Brigham Young University

Fulton Supercomputing Lab

Ryan Cox

SC15 – November 17, 2015 – Austin, TX

Outline

- Solution for ssh-launched processes
  - Identify initiator of ssh connection
  - pam_slurm_adopt: Adopt ssh-launched processes into Slurm job
- New web-based tools for account coordinators to manage their accounts

History

- Switched to Slurm 2.5 from Moab/Torque in January 2013
- SchedMD support
- We're crazy
  - We did a split-brain, rolling transition to Slurm
  - Then we told our users
    - Almost flawless
- Resources are free to use

Node sharing

- Shared node access unless user specifies --exclusive
- cgroups plugins enforce resource requests
- Linux namespaces control /tmp, /dev/shm
- Users are prevented from harming each other
- Works great except for ssh, the poor man's MPI

ssh-launched processes

- ssh-launched processes escape Slurm's oversight
- Instead of being a child of slurmd/slurmstepd, processes are children of sshd
- sshd knows nothing about Slurm
  - No accounting, cgroups, cleanup on exit, etc.
- Best answer: Don't use ssh to launch tasks
- Problem: Reality

ssh->srun wrapper?

- An ssh wrapper sounds like a good idea...
- But is it 1:1 or 1:many tasks that will be launched on the remote node? In other words...
  - Will the job run *“ssh node7 ./dostuff-threaded -n 16”* **once?**
  - Or will the job run *“ssh node7 ./dostuff-serial”* **16 times**?
- If the former, *srun -n1* will only allow it access to one core but *srun -N1 --exclusive* will work
- If the latter, *srun -n1* will work but *srun -N1 --exclusive* will block the next 15 instances
- Since it's just a wrapper, the wrapper can't know which behavior to use
- **Each** ssh wrapper instance would result in RPC to slurmctld to create a new step. That can be **many** steps per job.

Obvious solution: PAM module

- PAM module should “adopt” ssh process into the correct job
  - Account for it, move into correct cgroups, etc.
- Problem: How does PAM know which job the ssh connection belongs to?

ssh's SendEnv/AcceptEnv + pam

- Use ssh's SendEnv/AcceptEnv to send $SLURM_JOB_ID?
- Confirmed to not work by openssh developer
  - https://lists.mindrot.org/pipermail/openssh-unix-dev/2013-October/031701.html
- ssh runs pam authentication and session modules *before* user-provided environment is set

sshrc?

- /etc/ssh/sshrc might actually work
  - **Does have access** to user-provided environment variables
  - Runs as user, not root
- …except that users can override with ~/.ssh/rc
  - Rare
  - Can disable as of openssh 6.7 w/PermitUserRC=no
    - https://bugzilla.mindrot.org/show_bug.cgi?id=2160
- Users can override $SLURM_JOB_ID
  - Intentionally, accidentally, exec*e functions
- Can mess up sshrc in a way that breaks X11 and other things
- Messy in general but would likely work if PermitUserRC=no

Does netstat have the answer?

- Trace the incoming connection
- netstat -np *# destination of ssh connection*

```
Proto Recv-Q Send-Q Local Address           Foreign Address         State       PID/Program name
tcp        0      0 10.1.123.106:42008      11.22.33.44:80          ESTABLISHED 6147/firefox
tcp        0      0 10.1.123.106:37461      10.22.234.210:22        ESTABLISHED 3804/ssh
tcp        0      0 10.1.123.106:59790      23.456.789.01:993       ESTABLISHED 28218/thunderbird
tcp        0      0 10.1.123.106:22         12.3.456.78:55777       ESTABLISHED 14881/sshd: ryancox
```

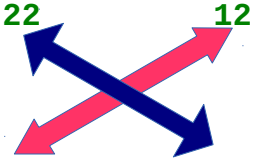
- Last line is incoming to sshd
- netstat -np # source of ssh connection

```
tcp        0      0 12.3.456.78:55777       10.1.123.106:22         ESTABLISHED 23181/ssh
```

Solution

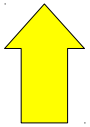
- Destination:

```
tcp        0      0 10.1.123.106:22         12.3.456.78:55777       ESTABLISHED 14881/sshd: ryancox
```



- Source:

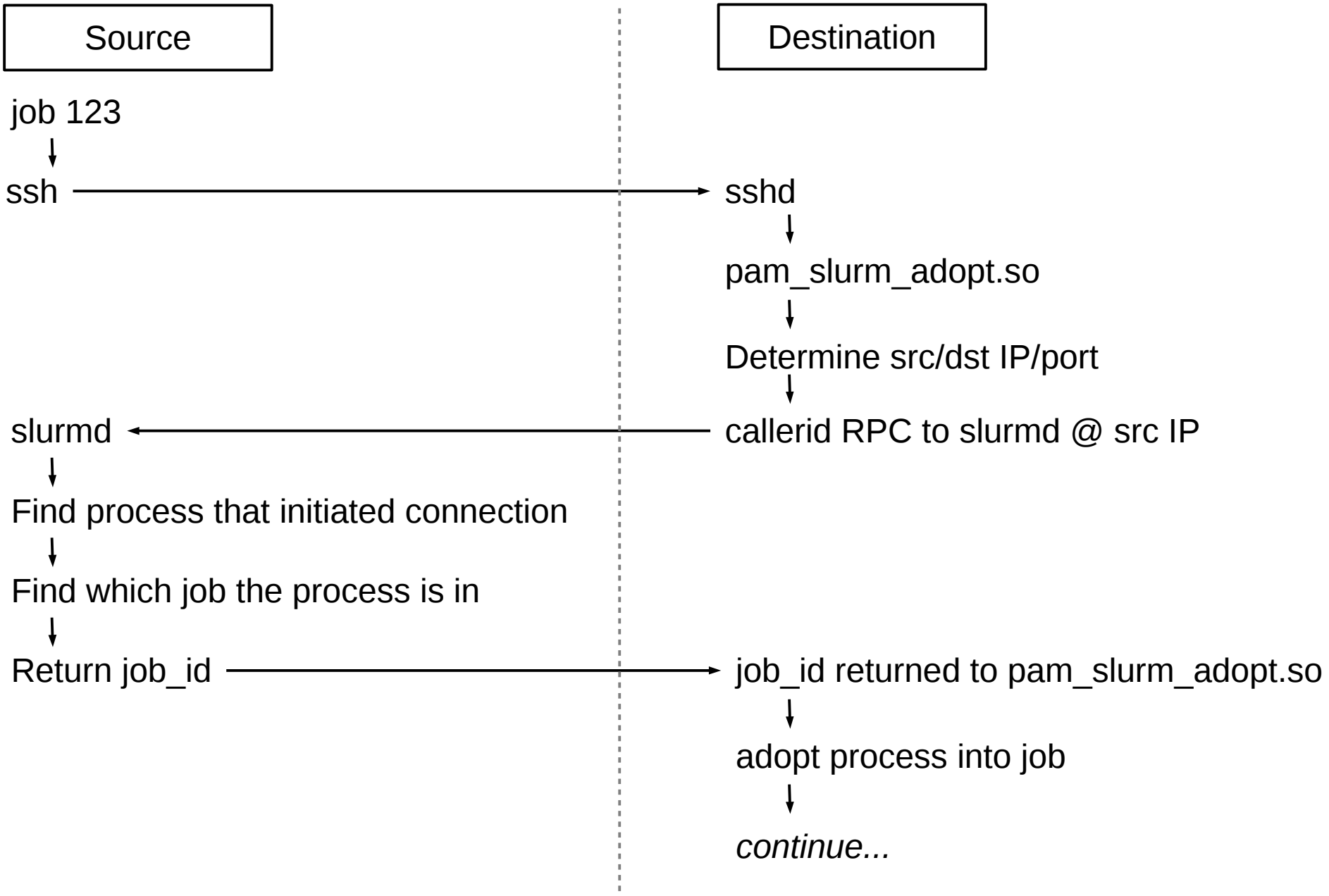
```
tcp        0      0 12.3.456.78:55777       10.1.123.106:22         ESTABLISHED 23181/ssh
```



- The source host can determine the pid of the originating process
- Slurm tracks all processes
- Look up the job_id by pid with existing *slurm_pid2jobid()*

New RPC

- “CallerID” - ask originating server for job ID using new RPC
- Created REQUEST_NETWORK_CALLERID
  - RPC itself is OS-agnostic
  - 5 fields: src and dst IP and port, IP version (4, 6)
- Response contains job_id and nodename
- Available starting with 15.08.0


Source
Destination
job 123
ssh
sshd
pam_slurm_adopt.so
Determine src/dst IP/port
slurmd
callerid RPC to slurmd @ src IP
Find process that initiated connection
Find which job the process is in
Return job_id
job_id returned to pam_slurm_adopt.so
adopt process into job
continue...

scontrol

- Support for RPC added to scontrol:

  scontrol callerid <src_ip> <src_port> <dst_ip> <dst_port> <4|6>

  - Remember that src is the source of the ssh connection, i.e. scontrol will connect to src, not dst
- `scontrol callerid 192.168.0.99 49129 192.168.0.43 22 4`
- `scontrol callerid ::2 7788 ::1 22 6`

Caveats

- Source IP address must have listening slurmd
  - i.e. ssh connection must originate from an IP address that the local slurmd is listening on
- Linux-only for now but probably not hard to port
  - RPC itself is OS-agnostic
- IPv6 is ready but Slurm RPCs are IPv4 only

Process Adoption

- Jobs are “adopted” into the appropriate job by pam_slurm_adopt.so
- A step must already exist on the node for processes to be adopted into
- SchedMD added a generic “extern” job step that is created at job launch time
  - Enabled with PrologFlags=contain
  - The adopted process will go into this step
- SchedMD also added stepd_add_extern_pid()

pam_slurm_adopt

- Several scenarios to handle in pam_slurm_adopt:
  1) User has zero jobs
  2) User has one job
  3) User has multiple jobs
     a) RPC successfully identifies the job
     b) RPC fails to identify the job

pam_slurm_adopt: Zero jobs

- If the user has zero jobs, deny the connection
  - Returns PAM_PERM_DENIED
  - Other PAM modules can allow access anyway, such as pam_access
- Can modify this behavior with *action_no_jobs* parameter

pam_slurm_adopt: One job

- If the user has one job, assume the connection is associated with that job
- The only reason the user has access to the node is due to this one job, so we'll assume that they're associated
- Adopt process into the job

pam_slurm_adopt: Multiple jobs

- Use CallerID RPC to identify the remote job
- If it identifies a job, adopt the process into it
- This will work for “poor man's MPI”
  - MPI without Slurm support (by design or incorrect compilation), Hadoop, ssh, etc.
  - Anything launched from a batch job

pam_slurm_adopt: Multiple jobs

- If the RPC fails to identify the job: problem!
- Most common case: User uses ssh on a login node to connect to a compute node w/multiple jobs
  - Checking on a job (top, ps, strace, etc.)
- There is no perfect solution
- *action_unknown* parameter defines the behavior

pam_slurm_adopt: action_unknown

- action_unknown=**newest** (default)
  - Pick the newest job on the node
  - User may be checking on job A but gets adopted into job B instead, then job B exits before job A
  - Seems least bad
- ~~action_unknown=**user**~~
  - Used to use uid_$UID cgroup. Removed when stepd_add_extern_pid was implemented since it can't be supported
- action_unknown=**allow**
- action_unknown=**deny**

Status

- 15.08.3 mostly works and is safe to use
  - Processes are adopted and limited
  - Accounting does not work and stray processes are not cleaned up after a job exits
- 15.08.4 reworked to use new stepd_add_extern_pid function
  - Bug 2096: Stray processes not cleaned up
  - Bug 2097: ~~Accounting not working~~ cpuset cgroup not created
- 15.08.5 will likely be completely functional
  - Bug 2096 resolved? *(as of this morning?)*

Fairshare

- Fair Tree fairshare algorithm
- Works well for us
- How many of you are using or planning to switch to Fair Tree?

Open Source Code

- I'll reference several codes we have open sourced

http://github.com/BYUHPC

Script Generator

- Available on github
  - https://github.com/BYUHPC/BYUJobScriptGenerator
- User education and support made easy
- We don't teach basic syntax anymore

Parameters	
Limit this job to one node:	☐
Number of processor cores **across all nodes**:	1024
Number of GPUs:	4
Memory per processor core:	4 GB ▼
Walltime:	72 hours 00 mins 00 secs
Job is a **test** job:	☐
Job is preemptable:	☐
I am in a file sharing group and my group members need to read/modify my output files:	☐
Need licenses?	☐
Job name:	
Receive email for job events:	☐ begin ☐ end ☐ abort
Email address:	myemail@example.com

Features:

☐ **amd [?]** Nodes avail: 0/2 Cores avail: 0/32	☒ **avx [?]** Nodes avail: 35/320 Cores avail: 817/5120	☒ **ib [?]** Nodes avail: 71/356 Cores avail: 1185/5488
☐ **intel [?]** Nodes avail: 114/894 Cores avail: 2266/12076	☐ **m2050 [?]** Nodes avail: 1/1 Cores avail: 12/12	☐ **s1070 [?]** Nodes avail: 1/2 Cores avail: 8/16
☐ **sse4.1 [?]** Nodes avail: 114/894 Cores avail: 2266/12076	☐ **sse4.2 [?]** Nodes avail: 113/892 Cores avail: 2258/12060	

Partitions:

☐ **p1 [?]** Nodes avail: 35/320 Cores avail: 817/5120	☐ **p2 [?]** Nodes avail: 78/572 Cores avail: 1441/6940	☐ **p3 [?]** Nodes avail: 2/32 Cores avail: 39/512

Job Script

Script format: Slurm ▼

```
#!/bin/bash

#Submit this script with: sbatch thefilename

#SBATCH --time=72:00:00   # walltime
#SBATCH --ntasks=1024   # number of processor cores (i.e. tasks)
#SBATCH --gres=gpu:4
#SBATCH -C 'avx&ib'   # features syntax (use quotes): -C 'a&b&c&d'
#SBATCH --mem-per-cpu=4G   # memory per CPU core


# LOAD MODULES, INSERT CODE, AND RUN YOUR PROGRAMS HERE
```

Features

- Users request features (-C avx) rather than partitions
  - avx, avx2, fma, ib
- Also request memory, time
  - Defaults: 1 core, 512 MB memory, 15 minutes
- Lua job submit plugin removes unusable partitions from the list of partitions they can use
- Least capable nodes scheduled first
- Better resource utilization, shorter avg. queue times

Job submit plugin

- Arbitrary business logic:
  - Users should have access to the “special_snowflake” QOS on the third Wednesday of the month if the user's uid modulo day of month == 3 but only if they request 7 cores, 2 GPUs, and between 64 and 70 GB of memory
- General layout:
  - all_partitions plugin sets a job's partition to all partitions, unless user requested a specific one(s)
  - Lua script checks if job can run in partition. If not, remove from list
    - Memory, CPU count, GRES
  - Assigns to particular QOS as applicable
- Business logic is worth the few hours of learning
  - Really, it's just a few hours

Maximum Job Timelimits

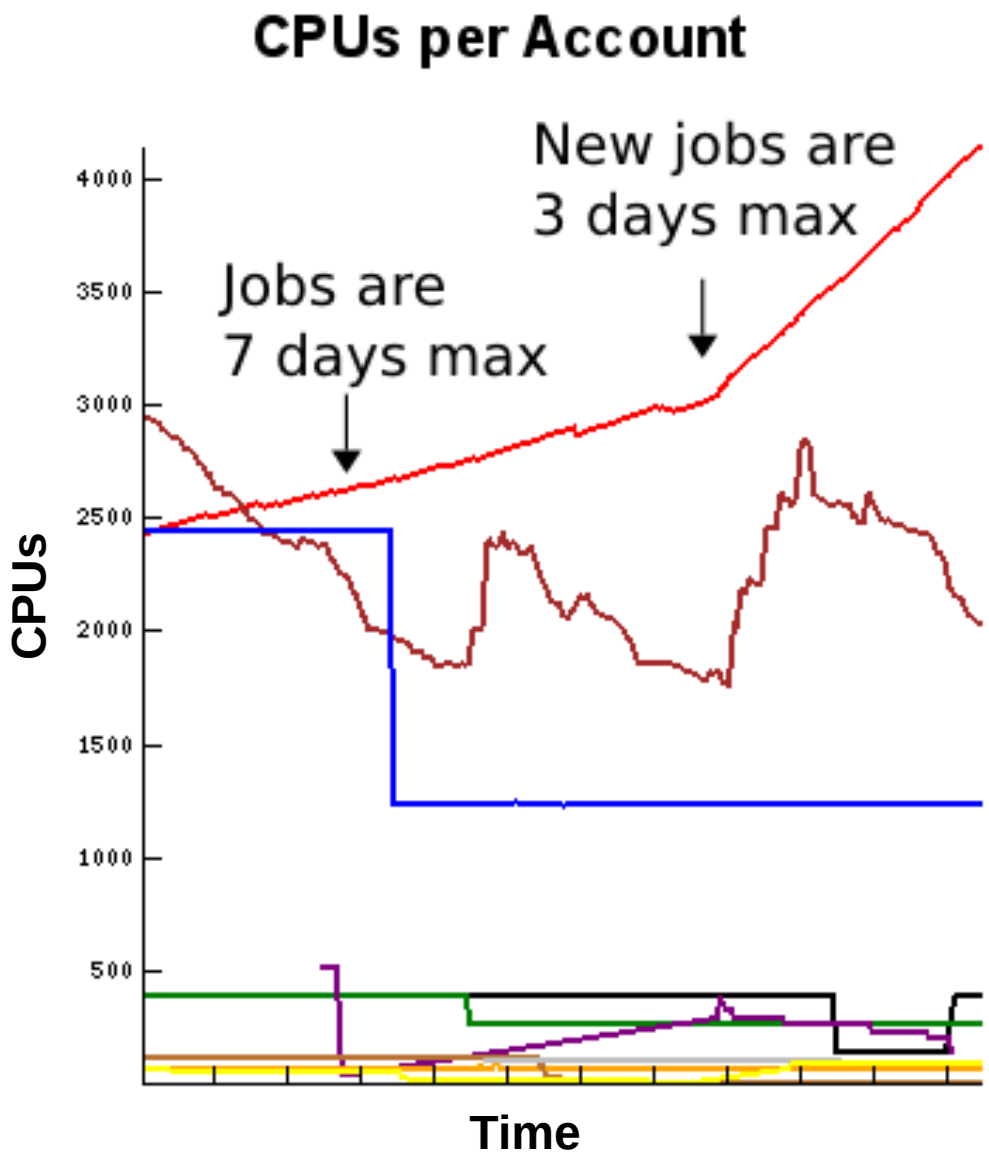
- 7 days on Ethernet cluster
- 3 days on everything else (IB, GPUs, fat nodes)
- Want to lower Ethernet cluster limit in future
- Installation of new clusters is a great time to transition to a lower walltime
  - Need 7 days? Use the old stuff
  - Only need 3 days? You can use both old and new!

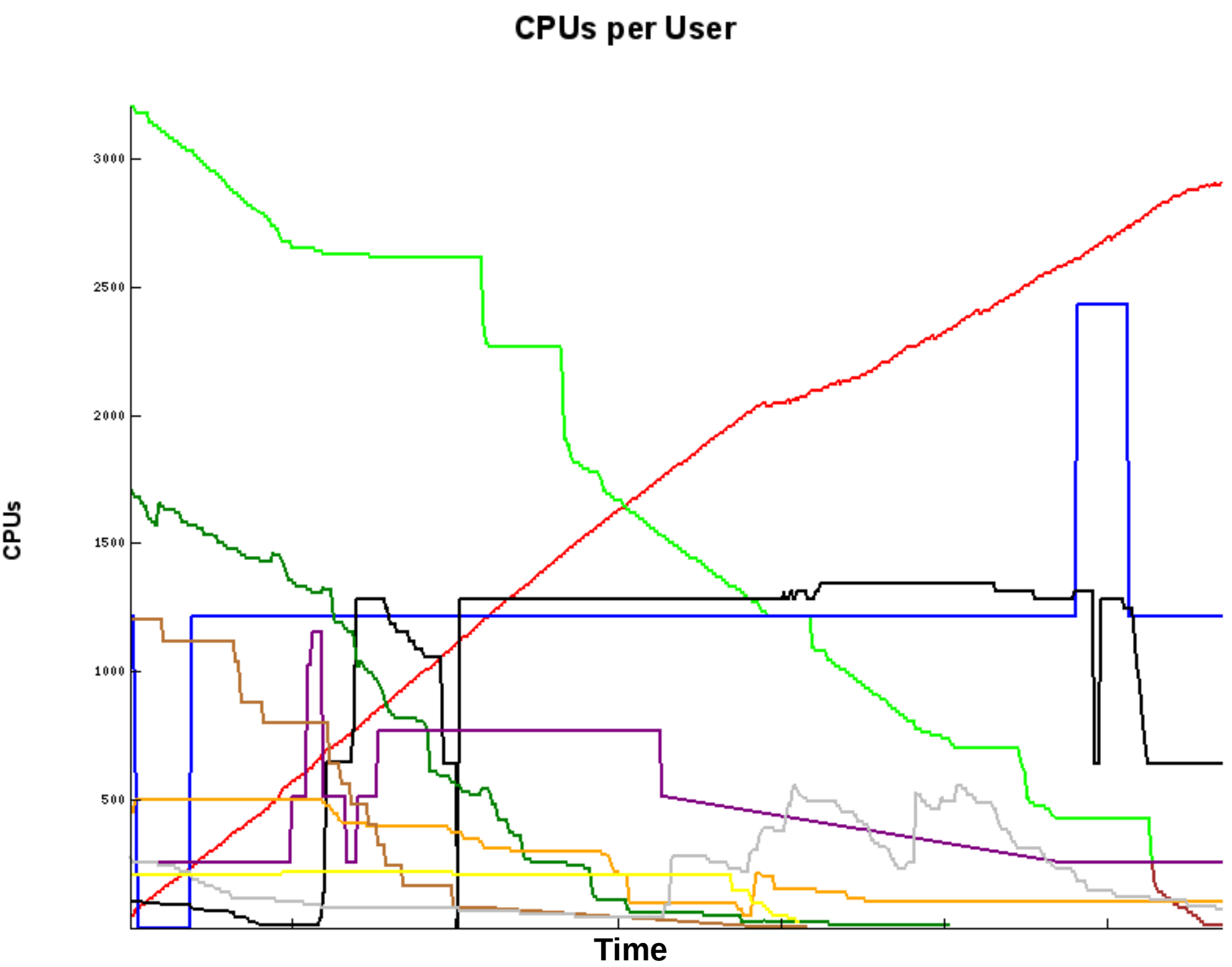
GrpCPURunMins

- Limit on cputime in use at one time:
  - Limit on *sum(cores allocated * remaining time)*
- We set it per-account
- Encourages shorter walltimes
  - The shorter your walltimes, the less you're affected
- Can stagger the start time of jobs
- Reduces average queue time for other accounts since nodes are more frequently freeing up (assuming lots of small jobs)

More info: http://tech.ryancox.net/2014/04/scheduler-limit-remaining-cputime-per.html

Someone lowered their walltimes...





Green and red are in same account. Red submitted jobs but green's jobs were fairly staggered due to GrpCPURunMins

GrpCPURunMins Simulator

- Available on github:
  - https://github.com/BYUHPC/GrpCPURunMins-Visualizer
- Compare effect of GrpCPURunMins with different limits, job sizes, and job walltimes
- Can currently “use” more resources than are available
  - We need to add a max core count (or GrpCPUs)

This graph simulates the interaction of job core counts, job walltimes, and the GrpCPURunMins limit. You can compare up to three different combinations by pressing the "+" button. An explanation is located below the graph.

Simulation #0

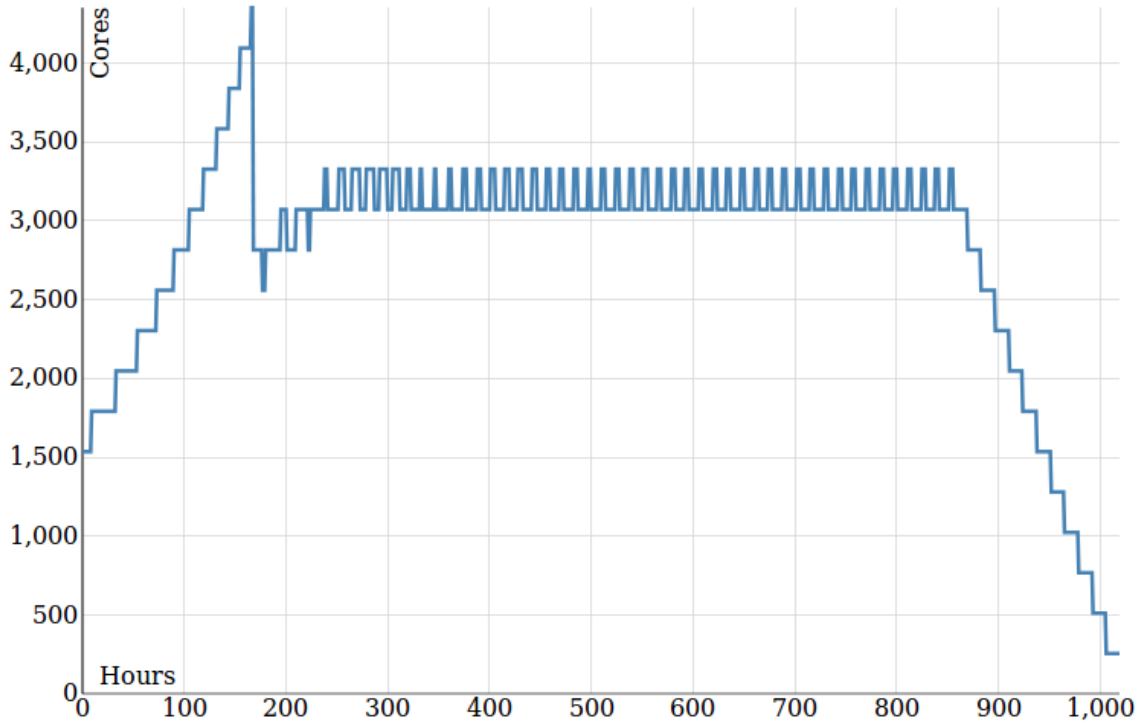
Job Walltime: Days ▼ 7

Cores per job: 256

GrpCPURunMins Limit ⓘ: 17280000

Update Chart +

Download Graph



The simulation assumes infinite resources and no contention from other users (i.e. your numbers may result in 100x the usage that is possible with the available hardware). There are four main features to note:

1. **The number of cores started at time 0** is the number of cores that start immediately before hitting the limit. It is equal to GrpCPURunMins / walltime_mins.
2. **The peak** is what happens when your initial jobs are all have a remaining time of a few seconds. Their cores*remaining_time number is now almost zero, so they are not replaced by many jobs once they complete.
3. **The plateau** is what happens when job start times are sufficiently staggered. The plateau is where your usage will stabilize over time.
4. **The tail off** shows how quickly your usage will drop when higher priority users submit jobs or you run out of queued jobs. This shows how quickly you can free up your resources. The steeper this slope across all users, the lower the average queue time will be.

This graph simulates the interaction of job core counts, job walltimes, and the GrpCPURunMins limit. You can compare up to three different combinations by pressing the "+" button. An explanation is located below the graph.

Simulation #0

Job Walltime: Days ▼ 7

Cores per job: 256

GrpCPURunMins Limit: 17280000

Update Chart

Simulation #1

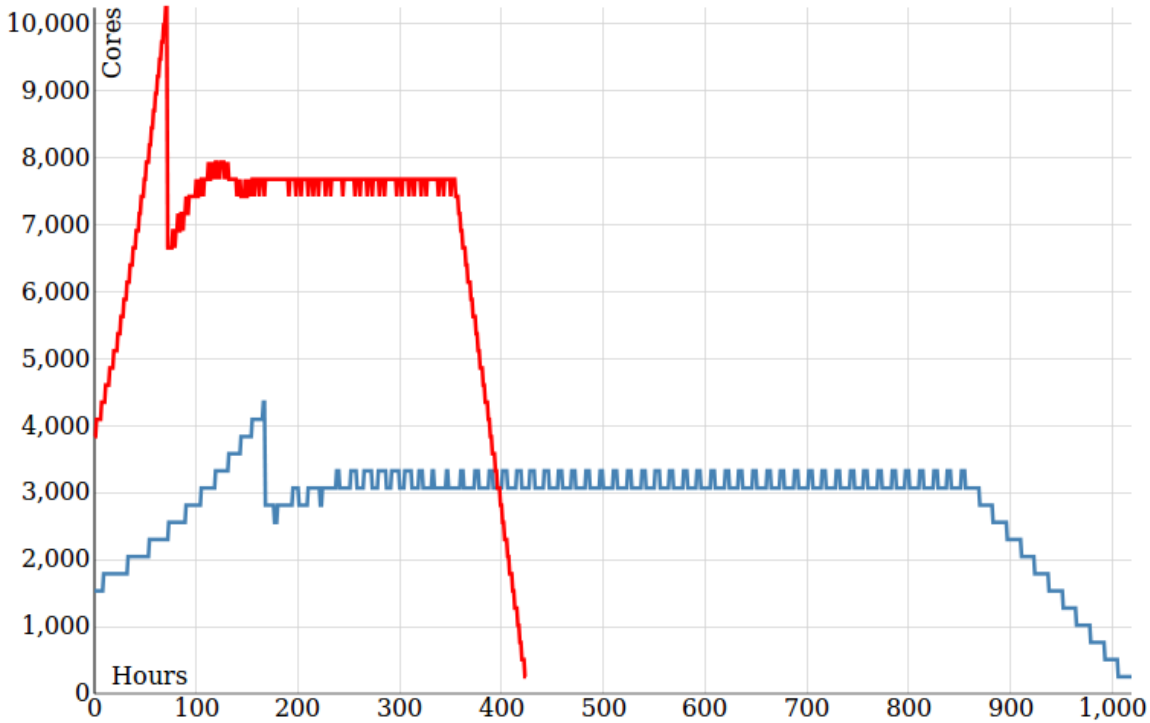
Job Walltime: Days ▼ 3

Cores per job: 256

GrpCPURunMins Limit: 17280000

Update Chart - +

Download Graph



The simulation assumes infinite resources and no contention from other users (i.e. your numbers may result in 100x the usage that is possible with the available hardware). There are four main features to note:

1. **The number of cores started at time 0** is the number of cores that start immediately before hitting the limit. It is equal to GrpCPURunMins / walltime_mins.
2. **The peak** is what happens when your initial jobs are all have a remaining time of a few seconds. Their cores*remaining_time number is now almost zero, so they are not replaced by many jobs once they complete.
3. **The plateau** is what happens when job start times are sufficiently staggered. The plateau is where your usage will stabilize over time.
4. **The tail off** shows how quickly your usage will drop when higher priority users submit jobs or you run out of queued jobs. This shows how quickly you can free up your resources. The steeper this slope across all users, the lower the average queue time will be.

This graph simulates the interaction of job core counts, job walltimes, and the GrpCPURunMins limit. You can compare up to three different combinations by pressing the "+" button. An explanation is located below the graph.

Simulation #0

Job Walltime	Days	7
Cores per job		256
GrpCPURunMins Limit		17280000

Update Chart

Simulation #1

Job Walltime	Days	3
Cores per job		256
GrpCPURunMins Limit		17280000

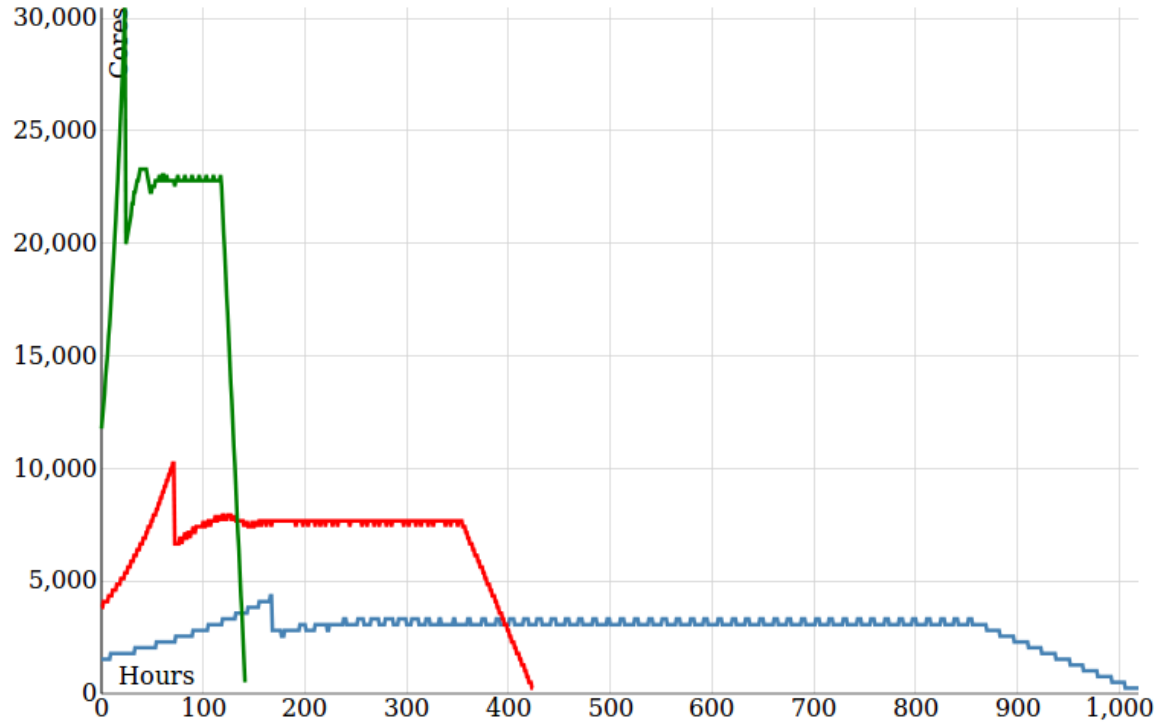
Update Chart

Simulation #2

Job Walltime	Days	1
Cores per job		256
GrpCPURunMins Limit		17280000

Update Chart -

Download Graph



The simulation assumes infinite resources and no contention from other users (i.e. your numbers may result in 100x the usage that is possible with the available hardware). There are four main features to note:

1. **The number of cores started at time 0** is the number of cores that start immediately before hitting the limit. It is equal to GrpCPURunMins / walltime_mins.
2. **The peak** is what happens when your initial jobs are all have a remaining time of a few seconds. Their cores*remaining_time number is now almost zero, so they are not replaced by many jobs once they complete.
3. **The plateau** is what happens when job start times are sufficiently staggered. The plateau is where your usage will stabilize over time.
4. **The tail off** shows how quickly your usage will drop when higher priority users submit jobs or you run out of queued jobs. This shows how quickly you can free up your resources. The steeper this slope across all users, the lower the average queue time will be.

This graph simulates the interaction of job core counts, job walltimes, and the GrpCPURunMins limit. You can compare up to three different combinations by pressing the "+" button. An explanation is located below the graph.

Simulation #0

Job Walltime	Hours ▼	72
Cores per job		16
GrpCPURunMins Limit ⓘ		2000000

Update Chart

Simulation #1

Job Walltime	Hours ▼	72
Cores per job		16
GrpCPURunMins Limit ⓘ		4000000

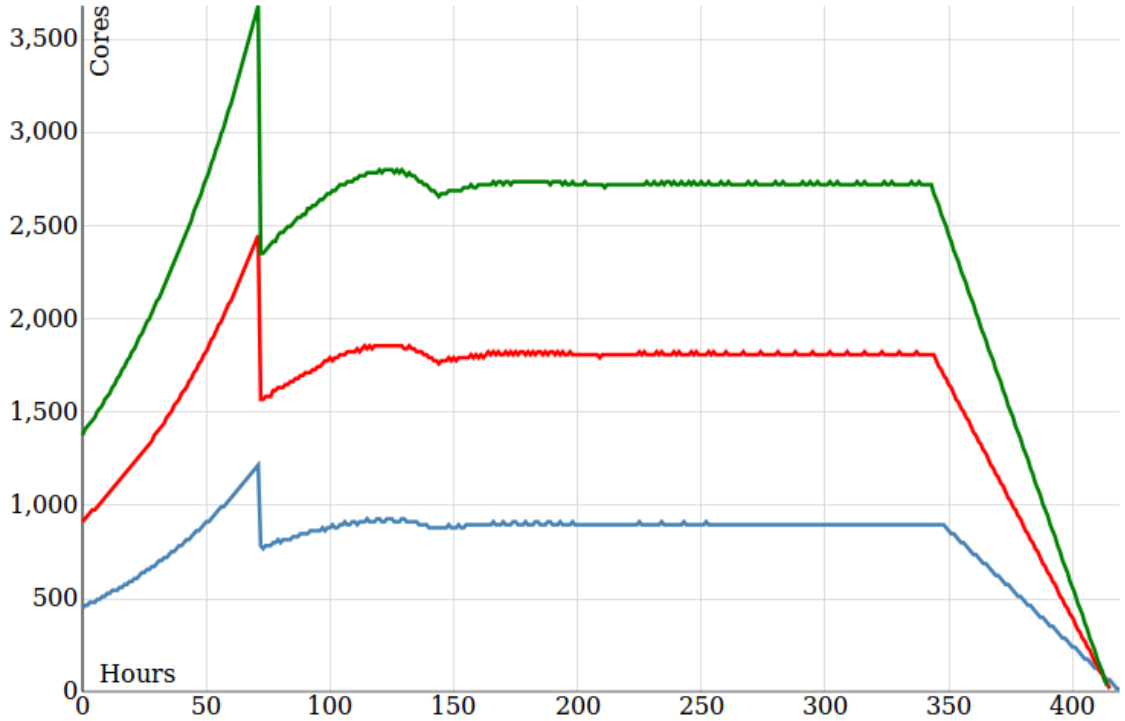
Update Chart

Simulation #2

Job Walltime	Hours ▼	72
Cores per job		16
GrpCPURunMins Limit ⓘ		6000000

Update Chart -

Download Graph



The simulation assumes infinite resources and no contention from other users (i.e. your numbers may result in 100x the usage that is possible with the available hardware). There are four main features to note:

1. **The number of cores started at time 0** is the number of cores that start immediately before hitting the limit. It is equal to GrpCPURunMins / walltime_mins.
2. **The peak** is what happens when your initial jobs are all have a remaining time of a few seconds. Their cores*remaining_time number is now almost zero, so they are not replaced by many jobs once they complete.
3. **The plateau** is what happens when job start times are sufficiently staggered. The plateau is where your usage will stabilize over time.
4. **The tail off** shows how quickly your usage will drop when higher priority users submit jobs or you run out of queued jobs. This shows how quickly you can free up your resources. The steeper this slope across all users, the lower the average queue time will be.

Account Coordinator

- Each PI has a Slurm account
- Each PI is the account coordinator of that account
- Can kill/hold user jobs
- Can set shares for Fairshare and limits
- Can add/remove QOS for users for access to private resources (AllowQOS on partition)
- Reduces admin overhead
- Some GUI tools to assist
  - https://github.com/BYUHPC/slurm-fairshare-pie
  - https://github.com/BYUHPC/slurm-limits-web

Slurm Fairshare: My Account

Edit the table below to see how changing the Fairshares for a group member will impact the group. More information about Fairshare is available under *Adjust the priority of individual users* in Extras for Faculty. See also: Slurm Limits.

To save these changes, log into ssh.fsl.byu.edu and paste the commands generated below into the terminal.

```
# command format below for reference
    sacctmgr -i modify user $user set fairshare=$shares
```

Users	Shares (editable)	% of Total
mreynolds	1024	20.00%
wash	1024	20.00%
jayne	1024	20.00%
zoe	1024	20.00%
kaylee	1024	20.00%

Hover over chart for details

Slurm Fairshare: My Account

Edit the table below to see how changing the Fairshares for a group member will impact the group. More information about Fairshare is available under *Adjust the priority of individual users* in Extras for Faculty. See also: Slurm Limits.

To save these changes, log into ssh.fsl.byu.edu and paste the commands generated below into the terminal.

```
sacctmgr -i modify user jayne set fairshare=0
```

Users	Shares (editable)	% of Total
mreynolds	1024	25.00%
wash	1024	25.00%
zoe	1024	25.00%
kaylee	1024	25.00%
jayne	0	0.00%

Hover over chart for details

Slurm Fairshare: My Account

Edit the table below to see how changing the Fairshares for a group member will impact the group. More information about Fairshare is available under *Adjust the priority of individual users* in Extras for Faculty. See also: Slurm Limits.

To save these changes, log into ssh.fsl.byu.edu and paste the commands generated below into the terminal.

```
sacctmgr -i modify user zoe set fairshare=2048
sacctmgr -i modify user jayne set fairshare=0
```

Users	Shares (editable)	% of Total
zoe	2048	40.00%
mreynolds	1024	20.00%
wash	1024	20.00%
kaylee	1024	20.00%
jayne	0	0.00%

Hover over chart for details

Slurm Limits: My Account

Edit the table below to generate commands that can be used to modify your group members' limits. See also: Slurm Shares.

To save these changes, log into ssh.fsl.byu.edu and paste the commands generated below into the terminal.

```
sacctmgr -i modify user mreynolds set MaxWall=30-12:30:00
```

User	Jobs	CPU cores	Nodes	Job Wall Time*	CPU-Minutes
mreynolds				30-12:30:00	
wash					
jayne					
zoe					
kaylee					

Slurm Limits: My Account

Edit the table below to generate commands that can be used to modify your group members' limits. See also: Slurm Shares.

To save these changes, log into ssh.fsl.byu.edu and paste the commands generated below into the terminal.

```
sacctmgr -i modify user mreynolds set MaxWall=30-12:30:00
sacctmgr -i modify user zoe set GrpNodes=4
sacctmgr -i modify user kaylee set GrpJobs=1
```

User	Jobs	CPU cores	Nodes	Job Wall Time*	CPU-Minutes
mreynolds				30-12:30:00	
wash					
jayne					
zoe			4		
kaylee	1				

Fairshare

- Fair Tree fairshare algorithm
- Works well for us
- How many of you are using or planning to switch to Fair Tree?

Questions?

http://github.com/BYUHPC