SLURM User Tutorial

June 2004

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

www.llnl.gov/linux/slurm

UCRL-WEB-204486

Disclaimer

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*)

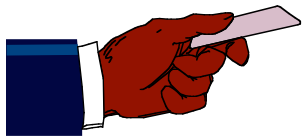
SLURM in a Nutshell

SLURM allocates nodes, starts and manages the jobs

LCRM

Job 1
Job 2
Job 3
Job 4

Node 0
Node 1
Node 2
Node 3
Node 4
Node 5
Node 6
Node 7



Users submit work, either to LCRM or directly to SLURM

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

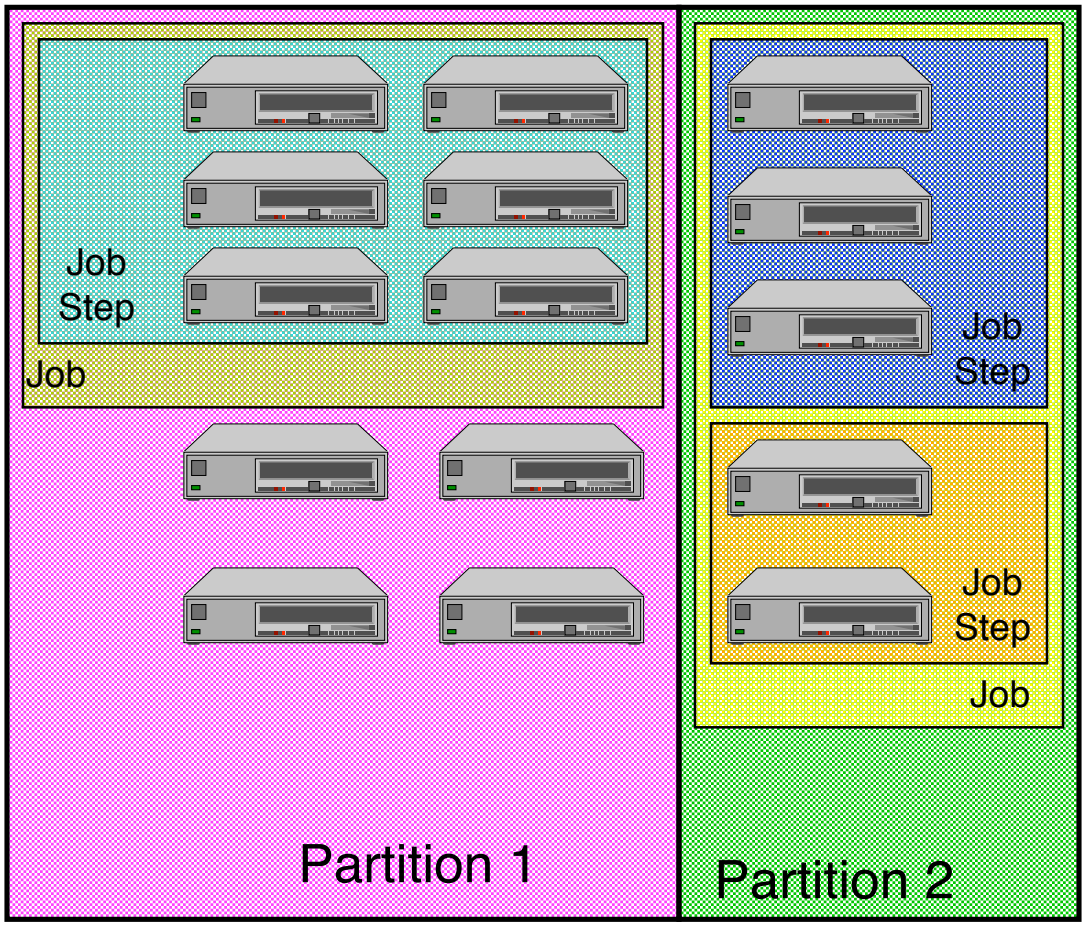
- Plans for 2004
  - Port to IBM SP (LoadLeveler replacement on ASCI Purple)
  - Scaling enhancements
  - Port to IBM BlueGene/L system
  - Support more flavors of MPI

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 not 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. *SLURM_JOBID*)
  - **WARNING:** LCRM does not set all of the same environment variables as SLURM (e.g. 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

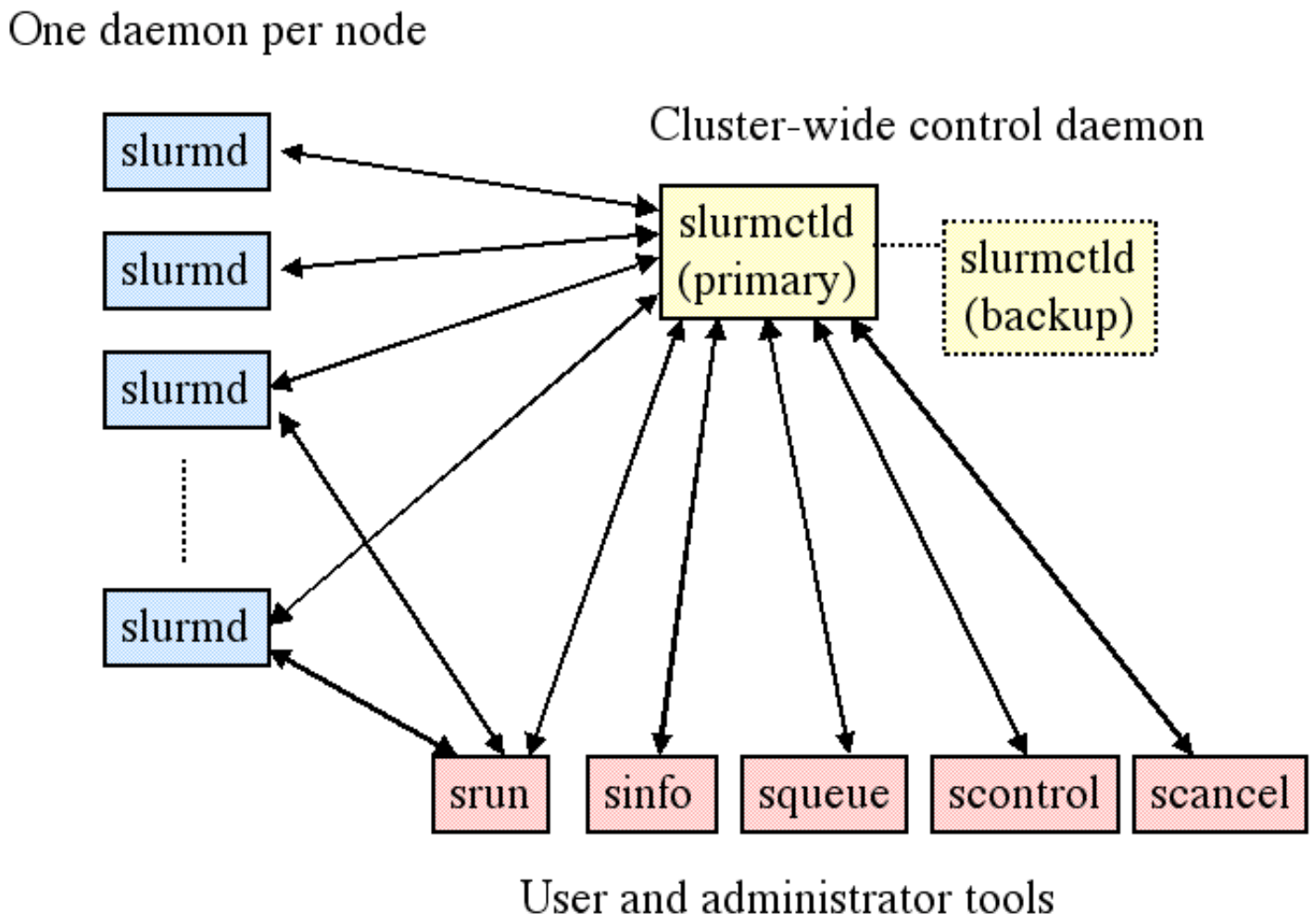
How is SLURM Used at LLNL
SLURM Initiated Jobs

- 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
  - slurmctld - 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



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 slurmctld 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

```
mcri: sinfo
PARTITION AVAIL  TIMELIMIT NODES  STATE NODELIST
pbatch*      up  infinite     1  down* mcr587
pbatch*      up  infinite  1034  alloc mcr[104-586,588-973,979-1143]
pbatch*      up  infinite    13   idle mcr[974-978,1144-1151]
pdebug       up     30:00    32  alloc mcr[40-55,64-79]
pdebug       up     30:00    32   idle mcr[56-63,80-103]
```

Asterisk after partition name indicates default partition

days:hours:minutes:seconds

Asterisk after node state indicates it is not responding

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

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

R = Running
PD = Pending

days:hours:minutes:seconds

squeue - Job Step Example

Display running job steps

```
mcri: squeue -s
STEPID  PARTITION     USER      TIME  NODELIST
16000.0     pbatch    alice  6:48:04  mcr[104-586,588-973]
13601.0     pbatch    brian  4:05:53  mcr[979-1143]
70569.0     pdebug   cheryl    22:07  mcr[40-55]
70569.1     pdebug   cheryl    22:06  mcr[40-55]
70569.2     pdebug   cheryl    22:05  mcr[40-55]
70569.3     pdebug   cheryl    22:05  mcr[40-55]
```

Job 70569 has four active steps

days:hours:minutes:seconds

srun

- 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.

```
mcri: 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 -n4 -p pdebug -l /bin/hostname
```

srun - Allocation Example

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:
```

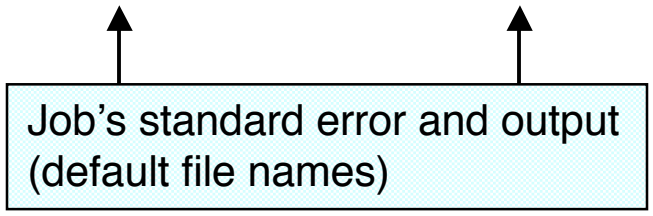
Job step maintains job's four tasks

Job step explicitly specifies two tasks

srun - Batch Example

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
```



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.

```
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.

```
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.

Cancel job id 12345

```
mcri: scancel 12345
```

Cancel all jobs belonging to user *brian* with interaction

```
mcri: 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

```
mcri: 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

mcri: scontrol show job 70573
JobId=70573 UserId=david(789) Name=winter JobState=RUNNING
   Priority=4294895192 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

More Common Questions

- 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