LCI workshop
on high performance clustered computing

LCI Advanced Workshop 2025: **Resource Limits**

Josh Burks
HPC Systems Analyst
Arizona State University Research Computing
josh.burks@asu.edu

ASU Research Computing

Outline

- Explore some of the different mechanisms for controlling the different resources different users can access
- Explore how condo models interact with resource limits
- Discuss the limits we can set to
    - Maintain fairness between research and classroom workloads
    - Enforce allocations that match funding or ownership
    - Keep utilization high while maintaining predictable access
- Explore how job priority is assigned
- Explore now nodes are selected for jobs

Resource Enforcement

Target	Role in Resource Limits	Example of Controls
User	Per-User Limits	MaxTRES, MaxJobs
Account	Groups users for accounting and fairshare	GrpTRES,MaxJobsPerAccount
QoS	Defines policy, priority, and quotas	MaxTRES, MaxWall, Preemption, PriorityWeightQOS
Partition	Groups nodes for scheduling and access control	MaxNodes, MaxTime, AllowAccounts, PriorityTier

Association, QoS, and Partitions

- Accounts are “billing” accounts
- Associations map users to accounts to clusters and are the primary vehicle for long-term fairshare calculator and per-group limits
- QoS’s create policy around scheduling priority and overriding limits, including the impact to fairshare with per-user/account TRES limits, fairshare decay, UsageFactors
- Partitions create groups of nodes with access policies and scheduling queues with basic job-level limits

Account Limits

LCI workshop
on high performance clustered computing

- Accounts define who can use resources and how much they can consume.
- Limits cascade through hierarchy:
    - User -> Account -> Cluster -> Global
- Departmental chargeback or usage tracking
- Restricting total cores, GPUs, or jobs per group
- Enforcing per-user quotas inside shared accounts

```
sacctmgr modify account research set \
 GrpTRES=cpu=2048 MaxJobs=100
```

ASU Research Computing

Association Limits

- Association is the mapping of User define who can use resources and how much they can consume.
- Limits cascade through hierarchy:
  - User -> Account -> Cluster -> Global

```
sacctmgr modify user where name=josh account=research set \
  MaxJobs=10 MaxSubmitJobs=20
```

Departmental Hierarchy

Parent-Child Hierarchical Limits

```
sacctmgr add account research parent=root GrpTRES=cpu=8000
sacctmgr add account chem parent=research GrpTRES=cpu=2000
sacctmgr add account pi_joe parent=chem GrpTRES=cpu=512
sacctmgr add user josh account=pi_joe MaxTRESPerJob=cpu=32
```

Departmental Hierarchy

Parent-Child Hierarchical Limits

```
# sacctmgr list association tree \
format=Account,User,GrpTRES,MaxTRES
Account                    User       GrpTRES       MaxTRES
-------------------- ---------- ------------- -------------
root                                cpu=20000
root                      root
research                            cpu=8000
 college_chem                       cpu=2000
  pi_jones                           cpu=512       cpu=512
   pi_jones               josh                      cpu=32
```

QoS Fundamentals

- QoS provides flexible controls: priority weight, max jobs, max walltime, max TRES, preemption flags, partition override flags
- PriorityWeightQOS multiplies into job priority to favor or deprioritize groups (Normalized value)
- QoS can set MaxTRESPerUser, MaxTRESPerJob, MaxSubmitJobs and enforce quotas
- QoS may be configured to preempt lower-QoS jobs

QoS Resource Limits

LCI workshop on high performance clustered computing

- Classroom policy: small MaxWall (2h), limited CPUs , fewer GPUs, higher QoS priority
- Research policy: larger MaxWall, larger MaxTRES limits, higher-priority QoS, access to private nodes.

```
sacctmgr add qos research Priority=100 \
  MaxWall=72:00:00


sacctmgr add qos classroom Priority=200 \
MaxWall=02:00:00 MaxSubmitJobs=2 \
```

ASU Research Computing

QoS Resource Limits:

- Default research QoS

```
sacctmgr create qos name=research
```

QoS Resource Limits:

- Longer run time

```
sacctmgr create qos name=long \
 MaxJobsPerUser=2 \
 MaxWallDurationPerJob=14-00:00:00 \
 MaxTRESPerUser=cpu=512 \
 Flags=PartitionTimeLimit

sacctmgr modify user where user=jeburks2 \
 account=pi_joe qos+=long
```

QoS Resource Limits:

LCI workshop on high performance clustered computing

- Debugging QoS

```
sacctmgr create qos name=debug \
 Priority=20000000 \
 MaxJobsPerUser=2 \
 UsageFactor=0.5 \
 MaxWallDurationPerJob=15 \
 Flags=DenyOnLimit
```

ASU Research Computing

QoS Resource Limits:

- Idle Cycles

```
sacctmgr create qos idlecycles \
 UsageFactor=0.001 \
 Flags=DenyOnLimit,NoReserve
```

QoS Resource Limits:

- Condo Node Owners

```
sacctmgr create qos pi_joe \
 Priority=500000 \
 Preempt=idlecycles \
 UsageFactor=0 \
 Flags=PartitionTimeLimit,DenyOnLimit,OverPartQOS
```

```
Assumes: PreemptType=preempt/qos
```

QoS Resource Limits:

- Classroom Limited Use

```
sacctmgr create qos name=class \
Priority=200 \
MaxJobsPerUser=2 \
MaxWallDurationPerJob=1440 \
MaxTRESPerUser=gres/gpu=1,cpu=32,mem=120G
```

Flexible Time-Based Limits

- Sometimes fixed limits are too broad
- Maybe limiting classroom users to 1 GPU is too strict
- Maybe classroom users need 4 GPUs, but for a shorter duration
- Flexible time-based limits allow users longer wall times when they use less resources
- Use MaxTRESRunMinsPU and MaxTRESRunMinsPerAccount

Maybe we want to classroom academic courses users run at most jobs combining to 16 hours (960 minutes) of GPU time

1 GPU for 16 Hours / 2 GPUs for 12 Hours / 3 GPUs for 8 Hours / 4 GPUs for 4 Hours

QoS Resource Limits:

- Classroom Limited Use

```
sacctmgr create qos name=class \
Priority=200 \
MaxJobsPerUser=2 \
MaxWallDurationPerJob=1440 \
MaxTRESRunMinsPU=gres/gpu=960,cpu=768
```

QoS in the Scheduling Formula

LCI workshop on high performance clustered computing

- Job priority in Slurm is calculated from multiple weighted factors:
- Priority = Age + Fairshare + JobSize + Partition + QOS + Site
- QoS priority contributes through PriorityWeightQOS in slurm.conf.
- Example:
  - PriorityWeightQOS=1000000
  - Higher weight = stronger impact of QoS on total job priority.
  - Use sprio --long to inspect the breakdown for queued jobs.
  - Combine with PriorityTier in partitions to fine-tune scheduling order.

Analyzing Priority

```
# sprio --long
  JOBID PARTITION     USER    PRIORITY        AGE  FAIRSHARE PARTITION    QOSNAME          QOS
33553665 public       userA   10000009   10000000         10         0     public            0
34374167 highmem      userB   10000011   10000000         11         0     public            0
34374170 highmem      userB   10000011   10000000         11         0     public            0
34398154 public       userA   10000009   10000000         10         0     public            0
34556382 highmem      userC   10000228   10000000        228         0     public            0
34556395 highmem      userC   10000228   10000000        228         0     public            0
34556399 highmem      userC   10000228   10000000        228         0     public            0
34834438 general      userD  219843814   10000000   99843815         0      grp_c    110000000
34852751 general      userE  168124584   10000000   97624585         0      debug     60500000
34854466 public       userF   10632112   10000000     632113         0     public            0
34864821 highmem      userG   10261876   10000000     261876         0     public            0
34871673 highmem      user    10000000   10000000          0         0     public            0
```

Analyzing Priority

```
Set DebugFlags to NO_CONF_HASH,Priority
debug:  slurmctld log levels: stderr=debug logfile=debug syslog=quiet
Set debug level to 'debug'
priority/multifactor: _get_fairshare_priority: PRIO: Fairshare priority of job 5 for user root in acct root is
2**(-1.000000/1.000000) = 0.500000
priority/multifactor: _get_priority_internal: Weighted Age priority is 0.000000 * 10000000 = 0.00
priority/multifactor: _get_priority_internal: Weighted Assoc priority is 0.000000 * 0 = 0.00
priority/multifactor: _get_priority_internal: Weighted Fairshare priority is 0.500000 * 100000000 = 50000000.00
priority/multifactor: _get_priority_internal: Weighted JobSize priority is 0.000000 * 0 = 0.00
priority/multifactor: _get_priority_internal: Weighted Partition priority is 0.000000 * 0 = 0.00
priority/multifactor: _get_priority_internal: Weighted QOS priority is 0.000000 * 110000000 = 0.00
priority/multifactor: _get_priority_internal: Site priority is 0
priority/multifactor: _get_priority_internal: Job 5 priority: 0 +  0 + 0.00 + 50000000.00 + 0.00 + 0.00 + 0.00 +  0 -
0 = 50000000.00
PRIO: BillingWeight: JobId=5 is either new or it was resized
PRIO: BillingWeight: JobId=5 using "CPU=1.0,Mem=.25G,gres/gpu=3.0" from partition general
PRIO: BillingWeight: JobId=5 SUM(TRES) = 1.500000
sched: _slurm_rpc_allocate_resources JobId=5 NodeList=lci-compute-XX-2 usec=673
```

Preemption with QoS

- QoS can define who gets to reclaim resources when the system is full
- Preemption modes determine what happens to lower-priority jobs
    - REQUEUE / CANCEL / SUSPEND
- Preemption rules are controlled by PreemptMode and PreemptType:

```
PreemptType=preempt/qos
PreemptMode=REQUEUE # REQUEUE / CANCEL / SUSPEND
```

Condo Model

LCI workshop
on high performance clustered computing

- PI or research group contributes hardware ('buys in') to a shared cluster
- Nodes integrated into the global system for scheduling and accounting
- Owners expect priority/guaranteed access, opportunistic use of other resources
- Admins want high utilization, enforceable policies, fairness across groups

ASU Condo Implementation

- Approach:
  - One 'condo' partition for all condo owners.
  - Each group has a QoS with a priority boost
  - Condo-owners nodes are targeted with JSP
  - Non-owner jobs may run on idle condo nodes with the idlecycles QoS with preemption
- Benefits:
  - Simple configuration (1 partition + per-group QoS).
  - Owners rarely wait on their nodes.
  - Idle time minimized.
- Drawbacks
  - In order to enforce node restrictions via JSP, editing of submitted jobs is disabled

Other Condo Strategies

- Dedicated Partitions or QoS (per group):
  - Strict node ownership, owners only
  - Pros: Clear ownership, owners own a specific piece of hardware, simple job submission for users
  - Cons: Many partitions to manage
- Reservation-Based Model:
  - Guarantees “equivalent hardware” via reservations.
  - Pros: flexible, better utilization.
  - Cons: owners may dislike non-dedicated nodes.
- Hybrid Approaches:
  - Nodes in owner-only and shared partitions.
  - QoS ensures owners get preference
  - Balances efficiency and ownership

Preemption in Condo Models

LCI workshop
high performance clustered computing

- Why Preemption?
  - Ensures owners regain condo resources quickly.
  - Protects guarantees while allowing opportunistic use.
- Strategies:
  - PreemptMode=REQUEUE - requeue outsider jobs.
  - PreemptMode=CANCEL - cancel outsider jobs outright.
  - PreemptMode=SUSPEND - pause jobs, resume later.
- QoS preemption rules - owners’ QoS can preempt others.
- Trade Offs:
  - REQUEUE: fair, but can frustrate users.
  - CANCEL: harsh, instant access for owners.
  - SUSPEND: jobs stay in memory
  - QoS layering: flexible but complex.

Partition Fundamentals

- Partitions are logical groups of nodes that define where jobs can run
- Each partition can have unique limits, scheduling behavior, and access policies
- Common parameters:
  - Nodes= list of nodes in the partition
  - Default=YES/NO jobs submitted without a -p flag will go there
  - PriorityTier= affects scheduling order relative to other partitions
  - MaxTime=, MaxNodes=, MaxCPUsPerUser= resource limits
  - AllowAccounts=, AllowQos= restrict who can submit

- Typical use cases:
  - Separate CPU, GPU, and high-memory nodes
  - Isolate public/condo nodes

Partition Fundamentals

LCI workshop
on high performance clustered computing

- Partitions can also inherit ***some*** of the resource limits of a QoS
- Only inherits limits that can already apply to partitions
    - Does not assign jobs to the QOS
    - Does not give the job any priority characteristics of the QoS
    - Does not give the job any preemption characteristics of the QOS
    - Does not inherit accounting limits (ie, time-based limits)

Partition Fundamentals

- Partitions are like “queues” in PBS/Torque - jobs are grouped and prioritized within each partition.
- Thinking of them as queues helps conceptualize how Slurm schedules jobs.
- The main scheduler processes partitions in order of PriorityTier.

Partition Fundamentals

- When it hits a high-priority job that can't start (e.g., needs 64 GPUs unavailable), it stops evaluating lower-priority jobs in that same partition for that pass
- The backfill scheduler, which runs afterward, can still start smaller or lower-priority jobs if they fit without delaying higher-priority ones
- This behavior means a single large job can block smaller jobs in the same partition.
- Creating dedicated partitions for resource-intensive hardware (e.g., GPUs, large-memory nodes) prevents these bottlenecks and improves overall throughput.

Node Weights

- Assuming all other factors are equal and several idle nodes can satisfy the requested resources - how does slurm pick which node to run on?
- Node “weights” in slurm.conf influence the order in which nodes are allocated
- Higher Weight = lower scheduling preference
- Useful for favoring newer, faster, or high-bandwidth nodes

```
NodeName=compute[001-199] Weight=1000 # Slow, Old Nodes
NodeName=compute[200-299] Weight=1 # Fast, New Nodes
```

Node Weights

- Try it on your lab VM:
- Submit a job, and notice how you land on node 1

```
[root@lci-head-XX-1 ~]# salloc -p general,
salloc: Granted job allocation 3
salloc: Nodes lci-compute-XX-1 are ready for job
```

Node Weights

- Try it on your lab VM:
- Update your nodes to assign different weights, and notice how you land on node 2

```
scontrol update nodename=lci-compute-XX-1 weight=1000


[root@lci-head-XX-1 ~]# salloc -p general
salloc: Granted job allocation 4
salloc: Waiting for resource configuration
salloc: Nodes lci-compute-XX-2 are ready for job
```

Node Weights

- To make this permanent, assign weight in /etc/slurm/slurm.conf to assign your nodes different weights

```
NodeName=lci-compute-XX-[1-2] \
Weight=1 \
CPUs=2 \
Boards=1 \
SocketsPerBoard=2 \
CoresPerSocket=1 \
ThreadsPerCore=1 \
RealMemory=7500
```

Node Weights

- To make this permanent, assign weight in /etc/slurm/slurm.conf to assign your nodes different weights

```
NodeName=lci-compute-XX-1 \
Weight=1000 \
CPUs=2 \
Boards=1 \
SocketsPerBoard=2 \
CoresPerSocket=1 \
ThreadsPerCore=1 \
RealMemory=7500
```

```
NodeName=lci-compute-XX-2 \
Weight=1 \
CPUs=2 \
Boards=1 \
SocketsPerBoard=2 \
CoresPerSocket=1 \
ThreadsPerCore=1 \
RealMemory=7500
```

Q & A