The SLURM Scheduler Design

SLURM User Group Meeting 2012

October 9, 2012

Don Lipari
Agenda

- Purpose of the talk
- Scope of the talk
- What the scheduler must do
- Job submission options
- How it does it
Purpose of the Talk

- To provide a conceptual framework to serve as a reference as you:
  - Read through the documentation
  - Diagnose problems
  - Particularly when trying to discover why a job is not being scheduled.
  - Modify the code
Scope of the Talk

- Explain how SLURM selects the resources needed to run the top priority job
- Explore the code design, SLURM v2.4
- Illuminate the scheduling loop
- Include a discussion of the supporting plugins
- Omit Blue Gene and Cray ports
- Omit the multi-factor priority plugin
- Less important details omitted in the interest of time
- Focus on scheduling activities, not all of the functionality of the SLURM control daemon
Scheduling

- The process of determining what job to run next and on which resources.
- Based on the job request, resources available, and policy limits imposed.
- Starts with job priority.
- Results in a resource allocation over a period of time.
- The slurmctld loops through a set of jobs and finds resources to schedule to those jobs.
Scheduling Jobs

- Job 1
- Job 2
- Job 3

(time)
Allocation vs. Task Placement

- Allocation is the selection of the resources needed for the job
- Each job includes zero or more job steps (srun)
- Each job step is comprised of one to multiple tasks
- Task placement is the process of assigning a subset of the job’s allocated resources (cpus) to each task
What the Scheduler Must Schedule

- Nodes
- Sockets
- Cores
- Hardware Threads (cpus)
- Memory
- Generic resources (e.g., gpus, file systems)
- Licenses
Effectively amounts to an n-dimensional game of Tetris
Scheduling-related Plugins - 1

- sched
  - backfill
  - builtin
  - hold
  - wiki
  - wiki2

- select
  - bluegene
  - cons_res
  - cray
  - linear
  - serial (in v2.5)
Scheduling-related Plugins - 2

- gres
  - gpu
  - nic

- preempt
  - none
  - partition_prio
  - qos

- priority
  - basic
  - multifactor

- topology
  - 3d_torus
  - node_rank
  - none
  - tree
Scheduling Modes

- Node scheduled
  - linear, bluegene, and cray select plugins
- CPU scheduled
  - cons_res (consumable resources) plugin
Job Submit Options

- **sbatch**
  - Submits a job command script for batch scheduling
  - **sbatch** returns immediately with the job ID

- **salloc**
  - Requests an interactive shell
  - **salloc** blocks until the job is scheduled, at which point a shell prompt appears

- **srun**
  - Run an executable as a single job and job step
  - **srun** blocks until the job is scheduled
sbatch/salloc/srun
Resource Allocation Specifications - 1

- Cluster  -M, --clusters (one or many)
- Partition -p, --partition (one or many)
- Node count-N, --nodes (accepts range)
- Node restrictions
  - --exclusive
  - --share
  - --contiguous
  - --geometry
  - -F, --nodefile
  - -w, --nodelist
  - -x, --exclude
  - --switches
sbatch/salloc/srun
Resource Allocation Specifications - 2

- Task count -n, --ntasks
- Task specifications
  --ntasks-per-node
  --ntasks-per-socket
  --ntasks-per-core
  --cpus-per-task
  -O, --overcommit
- Memory (node) --mem
- Memory (cpu) --mem-per-cpu
- Generic resources --gres
- Licenses --licenses
sbatch/salloc/srun Filters
Eliminate Nodes from Consideration

-C, --constraint
--sockets-per-node=<sockets>
--cores-per-socket=<cores>
--threads-per-core=<threads>
--mem
--mem-per-cpu
--mincpus
--tmp=<min disk space>
Further Constraints

- Job dependency  -d, --dependency
- Node reservation  --reservation
Time Dimension

- Duration -t, --time
- Min duration --time-min
- Start after --begin
Task Placement Directives

- --cpus-per-task
- -m, --distribution
- --ntasks-per-node
- --ntasks-per-socket
- --ntasks-per-core
- -O, --overcommit
- --hint
Task Binding

- For info on task binding, see the 2011 SLURM User Group Meeting presentation:
  Resource Management for Multi-Core/Multi-Threaded Usage by Martin Perry
Configuration Options Affecting Scheduling

- **DefMemPerCPU**
  - MaxMemPerCPU

- **DefMemPerNode**
  - MaxMemPerNode

- **FastSchedule**

- **GresTypes**

- **Licenses**

- **MaxTasksPerNode**

- **ResvOverRun**

- **SchedulerTimeSlice**

- **SchedulerType**

- **SchedulerParameters**

- **SelectType**

- **SelectTypeParameters**

- **TopologyPlugin**

- **Node and Partition settings**
slurmctld standard threads

- _slurmctld_background
  actually, the main thread

- _slurmctld_rpc_mgr
  responds to remote procedure calls

- _slurmctld_signal_hand
  responds to signals to transition the system to a different state

- slurmd_state_save
  repeatedly runs to save system state
Examples of Other Threads

- Multifactor Priority Plugin Threads
  _decay_thread
  _cleanup_thread

- Sched Plugin Threads
  builtin_agent
  backfill_agent

- Accounting Storage slurmdbd Plugin Thread
  _set_db_inx_thread
Agents

- Responsible for transmitting a common RPC in parallel across a set of nodes
  - `agent_queue_request()` is the standard call to initiate an agent
- Allows services that send messages to continue without waiting for the delivery of a message.
- These threads live just long enough to deliver (and perhaps confirm delivery of) the message.
Global objects of interest

- node_record_table_ptr - table of node_record structures
- node_record_count - number of entries
- node_hash_table - used to quickly search the node_record_table
- avail_node_bitmap
- job_list - unsorted List of job_record structures
- part_list - List of part_record structures
- Protected by read/write locks defined at the start of _slurmctld_background()
Node Bitmaps

- One bit per node
- Node bitmaps always include all nodes in the cluster
- Part of the secret sauce contributing to the scheduler’s performance
  - bitwise and operations are faster than if clauses
Slurmctld’s main()

1. Read slurm.conf
2. Load Plugins
3. Negotiate control with backup
4. Read state files / create threads
5. Enter _slurmctld_background
Update Node States

Send Updates to Accounting DB

Trigger Events

Schedule

Node Health Check/Ping

Terminate Jobs Exceeding Limits

_sendmctld_background
Generic Scheduling Loop

1. Build job/part queue
2. Pop top priority job
3. Reject non-runnable jobs
4. Find nodes for job
5. Preempt jobs if necessary
6. Start job
7. Update job and nodes states

Loop back to step 1.
Scheduling Call Stack

schedule()
  - Build job queue
  - Pop top priority, least preemptable job

select_nodes()
  - Build node candidate list

_get_req_features()
  - Omit reserved nodes
  - Build preemptable jobs list

_pick_best_nodes()
  - Cycles through each feature
  - Consider where nodes can be shared

select_g_job_test()
  - Selects “best” nodes
  - Returns preemptee list
select_g_job_test()

- Selects the “best” nodes for the job
- Depending on mode, decides whether the job:
  - Can ever run (SELECT_MODE_TEST_ONLY)
  - Can run now (SELECT_MODE_RUN_NOW)
  - Can run later (SELECT_MODE_WILL_RUN) - used to provide start time estimates
- Considers switch_record_table entries when present
- Inputs a list of running jobs that are candidates for preemption
- RUN_NOW and WILL_RUN return
  - Where job can run
  - A list of jobs to preempt
Scheduling-Related Plugins
select Plugin-Specific Behavior

- linear
  - schedules whole nodes and memory
- con_res
  - schedules sockets/cores/threads and memory
- bluegene
- cray
- serial
sched plugin

- backfill
  - Separate thread schedules lower priority jobs when they will not delay the start of the top priority job.

- builtin
  - Implements FIFO scheduling when paired with priority/basic plugin
  - Creates a new thread to calculate the projected start times of pending jobs

- hold
  - Assigns a zero priority to every job submitted

- wiki, wiki2
  - Contains the code necessary to dialog with an external scheduler (Maui / Moab)
sched/builtin plugin

- A separate thread that periodically cycles through each pending job

- First calculation:
  - `select_g_job_test(SELECT_MODE_WILL_RUN)` to determine the job’s start time

- Second calculation:
  - Delays the start time of a subsequent job using the same nodes as a previous job to start after the time limit of the previous job.
Bacfill Scheduling Jobs

Job 1

Job 2

Job 3

Job 4
sched/backfill plugin

- SchedulerType=sched/backfill
- SchedulerParameters
  - bf_interval - minimum backfill cycle period (30 sec default)
  - bf_max_job_user - limits number of backfilled jobs per user per cycle
  - bf_resolution - job start / end time accuracy (60 sec default)
  - bf_window - how many minutes into the future to look (1 day default)
  - max_job_bf - limits number of backfilled jobs per cycle
- An additional thread, separate from the _slurmctld_background thread
- Uses lock_slurmctld() to keep from clobbering the slurmctld’s objects
- Descends the job queue running every job that will complete before the expected start time of the top priority job
- Follows logic similar to the schedule() function
Generic Scheduling Loop (again)

- Build queue of runnable job/parts
- Remove top priority job from queue
- Reject jobs that cannot or should not run
- Find nodes for job
- Create preemption list and preempt jobs if necessary
- Start job
- Update job and node states
topology plugin

- **3d_torus**
  - modifies the node_rank member of each node_record which is used by the slurmctld to order the node_record_table and thereby influence the order of the nodes that are allocated

- **tree**
  - populates the switch_record_table which is used by the linear and cons_res select plugins to select the nodes for each job

- **node_rank, none**
  - stub functions
topology/3d_torus plugin
Implements Hilbert Curve

- Maps 3D coordinates into 1D such that contiguous coordinates in 1D map to 3D coordinates with the shortest communication path between them.
gres plugin

- GresTypes can specify multiple gres plugins
- Gres can be defined for nodes in slurm.conf and does not require a plugin to be created.
- Each node reads its own, optional gres.conf
  - name, count, device file, associated cpus
  - slurmd sends this info to slurmctld at registration
- A job can request gres(s)
- Generic resources are searched and allocated in select plugin’s select_p_job_test() by invoking gres plugin’s gres_plugin_job_test()
preempt plugin

- partition_prio, qos, or none

- Provides functions to assist scheduling
  - job_preempt_check() used in job queue sorting
  - preemption_enabled() used by _pick_best_nodes() to build the bitmap of candidate nodes
  - find_jobs() used by _get_req_features() and backfill scheduler to generate list of candidates for job preemption
  - job_preempt_mode() determines the mode of preemption select_nodes() uses to preempt jobs
Gang Scheduling

- `gs_job_start()` adds jobs to be gang scheduled to a list
- The `_timeslicer_thread` cycles through jobs in this list, resuming or suspending each job.
- Gang scheduling can work in concert with the preempt plugin
### Possible PreemptMode Values

<table>
<thead>
<tr>
<th>PreemptType=preempt/none</th>
<th>PreemptType=preempt/partition_prio</th>
<th>PreemptType=preempt/qos</th>
</tr>
</thead>
<tbody>
<tr>
<td>OFF</td>
<td>OFF</td>
<td>OFF</td>
</tr>
<tr>
<td>OFF,GANG</td>
<td>CANCEL</td>
<td>CANCEL</td>
</tr>
<tr>
<td></td>
<td>CANCEL,GANG</td>
<td>CANCEL,GANG</td>
</tr>
<tr>
<td></td>
<td>CHECKPOINT</td>
<td>CHECKPOINT</td>
</tr>
<tr>
<td></td>
<td>CHECKPOINT,GANG</td>
<td>CHECKPOINT,GANG</td>
</tr>
<tr>
<td></td>
<td>REQUEUE</td>
<td>REQUEUE</td>
</tr>
<tr>
<td></td>
<td>REQUEUE,GANG</td>
<td>REQUEUE,GANG</td>
</tr>
<tr>
<td></td>
<td>SUSPEND,GANG</td>
<td>SUSPEND,GANG</td>
</tr>
</tbody>
</table>
Reservations

- Reserves nodes, and CPUs in the future
- Slurmctld maintains a list of reservations
- The req_node_bitmap of jobs with a reservation is populated with the nodes from that reservation \[\_\text{build\_node\_list()}\]
- Jobs without a reservation are can only select nodes from an avail_node_bitmap that is missing the reserved nodes \[\_\text{get\_req\_features()}\]
Job Reduction and Expansion

- Both operations use `update_job()` to do the work.
- Job reduction done by `excise_node_from_job()`.
- Job expansion requires the submission of a new job with the “expand” dependency, requesting additional nodes.
- Once the expand job is scheduled, `select_p_job_expand()` does the work of transferring the original job’s resources to the expanded job.
Checkpoint / restart

- "scontrol checkpoint vacate" releases nodes to the scheduling pool, making them available to other jobs.
- Restart relies on a new job being scheduled.
job_allocate

- In response to an sbatch invocation (as well as other scenarios)
- Provides will-run data if requested and returns
  - Example: `srun --test-only`
- Creates the job and adds it to the job list
- Provides a path for a submitted job to be immediately scheduled - calls select_nodes()
Immediate Scheduling Path

**job_allocate**
- Confirm job is independent
- Confirm job is top priority

**select_nodes**
- Build node candidate list

**_get_req_features**
- Omit reserved nodes
- Build preemptable jobs list

**_pick_best_nodes**
- Cycles through each feature
- Consider where nodes can be shared

**select_g_job_test**
- Selects “best” nodes
- Returns preemptee list
## (Most Common) Reasons for Job Pending State

<table>
<thead>
<tr>
<th>Reason</th>
<th>Enum</th>
<th>When it gets set</th>
</tr>
</thead>
<tbody>
<tr>
<td>BeginTime</td>
<td>WAIT_TIME</td>
<td>build_job_queue()</td>
</tr>
<tr>
<td>Dependency</td>
<td>WAIT_DEPENDENCY</td>
<td>build_job_queue()</td>
</tr>
<tr>
<td>JobHeldAdmin</td>
<td>WAIT_HELD</td>
<td>Job create / update</td>
</tr>
<tr>
<td>JobHeldUser</td>
<td>WAIT_HELD_USER</td>
<td>Job create / update</td>
</tr>
<tr>
<td>Licenses</td>
<td>WAIT_LICENSES</td>
<td>schedule() while loop</td>
</tr>
<tr>
<td>PartitionTimeLimit</td>
<td>WAIT_PART_TIME_LIMIT</td>
<td>build_job_queue() and select_nodes()</td>
</tr>
<tr>
<td>PartitionNodeLimit</td>
<td>WAIT_PART_NODE_LIMIT</td>
<td>build_job_queue()</td>
</tr>
<tr>
<td>Priority</td>
<td>WAIT_PRIORITY</td>
<td>schedule() while loop</td>
</tr>
<tr>
<td>ReqNodeNotAvail</td>
<td>WAIT_NODE_NOT_AVAIL</td>
<td>select_nodes()</td>
</tr>
<tr>
<td>Reservation</td>
<td>WAIT_RESERVATION</td>
<td>select_nodes()</td>
</tr>
<tr>
<td>Resources</td>
<td>WAIT_RESOURCES</td>
<td>schedule() while loop and select_nodes()</td>
</tr>
</tbody>
</table>
References

- Documentation
  http://www.schedmd.com/slurmdocs/documentation.html

- Publications
  http://www.schedmd.com/slurmdocs/publications.html

- Man Pages
  http://www.schedmd.com/slurmdocs/man_index.html

- Job scheduling with the SLURM resource manager by Michal Novotný