SLURM User Tutorial

June 2004

Morris Jette (jette@llnl.gov)
Lawrence Livermore National Laboratory

www.llnl.gov/linux/slurm
Disclaimer

This document was prepared as an account of work sponsored by an agency of the United States Government. Neither the United States Government nor the University of California nor any of their employees, makes any warranty, express or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process discloses, or represents that its use would not infringe privately owned rights. References herein to any specific commercial product, process, or service by trade name, trademark, manufacture, or otherwise, does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government or the University of California. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States Government or the University of California, and shall not be used for advertising or product endorsement purposes.

This work was performed under the auspices of the U.S. Department of Energy by the University of California, Lawrence Livermore National Laboratory under Contract No. W-7405-Eng-48.
What is SLURM?

- Arbitrates requests by managing queue of pending work
- Allocates access to computer nodes within a cluster
- Launches parallel jobs and manages them (I/O, signals, limits, etc.)

- NOT a comprehensive cluster administration or monitoring package
- NOT a sophisticated scheduling system
  - An external entity can manage the SLURM queues via plugin (e.g. LCRM or Maui Scheduler)
SLURM in a Nutshell

Users submit work, either to LCRM or directly to SLURM.

SLURM allocates nodes, starts and manages the jobs.

- Node 0
- Node 1
- Node 2
- Node 3
- Node 4
- Node 5
- Node 6
- Node 7
SLURM History

- Jointly developed by LLNL and Linux NetworX
  - Mark Grondona (LLNL), Moe Jette (LLNL), Jay Windley (LNXI)
- Development began in 2002
- Production use at LLNL since 2Q 2003 on Linux clusters with Quadrics switch
- Distributed by Linux NetworX since 1Q 2004
- Total distribution ~400 Linux Clusters world-wide

- Plans for 2004
  - Port to IBM SP (LoadLeveler replacement on ASCI Purple)
  - Scaling enhancements
  - Port to IBM BlueGene/L system
  - Support more flavors of MPI
Why Did LLNL Develop SLURM

> Alternatives have serious limitations
  - Quadrics RMS - Works well, but only with Quadrics network
  - Portable Batch System (PBS) - Portable, but not scalable
  - IBM LoadLeveler - Neither portable nor scalable
  - Load Sharing Facility - Portable and fairly scalable, but very expensive for large clusters

> SLURM is portable, scalable, and fault-tolerant
SLURM Entities

- Nodes
- Partitions
- Jobs
- Job steps
How is SLURM Used at LLNL
LCRM Initiated Jobs

> LCRM decides when and where to initiate a job
> LCRM makes resource allocation in SLURM for the job
> LCRM sets some environment variables for the job (e.g. \textit{SLURM\_JOBID})
  - \textbf{WARNING:} LCRM does not set all of the same environment variables as SLURM (e.g. \textit{SLURM\_NODELIST} is not set)
> LCRM initiates the job script and it runs as any other SLURM job
> LCRM releases the SLURM resource allocation at job termination
How is SLURM Used at LLNL
SLURM Initiated Jobs

> Interactive jobs are submitted directly to SLURM
> Jobs are scheduled on a FIFO (First-In First-Out) basis per partition (backfill scheduling is an option)
> Job scripts can be submitted to SLURM using a “batch” option. These jobs are independent of LCRM
> Only certain partitions can be used interactively to avoid scheduling conflicts with LCRM
SLURM Architecture

> Two daemons
  - slurmd - controller, optional backup
  - slurmd - computer node daemon

> Five user commands
  - scontrol - administration tool, get/set configuration
  - sinfo  - reports general system information
  - squeue - reports job and job step information
  - srun   - submit/initiate job or job step
  - scancel - signal or cancel a job or job step
SLURM Architecture

One daemon per node

Cluster-wide control daemon

slurmd

slurmd

slurmd

slurmd (primary)

slurmd (backup)

sr

sinfo

squeue

scontrol

scancel

User and administrator tools
slurmctld

- Orchestrates SLURM activities across entire cluster (with optional backup)

- Components
  - Job Manager - manages queue of pending jobs
  - Node Manager - node state information
  - Partition Manager - allocates nodes
slurmd

> Daemon executing on each compute node

> Performs actions as directed by slurmdctld and srun

> Components
  - Machine Status
  - Job Status
  - Remote Execution
  - Stream Copy (stdin, stdout, and stderr)
  - Job Control (signal)
**sinfo**

> Displays node and partition information
> Options permit you to filter, sort, and output information in almost any way desired

### Display partition and node state

<table>
<thead>
<tr>
<th>mcrl:</th>
<th>sinfo</th>
</tr>
</thead>
<tbody>
<tr>
<td>PARTITION</td>
<td>AVAIL</td>
</tr>
<tr>
<td>pbatch*</td>
<td>up</td>
</tr>
<tr>
<td>pbatch*</td>
<td>up</td>
</tr>
<tr>
<td>pdebug</td>
<td>up</td>
</tr>
<tr>
<td>pdebug</td>
<td>up</td>
</tr>
</tbody>
</table>

- Asterisk after partition name indicates default partition
- Asterisk after node state indicates it is not responding
- `days:hours:minutes:seconds`
squeue

Displays job and job step information
Options permit you to filter, sort, and output information in almost any way desired

Display running and pending jobs

mcrit: squeue

<table>
<thead>
<tr>
<th>JOBID</th>
<th>PARTITION</th>
<th>NAME</th>
<th>USER</th>
<th>ST</th>
<th>TIME</th>
<th>NODES</th>
<th>NODELIST</th>
</tr>
</thead>
<tbody>
<tr>
<td>16000</td>
<td>pbatch</td>
<td>spring</td>
<td>alice</td>
<td>R</td>
<td>6:46:04</td>
<td>869</td>
<td>mcr[104-586,588-973]</td>
</tr>
<tr>
<td>13601</td>
<td>pbatch</td>
<td>summer</td>
<td>brian</td>
<td>R</td>
<td>4:03:53</td>
<td>165</td>
<td>mcr[979-1143]</td>
</tr>
<tr>
<td>70569</td>
<td>pdebug</td>
<td>fall</td>
<td>cheryl</td>
<td>R</td>
<td>20:07</td>
<td>16</td>
<td>mcr[40-55]</td>
</tr>
<tr>
<td>70573</td>
<td>pdebug</td>
<td>winter</td>
<td>david</td>
<td>R</td>
<td>6:40</td>
<td>16</td>
<td>mcr[64-79]</td>
</tr>
<tr>
<td>70574</td>
<td>pdebug</td>
<td>season</td>
<td>edith</td>
<td>PD</td>
<td>0:00</td>
<td>64</td>
<td></td>
</tr>
</tbody>
</table>

R = Running
PD = Pending
days:hours:minutes:seconds
### squeue - Job Step Example

Display running job steps

**mcrit:** `squeue -s`

<table>
<thead>
<tr>
<th>STEPID</th>
<th>PARTITION</th>
<th>USER</th>
<th>TIME</th>
<th>NODELIST</th>
</tr>
</thead>
<tbody>
<tr>
<td>16000.0</td>
<td>pbatch</td>
<td>alice</td>
<td>6:48:04</td>
<td>mcr[104-586,588-973]</td>
</tr>
<tr>
<td>13601.0</td>
<td>pbatch</td>
<td>brian</td>
<td>4:05:53</td>
<td>mcr[979-1143]</td>
</tr>
<tr>
<td>70569.0</td>
<td>pdebug</td>
<td>cheryl</td>
<td>22:07</td>
<td>mcr[40-55]</td>
</tr>
<tr>
<td>70569.1</td>
<td>pdebug</td>
<td>cheryl</td>
<td>22:06</td>
<td>mcr[40-55]</td>
</tr>
<tr>
<td>70569.2</td>
<td>pdebug</td>
<td>cheryl</td>
<td>22:05</td>
<td>mcr[40-55]</td>
</tr>
<tr>
<td>70569.3</td>
<td>pdebug</td>
<td>cheryl</td>
<td>22:05</td>
<td>mcr[40-55]</td>
</tr>
</tbody>
</table>

Job 70569 has four active steps

<table>
<thead>
<tr>
<th>days:hours:minutes:seconds</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

---

days:hours:minutes:seconds
sr objects

> User tool to initiate jobs and job steps
  - Run jobs interactively
  - Allocate resources
  - Submit batch jobs
  - Attach to currently running job
  - Launch a set of parallel tasks (job step)

> 13 options to specify resource requirements
  - Partition, processor count, node count, minimum memory per node, minimum processor count per node, specific nodes to use or avoid, node can be shared, etc.
**srun - Interactive Example**

Run a job interactively (waits for execution).
Create a four task (and implicitly four processor) resource allocation (job) in the partition *pdebug* and execute the program */bin/hostname* in it labeling the output.
The job’s resource allocation is automatically released upon termination of all tasks.

```
mcrit: srun --ntasks=4 --partition=pdebug --label /bin/hostname
0: mcr56
1: mcr56
2: mcr57
3: mcr57
```

Could be MPI job

**NOTE:** Most SLURM command options have both a long form and a single letter equivalent. The alternate form of the above command is

```
srun -n 4 -p pdebug -l /bin/hostname
```
Create a four task (and implicitly four processor) resource allocation (job) in the partition *pdebug* and spawn a shell to use it.
Launch two job steps (sequentially) to use the job’s allocation.
The job’s resource allocation is automatically released upon termination of the shell.

```
mcri: `srun --ntasks=4 --partition=pdebug --allocate`
mcr56: `hostname`
mcr56
mcr56: `srun --label hostname`
  0: mcr56
  1: mcr56
  2: mcr57
  3: mcr57
mcr56: `srun --label --ntasks=2 hostname`
  0: mcr56
  1: mcr57
mcr56: `exit`
mcri:
```
Job step maintains job’s four tasks
Job step explicitly specifies two tasks
Submit a batch job that executes different job steps on different nodes simultaneously

```
mcri: cat my_script
#!/bin/bash
srun --relative=0 --nodes=1 master &
srun --relative=1 --nodes=3 slave
wait
mcri: srun --nodes=4 --partition=pdebug --batch my_script
srun: jobid 13776 submitted
Later...
mcri: ls
my_script  slurm-13776.err  slurm-13776.out
```

Job’s standard error and output (default file names)
srun - Attach Example

Attach to a existing SLURM job in Read-only mode. Standard error and output from the job are sent to this `srun` in addition to any pre-existing `srun` command associated with the job.

```bash
mcri: srun --attach=13780
Output from job 13780
```

Attach to a existing SLURM job in Read-write mode. The `--join` option permits standard input and signals to be forwarded from this `srun` command to the job.

```bash
mcri: srun --attach=13781 --join
Output from job 13781
exit
```

Input to job from this command
scancel

> Send arbitrary signal to a jobs and/or job step
> By default, sends SIGKILL terminating job
> Filters can be used to specify user, program name, partition, job state, etc.

```
mcrl: scancel 12345
```

Cancel all jobs belonging to user brian with interaction

```
mcrl: scancel --interactive --user=brian
Cancel job id=13601 name=summer partition=pdebug [y/n]? y
Cancel job id=13777 name=NewJob partition=pdebug [y/n]? n
```
scontrol

> Administrative tool to set and get configuration information
> Can be useful to users who want to see full state information without fancy filtering or formatting

mcni: scontrol show partition pdebug
  PartitionName=pdebug TotalNodes=64 TotalCPUs=128 RootOnly=NO
  Default=NO Shared=NO State=UP MaxTime=30
  MinNodes=1 MaxNodes=UNLIMITED AllowGroups=(null)
  Nodes=mcr[40-103] NodeIndecies=0,63,-1

mcni: scontrol show job 70573
  JobId=70573 UserId=david(789) Name=winter JobState=RUNNING
  Priority=4294967295 Partition=pdebug BatchFlag=0
  AllocNode:Sid=mcr39:4277 TimeLimit=30
  StartTime=02/03-14:00:49 EndTime=02/03-14:30:49
  NodeList=mcr[64-79] NodeListIndecies=64,79,-1
  ReqProcs=0 MinNodes=0 Shared=0 Contiguous=0
  MinProcs=0 MinMemory=0 Features=(null) MinTmpDisk=0
  ReqNodeList=(null) ReqNodeListIndecies=-1
Common Questions

> Why isn’t my job running?
  - First-In First-Out scheduling (backfill is configuration option)
  - Jobs get held (priority reset to zero) if they can’t run due to partition constraints (e.g. node count, time limit, etc.)

> Can I use MPICH, LAM/MPI, other version of MPI?
  - Yes, but only Quadrics MPI uses slurmd to initiate tasks
  - Other versions of MPI spawn processes that are not under SLURM management
  - Work to support MPICH and LAM/MPI is planned in 2004
More Common Questions

> How can I control the layout of my tasks?
- srun has a multitude of control mechanisms for this
  - --ntasks=# // task count
  - --nodes=min-max // node count (minimum and maximum)
  - --nodes=# // minimum node count
  - --cpus-per-task=# // count of CPUs required per task
  - --relative=# // start allocation on specified node
  - --nodelist=mcr[10-20] // include (at least) the listed nodes
  - --exclude=mcr34,mcr40 // exclude listed node(s) from allocation
- We plan to add support for explicit task layout file
> How can I establish different executables and/or arguments for each task?
  - We plan to add support for a file to control this
  - For now, you will need to initiate the same executable on each node. This can read the SLURM_PROCID environment variable (same value as MPI rank) and execute a different program using different arguments based upon this
More Information

> All SLURM commands and daemons have *man* pages available

> All SLURM commands have summary of options available
  - “--usage” lists the options
  - “--help” briefly explains all options

> SLURM web site: http://www.llnl.gov/linux/slurm/

> SLURM reference manual:
  http://www.llnl.gov/LCdocs/slurm/

> LLNL users only: lc-hotline@llnl.gov, x24531