Job Scheduling Using SLURM

Morris Jette
jette@schedmd.com

SchedMD LLC
http://www.schedmd.com
Goals

- Learn the basics of SLURM's architecture, daemons and commands
- Learn how to use a basic set of commands
- Learn how to build, configure and install SLURM with a simple configuration
- This is only an introduction, but it should provide you a good start
Agenda

- Role of a resource manager and job scheduler
- SLURM design and architecture
- SLURM commands
- SLURM build and configuration
- Demonstration of SLURM configuration and use
Role of a Resource Manager

- The “glue” for a parallel computer to execute parallel jobs
- It should make a parallel computer as almost easy to use as a PC

On a PC. Execute program “a.out”:
```
a.out
```

On a cluster. Execute 8 copies of “a.out”:
```
srun -n8 a.out
```

- MPI would typically be used to manage communications within the parallel program
Role of a Resource Manager

- Allocate resources within a cluster
  - Nodes (typically a unique IP address)
    - NUMA boards
      - Sockets
        - Cores
          - Hyperthreads
      - Memory
    - Interconnect/switch resources
    - Generic resources (e.g. GPUs)
  - Licenses
- Launch and otherwise manage jobs

Can require extensive knowledge about the hardware and system software (e.g. to alter network routing or manage switch window)
Role of a Job Scheduler

- When there is more work than resources, the job scheduler manages queue(s) of work
  - Supports complex scheduling algorithms
    - Optimized for network topology, fair share scheduling, advanced reservations, preemption, gang scheduling (time-slicing jobs), etc.
  - Supports resource limits (by queue, user, group, etc.)

- Many batch systems provide both resource management and job scheduling within a single product (e.g. LSF) while others use distinct products (e.g. Torque resource manager and Moab job scheduler)
SLURM and ALPS Functionality

• **SLURM**
  - Prioritizes queue(s) of work and enforces limits
  - Decides when and where to start jobs
  - Terminates job when appropriate
  - Accounts for jobs
  - No daemons on compute nodes

• **ALPS**
  - Allocates and releases resources for jobs
  - Launches tasks
  - Monitors node health
  - Manages node state
  - Has daemons on compute nodes
Agenda

- Role of a resource manager and job scheduler
- SLURM design and architecture
- SLURM commands
- SLURM build and configuration
- Demonstration of SLURM configuration and use
What is SLURM?

- **Simple Linux Utility for Resource Management**
- Development started in 2002 as a simple resource manager for Linux clusters
- Has evolved into a capable job scheduler through use of optional plugins
- About 500,000 lines of C code today.
- Supports AIX, Linux, Solaris, other Unix variants
- Used on many of the world's largest computers
SLURM Design Goals

- Small and simple (depends upon configuration, used by Intel for their “cluster on a chip”)
- Highly scalable (managing 1.6 million core IBM BlueGene/Q, tested to 33 million cores using emulation)
- Fast (throughput up to 600 jobs per second and up to 1000 job submissions per second)
- Open source (GPL v2, active world-wide development)
- System administrator friendly
- Secure
- Fault-tolerant (no single point of failure)
- Portable
SLURM Portability

- No kernel modifications
- C language
- *Autoconf* configuration engine adapts to environment
- Provides skeleton of functionality with general-purpose plugin mechanism. System administrator can extensively customize installation using a building-block approach
- Various system-specific plugins available and more under development (e.g. *select/bluegene*, *select/cray*)
Plugins

- Dynamically linked objects loaded at run time based upon configuration file and/or user options
- 70 plugins of 17 different varieties currently available
  - Accounting storage: MySQL, PostgreSQL, text file
  - Network topology: 3D-torus, tree
  - MPI: OpenMPI, MPICH1, MVAPICH, MPICH2, etc.

<table>
<thead>
<tr>
<th>SLURM Kernel</th>
<th>Authentication Plugin</th>
<th>MPI Plugin</th>
<th>Checkpoint Plugin</th>
<th>Topology Plugin</th>
<th>Accounting Storage Plugin</th>
</tr>
</thead>
<tbody>
<tr>
<td>Munge</td>
<td>mvapich</td>
<td>BLCR</td>
<td>Tree</td>
<td>MySQL</td>
<td></td>
</tr>
<tr>
<td>Munge</td>
<td>mvapich</td>
<td></td>
<td>Tree</td>
<td>MySQL</td>
<td></td>
</tr>
</tbody>
</table>
Plugin Design

- Plugins typically loaded when the daemon or command starts and persist indefinitely
- Provide a level of indirection to a configurable underlying function

Write job completion accounting record

- Write it to text file
  - OR
  - Write it to slurmdbd daemon
    - OR
    - Write it to MySQL database
      - OR
      - Ignore request
Plugin Development

• Interfaces are all documented for custom development (e.g. GreenSpot for optimized use of green energy sources)
• Most plugins have several examples available
• Some plugins have a LUA script interface
Job Submit Plugin

- Call for each job submission or modification
- Can be used to set default values or enforce limits using functionality outside of SLURM proper

Two functions need to be supplied:

```c
int job_submit(struct job_descriptor *job_desc, uint32_t submit_uid);
int job_modify(struct job_descriptor *job_desc, struct job_record *job_ptr, uint32_t submit_uid);
```
Resource Selection Plugin

- Whole node allocations (select/linear)
- Socket/Core/Hyperthread allocation (select/cons_res)
- IBM BlueGene - Interfaces to IBM's BlueGene APIs
- Cray – Interfaces with Cray's APIs (BASIL) then uses SLURM's whole node allocation plugin

```
SLURM Kernel

Whole Node selection plugin       OR       Cray selection plugin

Whole Node selection plugin       ALPS
```
SLURM's `select/cray` plugin

- **select/cray**
  - **select/linear**
  - **libalps (SLURM module)**
  - **BASIL**
    - **ALPS (MySQL database)**
  - **libemulate (SLURM module)**

**OR**

- Emulates BASIL and ALPS for development and testing
- Emulate any size Cray on any test system
- To use, build SLURM with `configure` option of `--with-alps-emulation`
SLURM Entities

- Jobs: Resource allocation requests
- Job steps: Set of (typically parallel) tasks
- Partitions: Job queues with limits and access controls
- Nodes
  - NUMA boards
    - Sockets
    - Cores
      - Hyperthreads
    - Memory
    - Generic Resources (e.g. GPUs)
SLURM Entities Example

- Users submit jobs to a partition (queue)

<table>
<thead>
<tr>
<th>Partition “debug”</th>
</tr>
</thead>
<tbody>
<tr>
<td>Job 1</td>
</tr>
<tr>
<td>Job 2</td>
</tr>
<tr>
<td>Job 3</td>
</tr>
</tbody>
</table>
SLURM Entities Example

- Jobs are allocated resources
SLURM Entities Example

- Jobs spawn steps, which are allocated resources from within the job's allocation

```bash
#!/bin/bash
srun -n4 --exclusive a.out &
srun -n2 --exclusive a.out &
wait
```
Node State Information

- NUMA boards, Sockets, Cores, Threads
- CPUs (can treat each core or each thread as a CPU for scheduling purposes)
- Memory size
- Temporary disk space
- Features (arbitrary string, e.g. OS version)
- Weight (scheduling priority, can favor least capable node that satisfies job requirement)
- Boot time
- CPU Load
- State (e.g. drain, down, etc.)
  - Reason, time and user ID (e.g. “Bad PDU [operator@12:40:10T12/20/2011]”)
SLURM/Cray Node Naming

- **NodeName** – “nid######”
- **NodeAddr** – X, Y and Z coordinate (e.g. “123”)
- **NodeHostName** – “c#-#c#s#n#” where the “#” represents the cabinet, row, cage, blade or slot, and node

- slurm.conf only needs to specify NodeName
  - NodeAddr and NodeHostName read from ALPS database (MySQL)
SLURM/Cray Node State

- Node state managed by ALPS
- SLURM reads node state from ALPS
- Node state changes by SLURM tools (e.g. `scontrol`) disabled on Cray systems
SLURM/Cray Node Ordering

- SLURM uses a best-fit algorithm to allocate resources, but honors the node ordering policy set by ALPS.
- Node ordering is configurable using `ALPS_NIDORDER` in `/etc/sysconfig/alps`.
- Configuration shown by `apstat -svv`.
- Ordering shown by `apstat -no`.
Node States

Down → Idle → Allocated → Completing (a flag)

Drained (Down or Idle with Drain flag set) → Completing (a flag)

Draining (Allocated or Completing with Drain flag set) → Drained (Down or Idle with Drain flag set)

> scontrol update NodeName=X State=[drain | resume] Reason=X
Queue/Partition State Information

- Associated with specific set of nodes
  - Nodes can be in more than one partition
- Job size and time limits (e.g. small size and time limits for some partition and larger limits for others)
- Access control list (by Linux group)
- Preemption rules
- State information (e.g. drain)
- Over-subscription and gang scheduling rules
Job State Information

- ID (a number)
- Name
- Time limit (minimum and/or maximum)
- Size specification (minimum and/or maximum; nodes, CPUs, sockets, cores, and/or threads)
- Specific node names to include or exclude in allocation
- Node features required in allocation
- Dependency
- Account name
- Quality Of Service (QOS)
- State (Pending, Running, Suspended, Cancelled, Failed, etc.)
Job States

Submission → Pending → Configuring (node booting) → Resizing → Running → Suspended → Completing

Completed (zero exit code) → Preempted
Failed (non-zero exit code) → TimeOut (time limit reached) → NodeFail
Step State Information

- ID (a number): <jobid>.<stepid>
- Name
- Time limit (maximum)
- Size specification (minimum and/or maximum; nodes, CPUs, sockets, cores, and/or threads)
- Specific node names to include or exclude in allocation
- Node features required in allocation
Cluster Architecture

Typical Linux Cluster

Slurmctld (master) -> Slurmctld (backup) -> Slurmd daemons on compute nodes

Slurmdbd (master) -> Slurmdbd (backup) -> MySQL

Accounting and configuration records

Slurm daemons on compute nodes
(Note hierarchical communications with configurable fanout)
SLURM Architecture for Cray

**Slurmctld**
(SLURM controller daemon)
(primary or backup)
Coordinates all activities

**Slurmd**
(SLURM job daemons)
(Active on one or more service nodes)
Runs batch script

- **BASIL**
- **ALPS**

**MySQL**

Direct communications to MySQL gets each node's X, Y and Z coordinates
Enterprise Architecture

SLURM user tools

SLURM (cluster 1)

Jobs & status

Accounting data

SLURM (cluster N)

User and bank Limits and preferences

SLurmDBD

MySQL

User account and limit info

Job and step accounting info

SLURM administration tools
Daemons

- **slurmctld** – Central controller (typically one per cluster)
  - Optional backup with automatic fail over
  - Monitors state of resources
  - Manages job queues
  - Allocates resources
- **slurmd** – Compute node daemon (typically one per compute node, on Cray and IBM Bluegene systems, one or more on front-end nodes)
  - Launches and manages tasks
  - Small and very light-weight (low memory and CPU use)
  - Quiescent after launch except for optional accounting
  - Supports hierarchical communications with configurable fanout
- **slurmdbd** – database daemon (typically one per enterprise)
  - Collects accounting information
  - Uploads configuration information (limits, fair-share, etc.)
  - Optional backup with automatic fail over
Daemon Command Line Options

- **-c**  Clear previous state, purge all job, step, partition state
- **-D**  Run in the foreground, logs are written to stdout
- **-v**  Verbose error messages, each “v” roughly doubles volume of messages

**Typical debug mode command lines**

```
> slurmctld -Dcvvvv
> slurmd -Dcvvv
```
Compute Node Configuration

- Execute `slurmd` with `-C` option to print the node's current configuration and exit
- This information can be used as input to the SLURM configuration file

```
> slurmd -C
NodeName=jette CPUs=6 Sockets=1 CoresPerSocket=6 ThreadsPerCore=1
  RealMemory=8000 TmpDisk=930837
```
slurmstepd
(SLURM daemon to shepherd a job step)

- One slurmstepd per job step
- Spawned by slurmd at job step initiation
- Manages a job step and processes its I/O
- Only persists while the job step is active
Each daemon writes its own logfile

- See configuration parameters `SlurmctldLogFile` and `SlurmdLogFile` in `slurm.conf` and `LogFile` in `slurmdbd.conf`
  - `SlurmdLogFile` name can include “%n” which is replaced by the node's name (e.g. “`SlurmdLogFile=slurmd.%n.log`”)
- `DebugLevel` configuration parameters control how verbose the logging is
- Important messages go to `syslog`
- The daemon's log file can be much more verbose
Logs

• Detailed logging can also be generated on 17 specific sub-systems using the `DebugFlags` configuration parameter
  • Backfill, CPU_Bind, Gang, Gres, Priority, Reservation, Steps, Triggers, etc.

• `DebugFlags` and `DebugLevel` can be reset in real time using the `scontrol` command
  • `scontrol setdebugflags +priority` (Add `DebugFlag` of “priority”)
  • `scontrol setdebug debug`
Agenda

- Role of a resource manager and job scheduler
- SLURM design and architecture
- SLURM commands
- SLURM build and configuration
- Demonstration of SLURM configuration and use
Commands: General Information

- Man pages available for all commands, daemons and configuration files
- --help option prints brief description of all options
- --usage option prints a list of the options
- Commands can be run on any node in the cluster
- Any failure results in a non-zero exit code
- APIs make new tool development easy
  - Man pages available for all APIs

SchedMD LLC
http://www.schedmd.com
Commands: General Information

- Almost all options have two formats
  - A single letter option (e.g. “-p debug” for partition debug)
  - A verbose option (e.g. “--partition=debug”)
- Time formats are days-hours:minutes:seconds
- Almost all commands support verbose logging with “-v” option, use more v’s for more verbosity, -vvvv
- Many environment variables can be used to establish site-specific and/or user-specific defaults
  - For example “SQUEUE_STATES=all” for the squeue command to display jobs in any state, including COMPLETED or CANCELLED
**SLURM Commands: Job/step Allocation**

- **sbatch** – Submit script for later execution (batch mode)
- **salloc** – Create job allocation and start a shell to use it (interactive mode)
- **srun** – Create a job allocation (if needed) and launch a job step (typically an MPI job)
- **sattach** – Connect stdin/out/err for an existing job or job step
Submit sequence of three batch jobs

\[
> \text{sbatch --ntasks=1 --time=10 pre_process.bash} \\
\text{Submitted batch job 45001}
\]

\[
> \text{sbatch --ntasks=128 --time=60 --depend=45001 do_work.bash} \\
\text{Submitted batch job 45002}
\]

\[
> \text{sbatch --ntasks=1 --time=30 --depend=45002 post_process.bash} \\
\text{Submitted batch job 45003}
\]
**srun Command**

- SLURM has *srun* command for Cray systems that is a wrapper for both *salloc* (to allocate resources as needed) and *aprun* (to launch tasks).
- Options are translated to the extent possible.
- Build SLURM with *configure* option –*with-srun2aprun* to build wrapper.
  - Otherwise *srun* command advises use of *aprun* and exits.

```
srun (wrapper)
```

```
  
  alloc
  
  aprun
```
**srunc Options**

- If no allocation exists, the `srunc` options are translated directly to `salloc` options to create a job allocation
  - Many `srunc` option only apply when a job allocation is created
- After an allocation is created
  - Most common options are translated from `srunc` to `aprun` (task count, node count, time limit, file names, support for multiple executables, etc.)
  - Some `srunc` options lack `aprun` equivalent and vice-versa
  - `srunc`'s "–alps=" option can pass any other options to `aprun`
  - SLURM environment variables are not currently set
  - There are fundamental differences in I/O
    - For example, ALPS does not support per-rank I/O streams
Job and Step Allocation Examples

Create allocation for 2 tasks then launch “hostname” on the allocation, label output with the task ID
> srun –ntasks=2 –label hostname
0: tux123
1: tux123

As above, but allocate the job two whole nodes
> srun –nnodes=2 --exclusive –label hostname
0: tux123
1: tux124
Create allocation for 4 tasks and 10 minutes for bash shell, then launch some tasks

> salloc –ntasks=4 –time=10 bash
salloc: Granted job allocation 45000

> env | grep SLURM
SLURM_JOBID=45000
SLURM_NPROCS=4
SLURM_JOB_NODELIST=tux[123-124]

... 

> hostname
tux_login

> srun –label hostname
0: tux123
1: tux123
2: tux124
3: tux124

> exit (terminate bash shell)
Different Executables by Task ID

- Different programs may be launched by task ID with different arguments
- Use `--multi-prog` option and specify configuration file instead of executable program
- Configuration file lists task IDs, executable programs, and arguments ("%t" mapped to task ID and "%o" mapped to offset within task ID range)

```
> cat master.conf
#TaskID  Program               Arguments
 0       /usr/me/master
 1-4     /usr/me/slave      --rank=%o

> srun –ntasks=5 –multi-prog master.conf
```
srun Caveats for Cray

• Some SLURM functionality has no ALPS equivalent
  • Independent I/O by task
  • Output labeled by task ID
• Some ALPS options have no SLURM equivalent
  • *srun* wrapper has –*alps* option to pass arbitrary arguments to *aprun*
• Some options are similar, but impossible to directly translate
  • Task binding syntax
  • Per-task vs. per-CPU limits
srun Caveats for Cray

- Some SLURM environment variables are not set
  - Many need to be set on a per-node or per-task basis, so ALPS would need to do this
MPI Support

• Many different MPI implementations are supported:
  • MPICH1, MPICH2, MVAPICH, OpenMPI, etc.
• Many use `srun` to launch the tasks directly
• Some use “`mpirun`” or another tool within an existing SLURM allocation (they reference SLURM environment variables to determine what resources are allocated to the job)
• Details are online:
Cray Job Launch Sequence

1. User submits script

slurmctld
(SLURM controller daemon)
(primary or backup)
Coordinates all activities

slurmd
(SLURM job daemons)
Runs batch script

BASIL
ALPS
Compute Nodes
1. User submits script
2. Slurmctld creates ALPS reservation

---

Cray Job Launch Sequence

- **slurmctld** (SLURM controller daemon (primary or backup))
  Coordinates all activities

- **slurmd** (SLURM job daemons)
  Runs batch script

---

BASIL

ALPS

Compute Nodes
Cray Job Launch Sequence

1. User submits script
2. Slurmctld creates ALPS reservation
3. Slurmctld sends script to slurmd
4. Slurmd claims reservation for specific session ID and launches interpreter for script

```
#!/bin/bash
srun a.out
```

- **slurmctld** (SLURM controller daemon) (primary or backup) Coordinates all activities
- **slurmd** (SLURM job daemons) Runs batch script

**Compute Nodes**

- **BASIL**
- **ALPS**

```
#!/bin/bash
srun a.out
```
1. User submits script

2. Slurmctld creates ALPS reservation

3. Slurmctld sends script to slurmd

4. Slurmd claims reservation for specific session ID and launches interpreter for script

5. aprun (optionally using the srun wrapper) launches tasks on compute nodes

---

**Slurmctld**

- (SLURM controller daemon)
- (primary or backup)
- Coordinates all activities

**Slurmd**

- (SLURM job daemons)
- Runs batch script

**BASIL**

**ALPS**

**Compute Nodes**

```
#!/bin/bash
srun a.out
```

```
slurmctld
```

```
slurmd
```

```
#!/bin/bash
```

```
•aprun a.out
```

---

SchedMD LLC
http://www.schedmd.com
Linux Job Launch Sequence

1a. srun sends job allocation request to slurmctld
1b. slurmctld grant allocation and returns details
2a. srun sends step create request to slurmctld
2b. slurmctld responds with step credential
**Linux Job Launch Sequence**

1. **srun (interactive)** sends job allocation request to **slurmctld**
2. **slurmctld** grants allocation and returns details
3. **srun** opens sockets for I/O
4. **srun** forwards credential with task info to **slurmd**

- **1a.** srun sends job allocation request to slurmdctld
- **1b.** slurmdctld grants allocation and returns details
- **2a.** srun sends step create request to slurmdctld
- **2b.** slurmdctld responds with step credential
- **3.** srun opens sockets for I/O
- **4.** srun forwards credential with task info to slurmd
Linux Job Launch Sequence

1a. srun sends job allocation request to slurmd
1b. slurmd grant allocation and returns details
2a. srun sends step create request to slurmd
2b. slurmd responds with step credential
3. srun opens sockets for I/O
4. srun forwards credential with task info to slurmd
5. slurmd forward request as needed (per fanout)
6. slurmd forks/execs slurmstepd
Linux Job Launch Sequence

1a. `srun` sends job allocation request to `slurmctld`
1b. `slurmctld` grant allocation and returns details
2a. `srun` sends step create request to `slurmctld`
2b. `slurmctld` responds with step credential
3. `srun` opens sockets for I/O
4. `srun` forwards credential with task info to `slurmd`
5. `slurmd` forward request as needed (per fanout)
6. `slurmd` forks/execs `slurmstepd`
7. `slurmstepd` connects I/O to run & launches tasks
Linux Job Launch Sequence

1. srun sends job allocation request to slurmctld
2. slurmctld grants allocation and returns details
3. srun sends step create request to slurmctld
4. slurmctld responds with step credential
5. srun opens sockets for I/O
6. srun forwards credential with task info to slurmd
7. slurmd forward request as needed (per fanout)
8. slurmd forks/execs slurmstepd
9. slurmstepd connects I/O to run & launches tasks
10. on task termination, slurmstepd notifies srun
11. srun notifies slurmctld of job termination
12. slurmctld verifies termination of all processes via slurmd and releases resources for next job
SLURM Commands: System Information

- **sinfo** – Report system status (nodes, queues, etc.)
- **squeue** – Report job and job step status
- **smap** – Report system, job or step status with topology (curses-based GUI), less functionality than sview
- **sview** – Report and/or update system, job, step, partition or reservation status with topology (GTK-based GUI)
- **scontrol** – Administrator tool to view and/or update system, job, step, partition or reservation status
sinfo Command

- Reports status of nodes or partitions
  - Partition-oriented format is the default
- Almost complete control over filtering, sorting and output format is available

```bash
> sinfo --Node (report status in node-oriented form)
NODELIST  NODES  PARTITION  STATE
  tux[000-099] 100  batch  idle
  tux[100-127] 28  debug  idle

> sinfo -p debug (report status of nodes in partition “debug”)
PARTITION AVAIL TIMELIMIT NODES NODELIST
d debug  up  60:00  28 tux[100-127]

> sinfo -i60  (report status every 60 seconds)
```
squeue Command

- Reports status of jobs and/or steps in slurmd daemon's records (recent job's only, older information available in accounting records only)
- Almost complete control over filtering, sorting and output format is available

> squeue -u alec -t all (report jobs for user “alec” in any state)

<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(REASON)</th>
</tr>
</thead>
<tbody>
<tr>
<td>45124</td>
<td>debug</td>
<td>a.out</td>
<td>alec</td>
<td>CD</td>
<td>0:12</td>
<td>1</td>
<td>tux123</td>
</tr>
</tbody>
</table>

> squeue -s -p debug (report steps in partition “debug”)

<table>
<thead>
<tr>
<th>STEPID</th>
<th>PARTITION</th>
<th>NAME</th>
<th>USER</th>
<th>TIME</th>
<th>NODELIST</th>
</tr>
</thead>
<tbody>
<tr>
<td>45144.0</td>
<td>debug</td>
<td>a.out</td>
<td>moe</td>
<td>12:18</td>
<td>tux[100-115]</td>
</tr>
</tbody>
</table>

> squeue -i60 (report currently active jobs every 60 seconds)
sview on Cray (3-D torus)
sview on Cray (3-D torus)

NodeAddr contains X,Y,Z-coordinate
smap on Cray (3-D torus)
Caveats

- SLURM GUIs (actually the *curses* and GTK libraries they use) have limited scalability
  - Scales to a few thousand nodes
  - Currently each position displayed represents a unique X, Y, Z-coordinate
  - If multiple nodes share an X, Y, Z-coordinate, the information for only one node is displayed
  - We found this better than displaying each node independently and providing confusing topology information, but could change this if desired
scontrol Command

- Designed for system administrator use
- Shows all available fields, but no filtering, sorting or formatting options
- Many fields can be modified

```
> scontrol show partition
PartitionName=debug
   AllocNodes=ALL AllowGroups=ALL Default=YES
   DefaultTime=NONE DisableRootJobs=NO GraceTime=0 Hidden=NO
   MaxNodes=UNLIMITED MaxTime=UNLIMITED MinNodes=1
   Nodes=tux[000-031]
   Priority=1 RootOnly=NO Shared=NO PreemptMode=OFF State=UP
   TotalCPUs=64 TotalNodes=32 DefMemPerNode=512
   MaxMemPerNode=1024
> scontrol update PartitionName=debug MaxTime=60
```
SLURM Commands: Accounting

• **sacct** – Report accounting information by individual job and job step

• **sstat** – Report accounting information about currently running jobs and job steps (more detailed than sacct)

• **sreport** – Report resources usage by cluster, partition, user, account, etc.
sacct Command

- Reports accounting information for jobs and steps
- Many filtering and output format options
- Uses accounting file or database (which may not exist depending upon SLURM configuration)

> sacct -u joseph (report accounting information for user “joseph”)
> sacct -p debug (report accounting information for partition “debug”)

SchedMD LLC
http://www.schedmd.com
SLURM Commands: Scheduling

- **sacctmgr** – Database management tool
  - Add/delete clusters, accounts, users, etc.
  - Get/set resource limits, fair-share allocations, etc.
- **sprio** – View factors comprising a job's priority
- **sshare** – View current hierarchical fair-share information
- **sdiag** – View statistics about scheduling module operations (execution time, queue length, etc.) New in SLURM version 2.4
Database Use

- Accounting information written to a database plus
  - Information pushed out live to scheduler daemons
  - Quality of Service (QOS) definitions
  - Fair-share resource allocations
  - Many limits (max job count, max job size, etc)
  - Based upon hierarchical banks
    - Limits by user AND by banks

“All I can say is wow – this is the most flexible, useful scheduling tool I’ve ever run across.”
Adam Todorski, Rensselaer Polytechnic Institute
Hierarchical bank example

- Root: 100%
  - Division A: 33.3%
    - Group Alpha: 50%
      - Pam: 20%
      - Ted: 30%
  - Division B: 33.3%
    - Group Beta: 30%
      - Pat: 25%
  - Division C: 33.3%
    - Group Gamma: 20%
      - Bob: 25%
SLURM Commands: Other

- **scancel** – Signal/cancel jobs or job steps
- **sbcast** – Transfer file to a compute nodes allocated to a job (uses hierarchical communications)
- **srun_cr** – Wrapper to srun for support of Berkeley checkpoint/restart
- **strigger** – Event trigger management tools
scancel Command

- Cancel a running or pending job or step
- Can send arbitrary signal to all processes on all nodes associated with a job or step
- Has filtering options (state, user, partition, etc.)
- Has interactive (verify) mode

> scancel 45001.1 (cancel job step 45001.1)
> scancel 45002    (cancel job 45002)
> scancel –user=alec –state=pending (cancel all pending jobs from user “alec”)
sbcast Command

- Copy a file to local disk on allocated nodes
  - Execute command after a resource allocation is made
- Data transferred using hierarchical slurmd daemons communications
- May be faster than shared file system

> salloc -N100 bash
salloc: Granted job allocation 45201
> sbcast --force my_data /tmp/moe/my_data (overwrite old files)
> srun a.out
> exit (terminate spawned "bash" shell)
strigger command

- SLURM can run an arbitrary script when certain events occur
  - Node goes DOWN
  - Daemon stops or restarts
  - Job close to time limit
  - Many others
- strigger command can be used to create, destroy or list event triggers
Agenda

- Role of a resource manager and job scheduler
- SLURM design and architecture
- SLURM commands
- SLURM build and configuration
- Demonstration of SLURM configuration and use
Select Distribution

- Download a tar-ball
  - http://www.schedmd.com/#repos
- New minor release about every 9 months
  - 2.2.x December 2010
  - 2.3.x September 2011
  - 2.4.x June 2012
- Micro releases with bug fixes about once each month
- Very latest code base available from github
  - “git clone git://github.com/SchedMD/slurm.git”
Install Needed Infrastructure

- expat-2.0.xxx
- libexpat-devel-2.0.xxx
- Munge (authentication)
  - RPMs: munge, munge-devel, munge-libs
  - Need to create some directories and configure keys
- MySQL (read ALPS database, job accounting)
  - Cray-MySQL-devel-enterprise-5.0.64 (from Cray iso)
Build and Install RPMs

- Build and install the relevant RPMs
  - rpmbuild -ta slurm-2.4.1.tar.bz2
  - rpm –install <the rpm files>
- NOTE: Some RPMs are infrastructure dependent
  - slurm-auth-authd*rpm  Authd authentication
  - slurm-bluegene*rpm    IBM BlueGene systems only
SLURM RPMs

- Slurm Commands, daemons
- Slurm-dev Header files and libraries
- Slurm-perlapi Perl API interface to SLURM
- Slurm-auth-none Trivial authentication plugin (avoid)
- Slurm-auth-authd Authd authentication (avoid)
- Slurm-auth-munge Munge authentication (recommended)
- Slurm-bluegene IBM BlueGene support
- Slurm-slurmdb-direct Direct database access tool (avoid)
- Slurm-slurmdbd Database daemon and plugins
- Slurm-sql Database plugins
- Slurm-plugins Most plugins
SLURM RPMs

- Slurm-torque  Torque/PBS command wrappers
- Slurm-srun2aprun  srun command wrapper for aprun
- Slurm-sjobexit  Job exit code management tool
- Slurm-aix  AIX systems only (avoid)
- Slurm-percs  IBM PERCS systems only (avoid)
- Slurm-proctrack-sgi-job  SGI job container plugin
- Slurm-lua  LUA bindings
- Slurm-sjstat  job stats tool (avoid)
- Slurm-pam-slurm  PAM module for restricting node access (avoid)
- Slurm-blcr  Berkely Lab Checkpoint Restart plugin (avoid)
Build and Install RPMs

- Build and install the relevant RPMs
  - `rpmbuild -ta slurm-2.4.1.tar.bz2`
  - `rpm --install <the rpm files>`
- NOTE: Some RPMs are infrastructure dependent
  - `slurm-auth-authd*rpm` Authd authentication
  - `slurm-bluegene*rpm` IBM BlueGene systems only
  - `slurm-switch-elan*rpm` Quadrics Elan switch
Build and Install without RPMs

- Uncompress and unpack the tar-ball (or get git repository)
- Create “build” directory and enter it
- Execute “configure <options>”
  - Typical options:
    - --enable-debug perform additional debugging
    - --prefix=<dir> installation directory
    - --sysconfdir=<dir> configuration file directory
    - --with-munge=<dir> Munge installation directory
    - --with-srun2aprun Build srun wrapper to aprun command
- Run “make” to build the executable
- Run “make install” to install the executable, header, and library files
- Individual commands can be edited, built and installed quickly
> cd /tmp/slurm
(get the slurm tar-ball)
> bunzip2 slurm-2.4.1.tar.bz2
> tar -xf slurm-2.4.1.tar
> ls
slurm-2.4.1
> mkdir build
> cd build
> /tmp/slurm/slurm-2.4.1/configure --enable-debug --prefix=/tmp/slurm/install
(identifies header files, libraries, etc.)
> make -j
(builds the SLURM executable and library files)
> make install
(install the files in /tmp/slurm/install)
Configuration

- Configuration `slurm.conf` file required on all compute and service nodes
- Most configuration parameters have usable defaults
- At least the nodes in the cluster and grouping into a partition is required
  - Open with web browser
  - Set values as appropriate
  - Select the “Submit” button on the bottom
  - Save resulting file in “sysconfdir” location
- See “man slurm.conf” for more help
SLURM Version 2.4 Configuration Tool

This form can be used to create a SLURM configuration file with you controlling many of the important configuration parameters.

This tool supports SLURM version 2.4 only. Configuration files for other versions of SLURM should be built using the tool distributed with it in doc/html/configurator.html. Some parameters will be set to default values, but you can manually edit the resulting slurm.conf as desired for greater flexibility. See man slurm.conf for more details about the configuration parameters.

Note the while SLURM daemons create log files and other files as needed, it treats the lack of parent directories as a fatal error. This prevents the daemons from running if critical file systems are not mounted and will minimize the risk of cold-starting (starting without preserving jobs).

Note that this configuration file must be installed on all nodes in your cluster.

After you have filled in the fields of interest, use the "Submit" button on the bottom of the page to build the slurm.conf file. It will appear on your web browser. Save the file in text format as slurm.conf for use by SLURM.

For more information about SLURM, see http://www.schedmd.com/slurmdocs/slurm.html

Control Machines

Define the hostname of the computer on which the SLURM controller and optional backup controller will execute. You can also specify addresses of these computers if desired (defaults to their hostnames). The IP addresses can be either numeric IP addresses or names. Hostname values should should not be the fully qualified domain name (e.g. use tux rather than tux.abc.com).

<table>
<thead>
<tr>
<th>linux</th>
<th><strong>ControlMachine</strong>: Master Controller Hostname</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td><strong>ControlAddr</strong>: Master Controller Address (optional)</td>
</tr>
</tbody>
</table>
Example slurm.conf file

# slurm.conf file generated by configurator.html.
# Put this file on all nodes of your cluster.
# See the slurm.conf man page for more information.
#
ControlMachine=linux0
#ControlAddr=
#BackupController=
#BackupAddr=
#
AuthType=auth/munge
CacheGroups=0
#CheckpointType=checkpoint/none
CryptoType=crypto/munge
… CONTENT REMOVED HERE …
#
FrontEndNodes= State=UNKNOWN
NodeName=nid[00000-01234] CPUs=1 State=UNKNOWN
PartitionName=debug Nodes=nid[00000-01234] Default=YES State=UP
Cray GPU Use

- SLURM supports allocation of Generic Resources (GRES)
- Configure `gpu` and `gpu_mem` (if desired) in `slurm.conf` and `gres.conf` files

```bash
# Excerpt of slurm.conf
GresType=gpu,gpu_mem
NodeName=nid[00000-00015] Gres=gpu:1,gpu_mem:2048
```

```bash
sbatch --gres=gpu:1,gpu_mem:1024 my.bash
```
Cray Configuration File

- SLURM has a configuration file for use only on Cray systems: *cray.conf*
- Some of the Cray-specific configuration parameters
  - `apbasil` – path to the apbasil command
  - `SDBdb` – Name of ALPS database (default XTAdmin)
  - `SDBhost` – Name of DB server (default sdb)
  - `SDBpass` – DB password (default load from *my.cnf*)
  - `SDBuser` – DB user (default load from *my.cnf*)
Authentication

- MUNGE is SLURM's default authentication and digital signature mechanism
  - http://code.google.com/p/munge/
- Each node in cluster must be configured with a MUNGE key and have the daemons running
- MUNGE generated credential includes
  - User ID
  - Group ID
  - Time stamp
  - Whatever else it is asked to sign and/or encrypt
    - Names of nodes allocated to a job/step
    - Specific CPUs on each node allocated to a job/step, etc.
Authentication

- If desired, multiple Munge daemons can be configured with different keys
  - One key for use within a cluster
  - A second key for communications between clusters
Test Suite

• SLURM includes an extensive test suite that can be used to validate proper operation
  • Includes over 300 test programs
  • Executes thousands of jobs
  • Executes tens of thousands of steps
• Change directory to “testsuite/expect”
• Create file “globals.local” with installation specific information
• Execute individual tests or run “regression” for all tests
Test Suite

- SLURM includes an extensive test suite that can be used to validate proper operation
  - Includes over 300 test programs
  - Executes thousands of jobs
  - Executes tens of thousands of steps
- Change directory to “testsuite/expect”
- Create file “globals.local” with installation specific information
- Execute individual tests or run “regression” for all tests
Test Suite Example

> cat globals.local
set slurm_dir "~/home/moe/Desktop/SLURM/install.linux"
set build_dir "~/home/moe/Desktop/SLURM/build.linux"
set src_dir "~/home/moe/Desktop/SLURM/slurm.git"
> regression >qa.tux.jun5
Completions: 315
Failures: 1
Time (sec): 3650

• Search output file “qa.tux.jun5” above for “FAILURE” to investigate test failures
System Emulation

- SLURM can easily be configured to emulate various system architectures or system sizes
  - Emulate a Cray or IBM BlueGene/Q on a laptop
    - Underlying database interactions are simulated
  - Emulate 64 cores per node when there are only 4 cores
  - Emulate a 128 node cluster using 4 nodes
    - Run 32 `slurmd` daemons on each compute node
  - Good to test resource allocation logic
  - Not so good to run MPI applications
Agenda

- Role of a resource manager and job scheduler
- SLURM design and architecture
- SLURM commands
- SLURM build and configuration
- Demonstration of SLURM configuration and use
Step By Step Directions

- Current on-line documentation applies to SLURM version 2.4.3+
  - Major updates since previous release for Cray build/installation
  - Extensive description plus logs
  - Question: How best to make RPMs available?

http://www.schedmd.com/slurmdocs/cray.html
Demonstration: Build and Install Munge

- SchedMD to provide Munge RPMs
- ssh boot -l root
  - xtopview
  - rpm -i munge*rpm (installs at /opt/munge)
  - mkdir /opt/munge/key
  - dd if=/dev/urandom bs=1 count=1024 >/opt/munge/key/munge.key (generates psuedo-random key)
  - chmod go-rwx /opt/munge/key/munge.key
  - chown daemon /opt/munge/key/munge.key
  - Edit /etc/init.d/munge, line 37
    - DAEMON_ARGS=""--key-file /opt/munge/key/munge.key""
Configure Munge

- On each login node and SDB where
  - SLURM daemon will execute and/or
  - Users will execute SLURM commands
- mkdir –mode=0711 -vp /var/lib/munge
- mkdir –mode=0700 -vp /var/log/munge
- mkdir –mode=0755 -vp /var/run/munge
- chown daemon /var/lib/munge
- chown daemon /var/log/munge
- chown daemon /var/run/munge
- /etc/init.d/munge start
Validate Munge

- export PATH=/opt/munge/bin:$PATH
- munge -n
- munge -n | unmunge
- munge -n | ssh <other_host> /opt/munge/bin/unmunge
Enable Cray Job Service

- ALPS relies on the Cray job service to generate cluster-wide unique job container IDs (PAGG IDs)
- Make sure that /etc/init.d/job service is enabled (ckconfig) and started
- Enable pam_job.so module from /opt/cray/job/default in /etc/pam.d/common-session (needed for interactive jobs, salloc)
Enable Cray Job Service

- ssh -l root boot
  - xtopview -c login
  - chkconfig job on
  - Edit /etc/pam.d/common-session
    - Uncomment the line with pam_job.so
Demonstration: Configuration

- cd /opt/slurm/etc
- Edit slurm.conf
  - ControlMachine (set to your SDB)
  - ProctrackType (set to “proctrack/sgi_job”)
    - Start service with “/etc/init.d/job”
  - SelectType (set to “select/cray”)
  - FrontEndName (identify where slurmd daemons will run)
  - NodeName (in node specification section, use “nid#####” names)
    - Get node IDs using “apstat -n”
    - Batch nodes only (reset with “xtprocadmin -k m batch -n NODEIDS”)
  - Nodes (in partition specification section)
  - SlurmUser (set to “root”)
Demonstration: Build and Install SLURM

- SchedMD to provide SLURM RPMs
- ssh boot -l root
  - xtopview
  - rpm -i slurm*rpm (RPMs that you want)
  - Install your slurm.conf file in /etc/slurm/slurm.conf
- On nodes used by SLURM daemons, create directories for logs and state files:
  - mkdir -p /ufs/slurm/log
  - mkdir -p /ufs/slurm/spool
  - module load slurm
  - /etc/init.d/surm start
Demonstration: Commands

- export PATH=/opt/slurm/bin:$PATH
- sinfo
- squeue
- srun hostname
- NOTE: Database and accounting not configured
  - This is a minimal configuration
What's Next

- This just covers the basics of SLURM administration and use
- Lots more documentation available online and in the man pages
  - Help available via email: slurm-dev@schedmd.com
  - Bugzilla available at: http://bugs.schedmd.com
  - Problem reports should include
    - SLURM version
    - Configuration details (output of “scontrol show config” or `slurm.conf` file contents)
    - Logs, scripts, etc.