Tuning Slurm Scheduling for Optimal Responsiveness and Utilization

Morris Jette
jette@schedmd.com

SchedMD LLC
http://www.schedmd.com

Outline

- Job priority
- Main scheduling logic
- Backfill scheduling
- Premption
- Gang scheduling
- Network topology
- Diagnostics

Outline

- Job priority
- Main scheduling logic
- Backfill scheduling
- Premption
- Gang scheduling
- Network topology
- Diagnostics

Job Prioritization

- Priority/multifactor plugin assigns priorities to jobs based upon configurable factors

```
JobPriority =
   PriorityWeightAge * AgeFactor +
   PriorityWeightFairshare * FairShareFactor +
   PriorityWeightJobSize * JobSizeFactor +
   PriorityWeightPartition * PartitionFactor +
   PriorityWeightQOS * QosFactor
```

Job Priority is 32-bit integer
Factors all floating point numbers between 0 and 1.0
PriorityWeight values all 32-bit integers

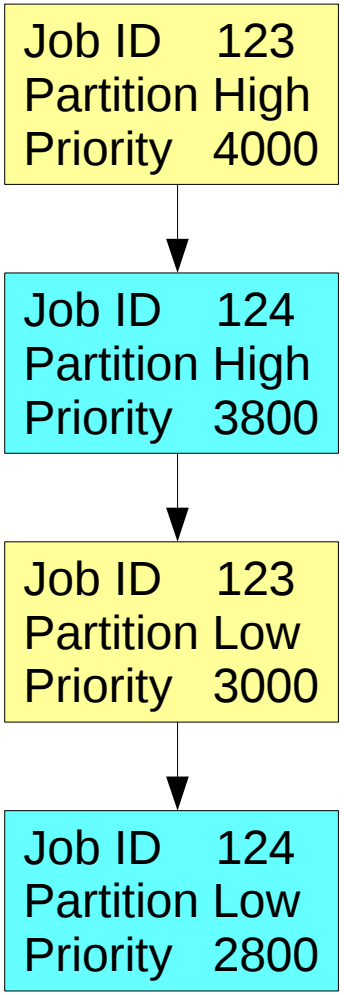
IMPORTANT: Set PriorityWeight values high to generate wide range of job priorities

Job Queue Order
All Jobs Placed in a Single Queue

1. Preemption order (preemptor higher priority than preemptee)
2. Advanced reservation (jobs with an advanced reservation are higher priority than other jobs)
3. Partition priority
4. Job priority
5. Job ID

NOTE: Job priority is only one factor in queue order

Jobs Submitted to Multiple Partitions



Jobs submitted to multiple partitions (queues) will appear in Slurm's scheduling queue once per partition, possibly with a different priority associated with each partition.

Once a job is allocated resources, other records for that job in the queue will be ignored. The partition used to run the job will be reported by Slurm commands.

If the job starts and is requeued, it once again can be started in any of its available partitions

Outline

- Job priority
- Main scheduling logic
- Backfill scheduling
- Premption
- Gang scheduling
- Network topology
- Diagnostics

Scheduling Algorithms

- Slurm is designed to perform a quick and simple scheduling attempt at frequent intervals
  - At each job submission
  - At job completion on each of it's allocated nodes
  - At configuration changes
- Slower and more comprehensive scheduling attempts performed less frequently

Quick Scheduling

- Designed to provide nearly instant response when possible
- SchedulerParameters=**default_queue_depth=#** defines how far down the job queue to test, default value is 100
- SchedulerParameters=**partition_job_depth=#** defines how many jobs are tested in any single partition, default value is 0 (no limit)
- Once any job in a partition is left pending, no other jobs in that partition are considered for scheduling (i.e. FIFO)
- Once any task for a job array is left pending, no other tasks in that job array are considered for scheduling

Less Quick Scheduling

- Same algorithm as described on last page, but performed once each minute
  - Value of **default_queue_depth** ignored
  - Tests all jobs in the queue or run until reaching the configured **max_sched_time** time limit, default value is half of **MessageTimeout**
- Strict priority ordering in the partition is preserved, so the overhead of this is typically still fairly low
- This logic can be useful to get jobs in lower priority partitions started

More SchedulingParameters

- **defer** - Do not attempt to schedule jobs individually at submit time
  - Disables the “Quick Scheduling”
    - Lowers overhead
  - Can be useful for high-throughput computing

Outline

- Job priority
- Main scheduling logic
- Backfill scheduling
- Premption
- Gang scheduling
- Network topology
- Diagnostics

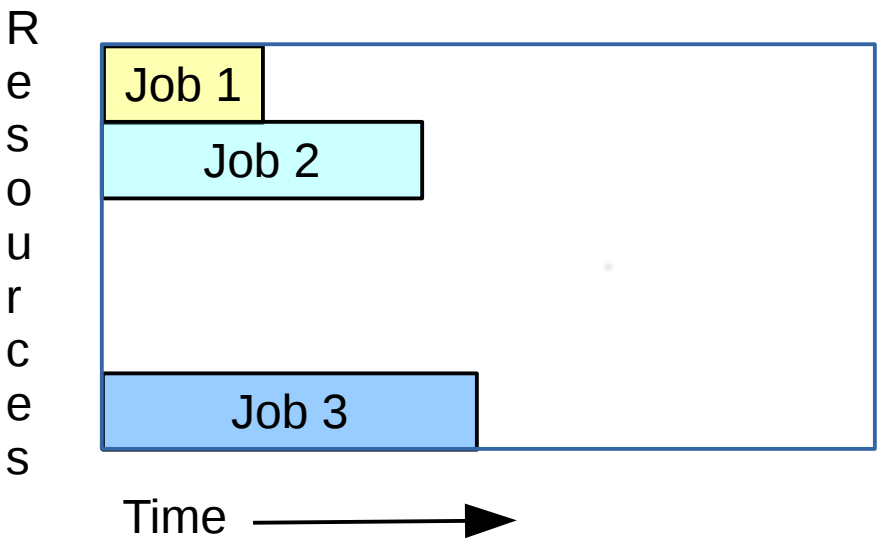
Scheduling Configuration

- **SchedulerType** - Specifies the scheduling plugin to use
  - **sched/backfill** - Backfill scheduling (default)
  - **sched/builtin** - Strict priority order within a partition/queue (e.g. First-In First-Out, FIFO)
- Both plugins set expected start time for pending jobs based upon job time limits. visible using the command “squeue –start”

Backfill Scheduling

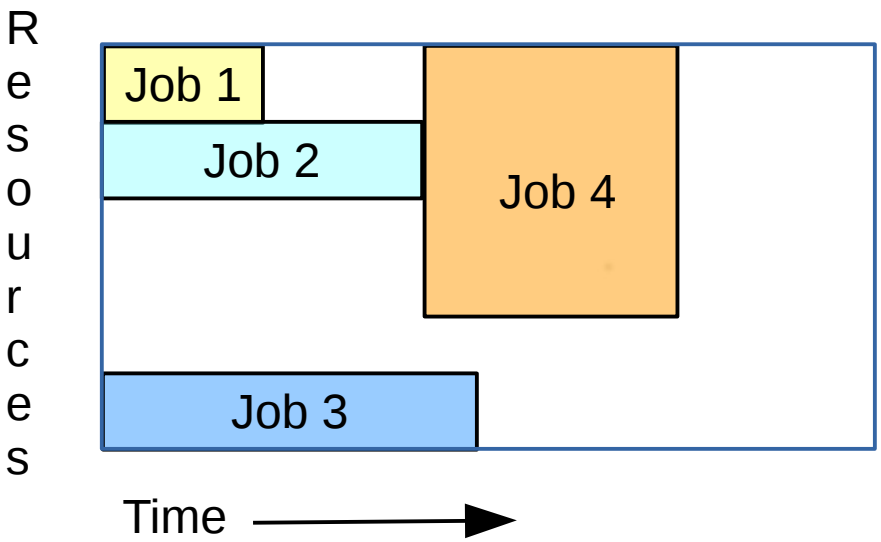
- Backfill scheduling plugin is loaded by default
- Without backfill, each partition is scheduled strictly in priority order
- Backfill scheduler will start lower priority jobs if doing so does not delay the expected start time of any higher priority job
- Since the expected start time of pending jobs depends upon the expected completion time of running jobs, reasonably accurate time limits are valuable for backfill scheduling to work well

Backfill Scheduling



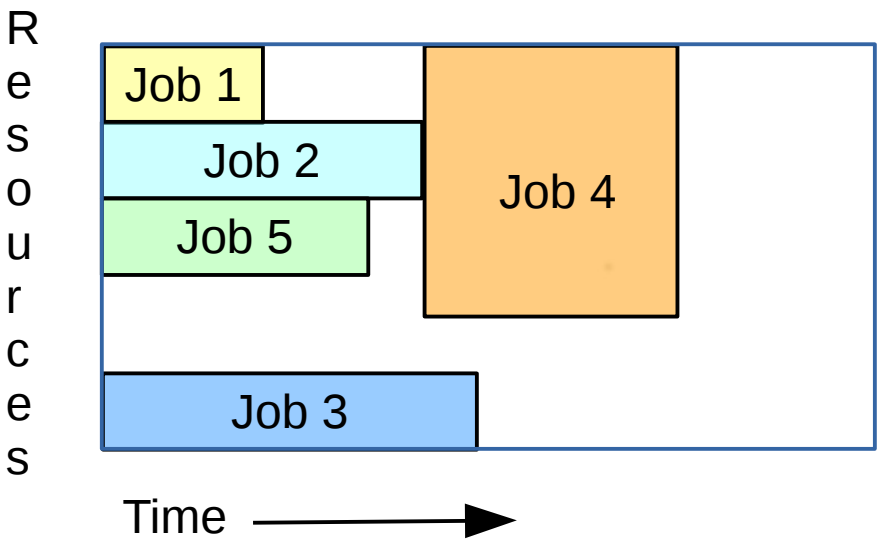
1. Jobs 1, 2 and 3 running now

Backfill Scheduling



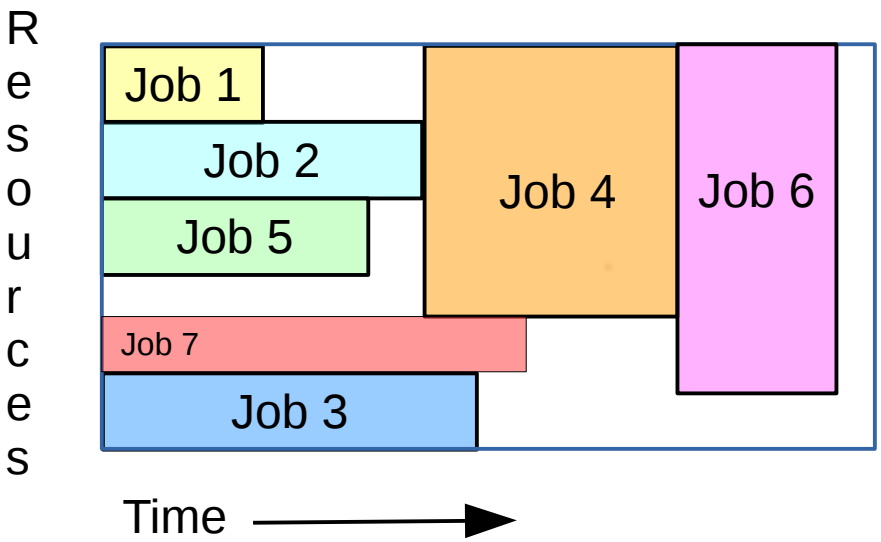
1. Jobs 1, 2 and 3 running now
2. Job 4 can not start now, waiting for job 2

Backfill Scheduling



1. Jobs 1, 2 and 3 running now
2. Job 4 can not start now, waiting for job 2
3. Job 5 can start now on available resources, but only if it completes no later than job 2 (when job 4 can start)

Backfill Scheduling



1. Jobs 1, 2 and 3 running now
2. Job 4 can not start now, waiting for job 2
3. Job 5 can start now on available resources, but only if it completes no later than job 2 (when job 4 can start)
4. Backfill scheduler goes down queue a configurable depth, determining where and when each pending job will run

Backfill Scheduling and Time Limits

- Backfill scheduling is difficult without reasonable time limit estimates for jobs, but some configuration parameters that can help

Partition configuration options:
DefaultTime - Default job time limit
MaxTime – Maximum job time limit

Global configuration option:
OverTimeLimit – Amount by which a job can exceed its time limit before it is killed

Backfill Scheduling Locks

- Backfill scheduling is a time consuming operation
- Locks are periodically released briefly so that other options can be processed (e.g. submit new jobs)
- Backfill scheduling can optionally continue execution after the lock release and ignore newly submitted jobs (**SchedulingParameters=bf_continue**). Doing so will permit consideration of more jobs, but may result in the delayed scheduling of newly submitted jobs

Backfill Scheduling Configuration

- **SchedulingParameters**
  - **bf_interval=**# - Interval between backfill scheduling attempts. Default value is 30 seconds
  - **bf_resolution=**# - Time resolution of backfill scheduling. Default value is 60 seconds
  - **bf_window=**# - How long into the future to look when determining when and where jobs can start. Default value is one day
  - **bf_continue** - If set, then continue backfill scheduling after periodically releasing locks for other operations

Backfill Scheduling Configuration

- **SchedulingParameters**
  - **bf_max_job_test=#** - Maximum number of jobs consider for backfill scheduling. Default value is 100 jobs
  - **bf_max_job_start=#** - Maximum number of jobs backfill schedule. Default value is 0 (no limit)
  - **bf_max_job_part=#** - Maximum number of jobs per partition to consider for backfill scheduling. Default value is 0 (no limit)
  - **bf_max_job_user=#** - Maximum number of jobs per user to consider for backfill scheduling. Default value is 0 (no limit)

Backfill Scheduling Configuration

- **SchedulingParameters**
  - **bf_yield_interval=#** - Time between backfill scheduler lock release. Default value 2000000 usec (2 seconds, new option in version 14.11)
  - **bf_yield_sleep=#** - Time that backfill scheduler sleeps for when locks are released. Default value 500000 usec (0.5 seconds, new option in version 14.11)
  - **max_rpc_cnt=#** - Sleep if the number of pending RPCs reaches this level. Default value is 0 (no limit)

Backfill Scheduling Configuration

- **DebugFlags**
  - **Backfill –** Log when and where jobs will start
  - **BackfillMap** – Log map of resources reserved through time for pending jobs (new option in version 14.11, same information displayed with **Backfill** flag in earlier versions of Slurm), very verbose

Backfill Limitations

- Reserves whole nodes for pending jobs rather than individual CPUs
- Does not track release of licenses through time
- Does not track how user/account/QOS limits change through time (i.e. resources might be reserved in the future for a job that will not be able to start as expected)

Outline

- Job priority
- Main scheduling logic
- Backfill scheduling
- Premption
- Gang scheduling
- Network topology
- Diagnostics

Job Preemption

- Mechanism to cause lower priority jobs to relinquish resources so that higher priority jobs can be run
- Mechanism used to determine what is a higher priority job and the preemption mechanism are configurable
- Within a given QOS or partition priority, larger jobs are given higher scheduling priority (less likely to be preempted)

Job Preemption

- The **PreemptType** configuration parameter determines which jobs can preempt each other
  - **preempt/part_prio**: Jobs from higher priority partitions can preempt jobs from lower priority partitions. This is only useful if partitions have overlapping resources
    - Set each partition's **Priority** parameter appropriately
  - **preempt/qos**: Preempt jobs based upon Quality Of Service (QOS). Jobs can be in the same or different partitions. Preemption rules defined in the database using the *sacctmgr* command

Job Preemption Mechanisms

- Preemption mechanism is configurable separately for each partition or QOS
- **Cancel**: Terminate the lower priority job
- **Checkpoint**: Checkpoint and terminate the job
- **Requeue**: Requeue the lower priority job
- **Suspend**: Suspend the lower priority job and automatically resume it when the higher priority job terminates (re-uses some gang scheduling logic). Make sure that memory use is managed
- **Off**: Preemption of these jobs is disabled

Job Preemption Sample Configuration

- Jobs in “low” partition are requeued for any higher priority jobs
- Jobs in “med” partition are suspended for jobs in “high” queue if they can both fit into memory
- Jobs in “high” partition preempt jobs from “low” and “med” partitions

```
#
# Excerpt from slurm.conf file
#
PartitionName=DEFAULT Nodes=tux[0-9]
PartitionName=high  Default=NO    Shared=FORCE:1  Priority=5  PreemptMode=off
PartitionName=med  Default=NO    Shared=FORCE:1  Priority=3  PreemptMode=suspend
PartitionName=low    Default=YES  Shared=NO          Priority=1  PreemptMode=requeue
```

Preemption Algorithm

- The preempt plugin generates a priority ordered list of preemption candidates
  - There might actually be a small number of priorities
- The select plugin simulates preemption of jobs on the list one at a time and tests the ability to start the pending job
- There are a couple of algorithms to re-sort the list in order to minimize the number of job's preempted

Preemption Example

- Option 1:
  - Higher priority jobs removed from the preemption candidate list
  - Last job needing preemption (highest priority job needed) is moved to top of preemption candidate list
  - Other jobs sorted by number of the number of their nodes in the desired allocation of the pending job
  - Algorithm is run again
  - This may result in more higher-priority jobs being preempted compared with option 2 (described later)

Preemption Example

- Option 2:
  - Higher priority jobs removed from the preemption list
  - Last job needing preemption is moved to top of preemption candidate list
  - No other sorting (vs. sorting by resources needed)
  - Algorithm is run again
  - Best used when there are multiple preemption priorities that should be considered

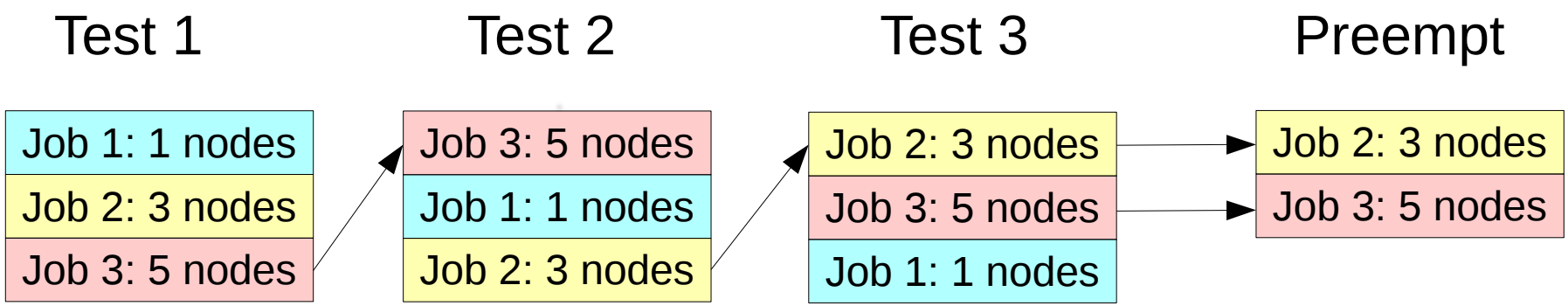
Preemption Example

- Pending job needs 8 nodes
- Running job 1 has 1 nodes
- Running job 2 has 3 nodes
- Running job 3 has 5 nodes

  - Pending job can not be started until third running job is preempted, but we only need 8 nodes, not 9 nodes
  - Reorder jobs to improve that...

Preemption Example

Pending job needs 8 nodes



Job Preemption Options

- **SchedulerParameters** options:
  - **preempt_strict_order**: Only reorder one job at a time (option 2 from previous slide)
  - **preempt_reorder_count**=#: Number of iterations for re-ordering jobs
  - Higher number reduce number of preempted jobs, but take more time to execute the scheduling logic

```
#
# Excerpt from slurm.conf file
#
SchedulerParamters=preempt_strict_order,preempt_reorder_count=3
```

Outline

- Job priority
- Main scheduling logic
- Backfill scheduling
- Premption
- Gang scheduling
- Network topology
- Diagnostics

Gang Scheduling

- Processors are over-subscribed by jobs, but the jobs take turns running by time-slicing
- Can improve system responsiveness and utilization
- All of the tasks of a job are either running or sleeping
- Job time slicing controlled by signals SIGSTOP and SIGCONT

	CPU 0	CPU 1	CPU 2	CPU 3
Time 0	Job 1	Job 2	Job 3	
Time 1	Job 4	Job 5	Job 3	
Time 2	Job 6	Job 2	Job 3	
Time 3	Job 7	Job 5	Job 3	

Gang Scheduling Memory

- The CPUs are time sliced, but all jobs must fit into memory at the same time to gang schedule them
- Use Slurm configuration parameters to control each job's memory use. Configure system wide and/or by partition
  - **DefMemPerCPU**
  - **MaxMemPerCPU**
  - **DefMemPerNode**
  - **MaxMemPerNode**

Gang Scheduling Configuration

- **SelectTypeParameter**: Configure to manage memory (e.g. “CR_Core_Memory”, “CR_CPU_Memory”, etc.)
- **PreemptMode=Gang**
- **SchedulerTimeSlice**=#: How long each time slice lasts. Default value is 30 seconds
- **Shared**: In partitions with gang scheduling, set to “FORCE:#”, where “#” is the maximum number of jobs which can be time sliced on each resource in that partition

Gang Scheduling and CPU Frequency

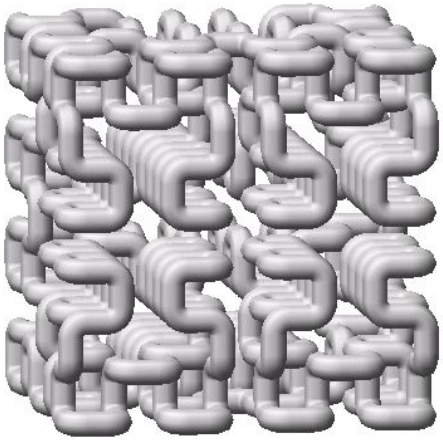
- Beginning with Slurm version 14.11, the job's CPU frequency and CPU governor are preserved (reset as needed as part of job resumption)

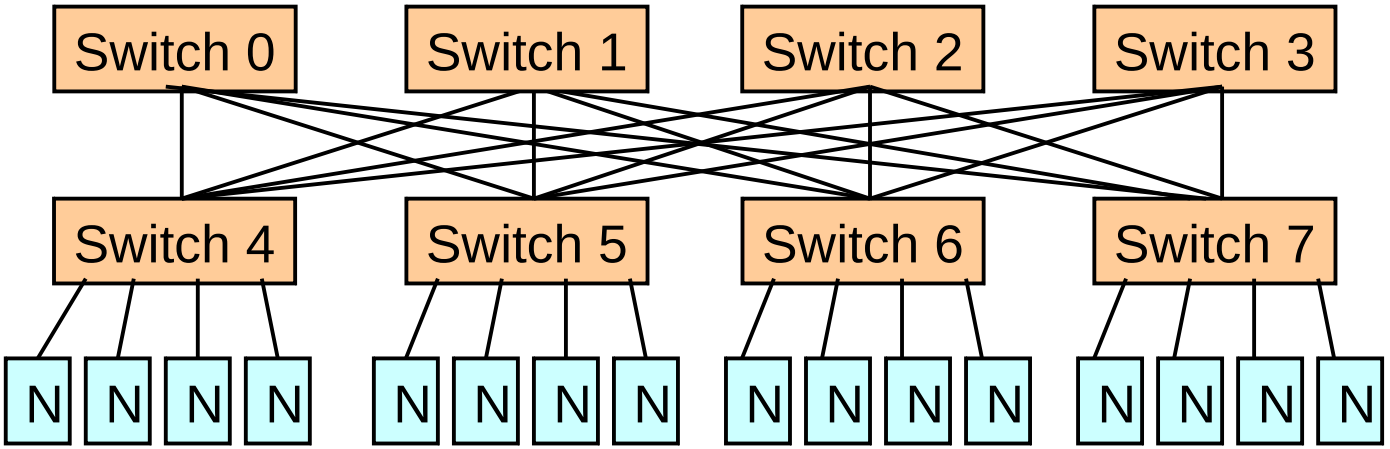
Outline

- Job priority
- Main scheduling logic
- Backfill scheduling
- Premption
- Gang scheduling
- Network topology
- Diagnostics

Network Topology

- Several plugins are available to optimize job allocations with respect to network topology
  - topology/3d_torus: Uses Hilbert space-filling curve to order nodes so that Slurm's native best-fit algorithm achieves good locality in 3d-space
  - topology/tree: Minimizes communication latency

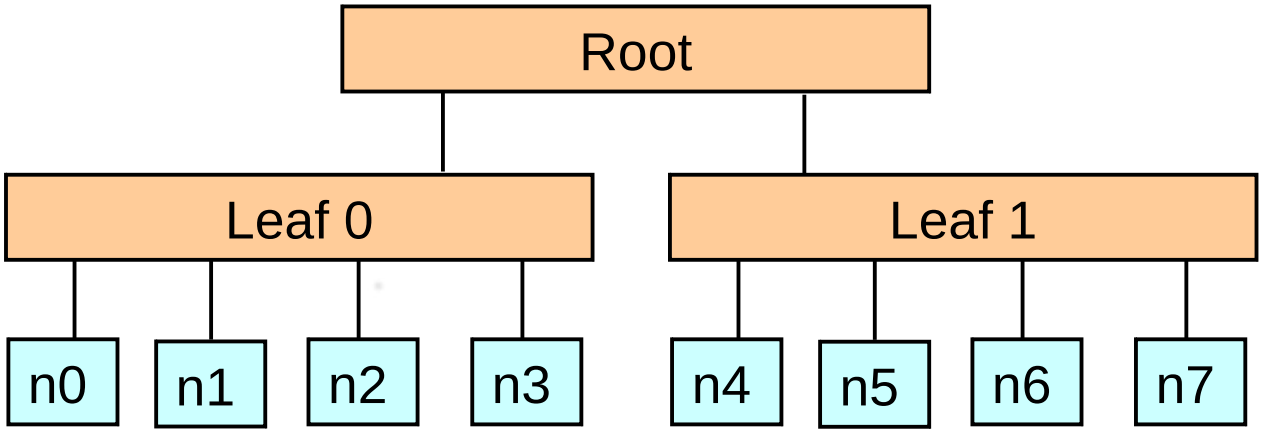




Topology/tree

- Relies upon administrator defined topology.conf file
- Define switches and their child switches or nodes
- Defining the leaf switches and nodes are important
- Defining other switches is generally less important and some sites just define a single top level switch (achieves good results, reduces scheduling overhead, and easier to maintain)
- The switch names are arbitrary, but available to jobs for managing their communications

Sample topology.conf




```
#
# Sample topology.conf file
#
switch=leaf0 nodes=n[0-3]
switch=leaf1 nodes=n[4-7]
switch=root switches=leaf[0-1]
```

Give switches any name that you find convenient

Job Topology Options

- Option for salloc, sbatch and srun:
  --switches=count[@time]

  Allocate job at most “count” leaf switches

  Wait up to “time” for that leaf count
- Administrator can configure maximum “time” honored using *SchedulingParameters* option *max_switch_wait*

Job Topology Environment

- Job environment variables define switches for each task up to the job's root switch:
  - SLURM_TOPOLOGY_ADDR (names)
  - SLURM_TOPOLOGY_ADDR_PATTERN (series of “switch” or “node”)

```
> srun –label -N3 env | grep TOPOLOGY
0: SLURM_TOPOLOGY_ADDR=root,leaf0,n0
0: SLURM_TOPOLOGY_ADDR_PATTERN=switch,switch,node
1: SLURM_TOPOLOGY_ADDR=root,leaf0,n1
1: SLURM_TOPOLOGY_ADDR_PATTERN=switch,switch,node
2: SLURM_TOPOLOGY_ADDR=root,leaf1,n4
2: SLURM_TOPOLOGY_ADDR_PATTERN=switch,switch,node
```

Outline

- Job priority
- Main scheduling logic
- Backfill scheduling
- Premption
- Gang scheduling
- Network topology
- Diagnostics

sdiag Command

- Reports scheduling performance
  - Scheduler run time
  - Queue depth
  - Jobs started, etc.
- Reports API calls made
  - Count and time
  - User ID making calls

When and where will my job start?

- Run “squeue –start”
  - Nodes expected to be used are reported started in Slurm version 14.11

Why is my job not running?

- As soon as some reason is found why a job can not be started, that is recorded in the job's “reason” field and the scheduler moves on to the next job

Some common reasons why jobs are pending:

Priority	Resources being reserved for higher priority job
Resources	Required resources are in use
Dependency	Job dependencies not yet satisfied
Reservation	Waiting for advanced reservation
AssociationJobLimit	User or bank account job limit reached
AssociationResourceLimit	User or bank account resource limit reached
AssociationTimeLimit	User or bank account time limit reached
QOSJobLimit	Quality Of Service (QOS) job limit reached
QOSResourceLimit	Quality Of Service (QOS) resource limit reached
QOSTimeLimit	Quality Of Service (QOS) time limit reached

Documentation

- Scheduling Configuration Guide
  http://slurm.schedmd.com/sched_config.html
- Gang Scheduling
  http://slurm.schedmd.com/gang_scheduling.html
- Preemption
  http://slurm.schedmd.com/preempt.html
- Network topology Guide
  http://slurm.schedmd.com/topology.html
- High Throughput Computing Guide
  http://slurm.schedmd.com/high_throughput.html
- slurm.conf configuration file
  http://slurm.schedmd.com/slurm.conf.html

Questions?