURM_JOB_USER	User name of the job's owner.
SLURM_RESTART_COUNT	Number of times job has restarted.
SLURM_PROCID	Task ID (MPI rank).
SLURM_STEP_ID	Job step ID.
SLURM_STEP_NUM_TASKS	Task count (number of MPI ranks).

Daemons

slurmctld	Executes on cluster's "head" node to manage workload.
slurmd	Executes on each compute node to locally manage resources.
slurmdbd	Manages database of resources limits, licenses, and archives accounting records.

SchedMD Slurm Support and Development

slurm® workload manager

http://www.schedmd.com