Brigham Young University

Fulton Supercomputing Lab

Ryan Cox

Slurm User Group – September 16, 2015 – Washington, D.C.

Open Source Code

- I'll reference several codes we have open sourced

http://github.com/BYUHPC

History

- Switched to Slurm 2.5 from Moab/Torque in January 2013
- SchedMD support
- We're crazy
  - If you remember my SLUG 2013 presentation, we were the crazy people who did a split-brain, rolling transition to Slurm without telling our users
    - Almost flawless

About

- Resources are free to use
- Each P.I. has an account, all accounts are treated equally
- Have about 6 partitions, access control with job submit plugin and AllowQOS
- Slurm 14.11.8
- Will update to 15.08 in a week or two
- Excited about TRES

User Training

- We offer 1-1 training to each new user but they almost never take us up on it due to training materials
- Several short training videos on YouTube channel
  - http://youtube.com/BYUSupercomputing
  - Intro to Slurm Tools
- Unix Tutorial
  - https://fsl.byu.edu/documentation/unix-tutorial/
- Script Generator
  - https://github.com/BYUHPC/BYUJobScriptGenerator

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
```

The latest version of the *Brigham Young University Job Script Generator* is available on github

Feature-Based Scheduling

- Users request features (-C avx) rather than partitions
  - avx, avx2, fma, ib
- Must request memory and timelimit for anything beyond HelloWorld
  - Defaults: 1 core, 512 MB memory, 30 minutes (sufficient even for python HelloWorld.py)
- Lua job submit plugin removes non-usable 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 of BYU's job submit plugin:
  - all_partitions plugin sets a job's partition to all partitions, unless user requested a specific one(s)
  - Empty “blankpart” allows easy detection of whether they requested a specific partition or not (users never request “blankpart” partition)
  - Lua script checks if job can run in partition. If not, remove from list
    - Based on memory, CPU count, GRES, timelimit, etc.
  - Assigns to particular QOS as applicable
- Business logic is worth the few hours of learning
  - Really, it's just a few hours to learn Lua and the job submit plugin

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
- *Survey...*

Fairshare

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

For more information, see SLUG 2014 presentation: http://slurm.schedmd.com/SUG14/fair_tree.pdf

Account Coordinator

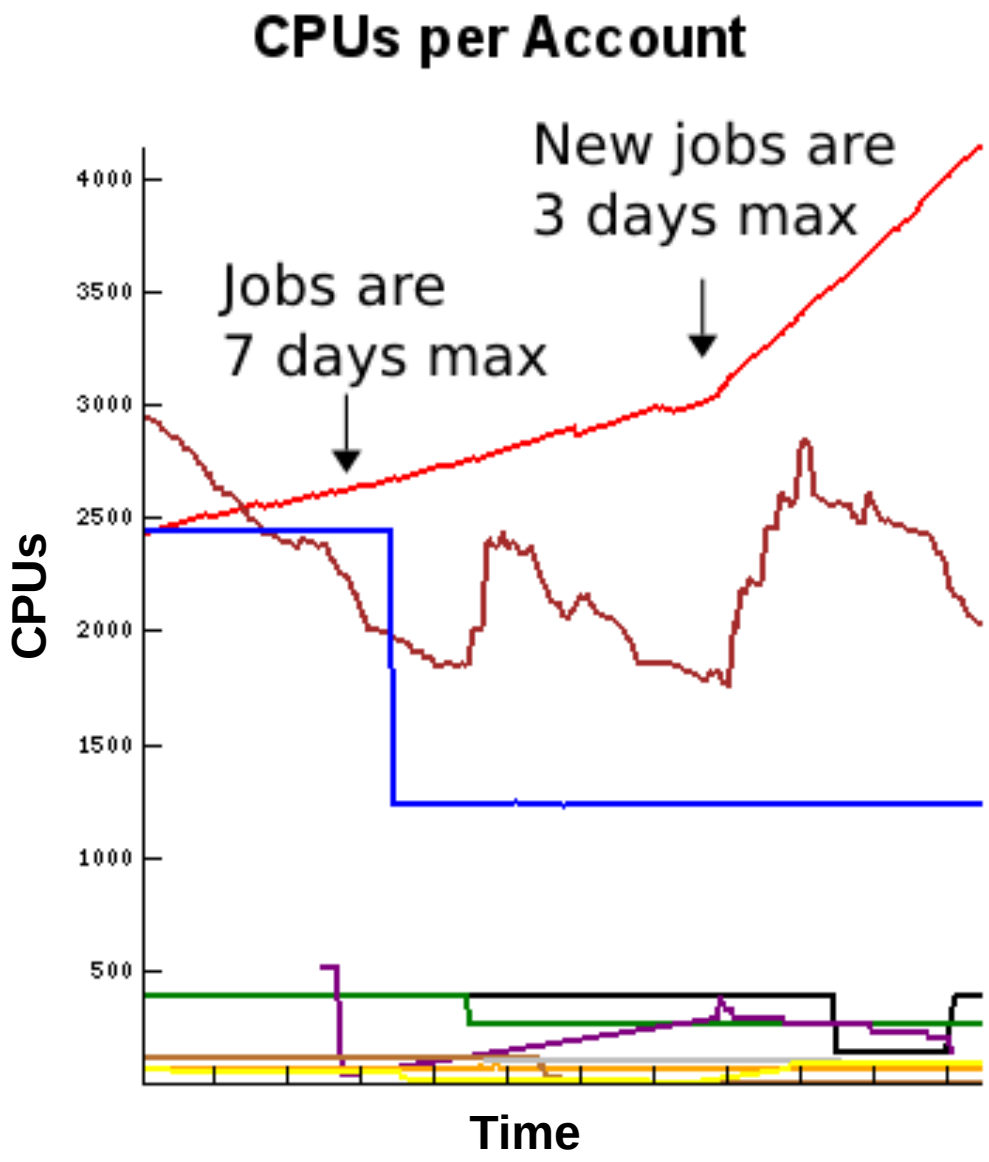
- Each PI has a Slurm account
- Each PI is the account coordinator of that account
  - sacctmgr add coordinator Names=$username Accounts=$account
- Can kill/hold user jobs
- Can set shares for Fairshare and limits
- Can add/remove QOS for users for access to private resources (AllowQOS on partition)
- Reduces admin overhead
- Some GUI tools to assist

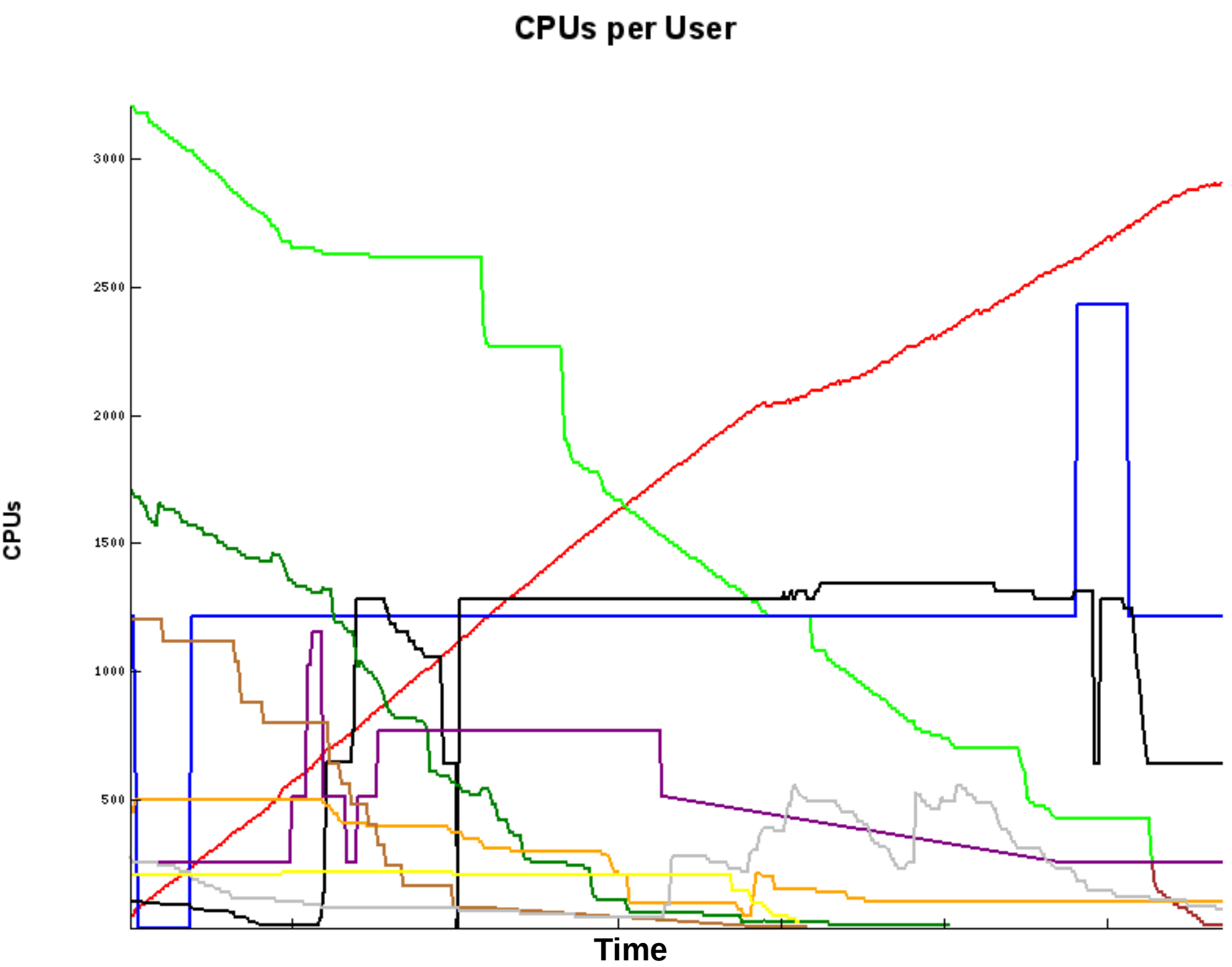
GrpCPURunMins

- 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

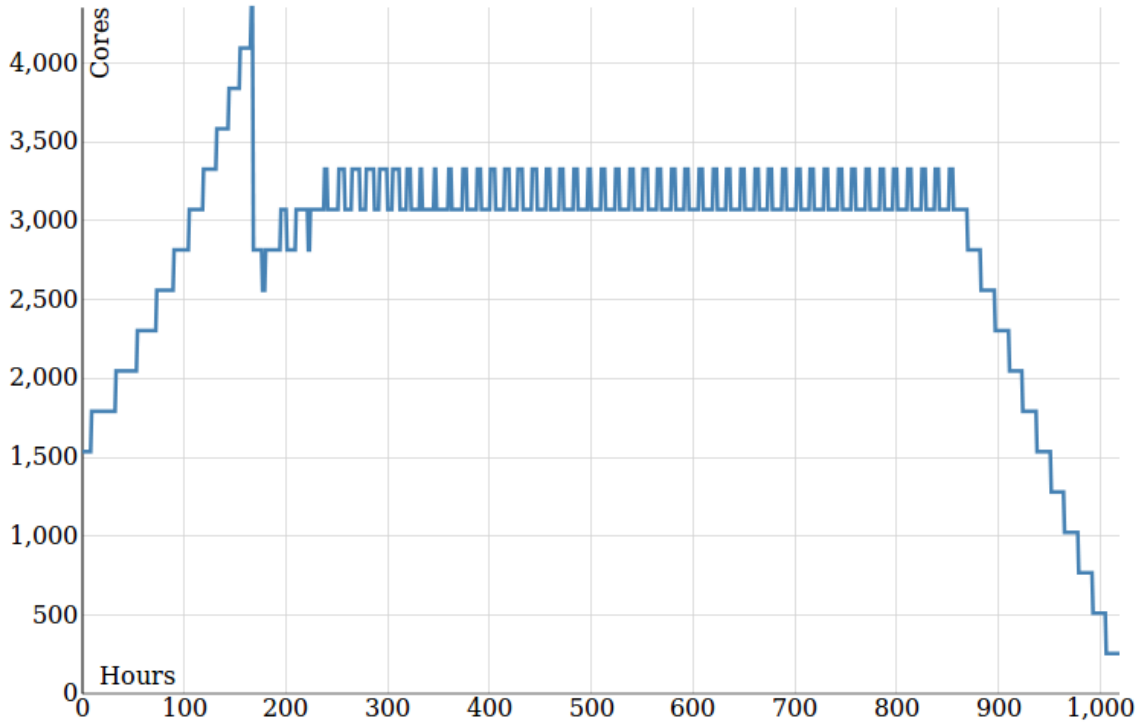
Job Walltime: Days ▼ 7

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

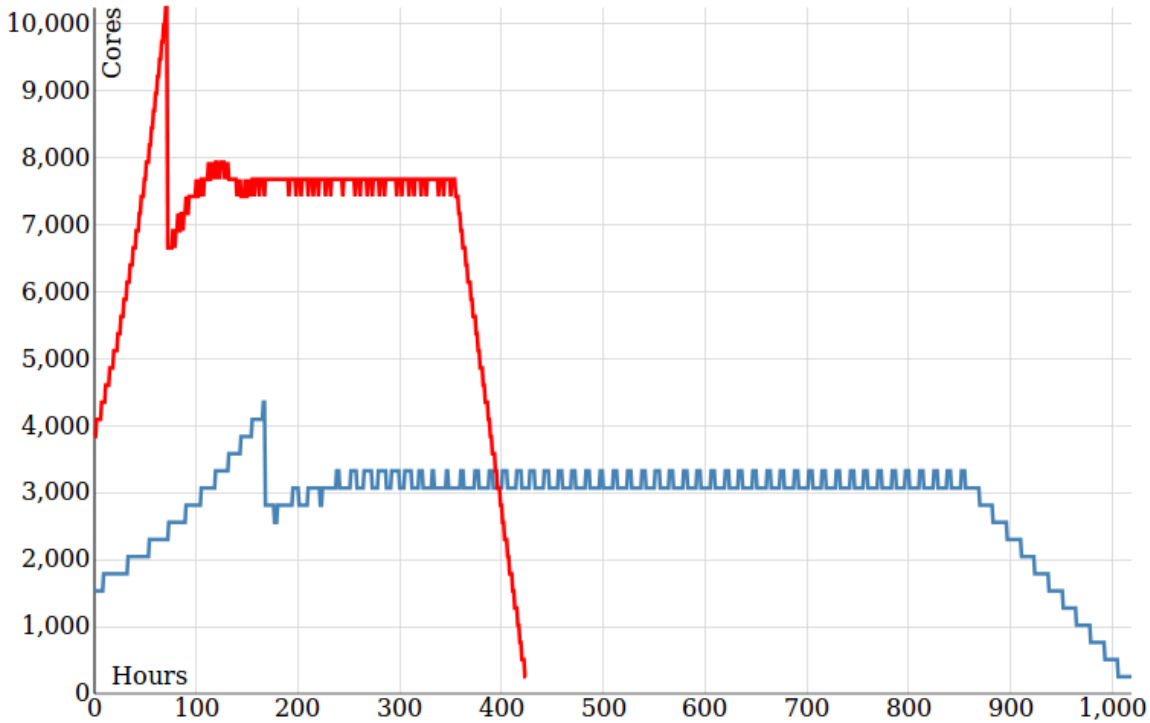
Job Walltime: Days ▼ 3

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

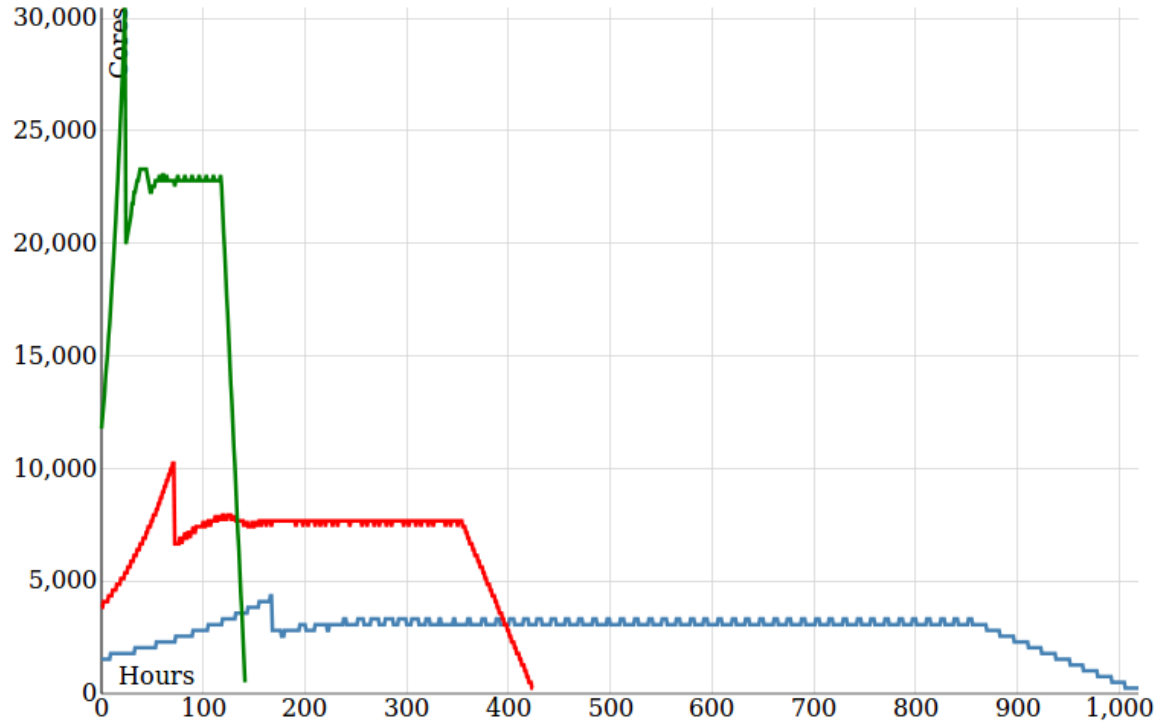
Update Chart

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

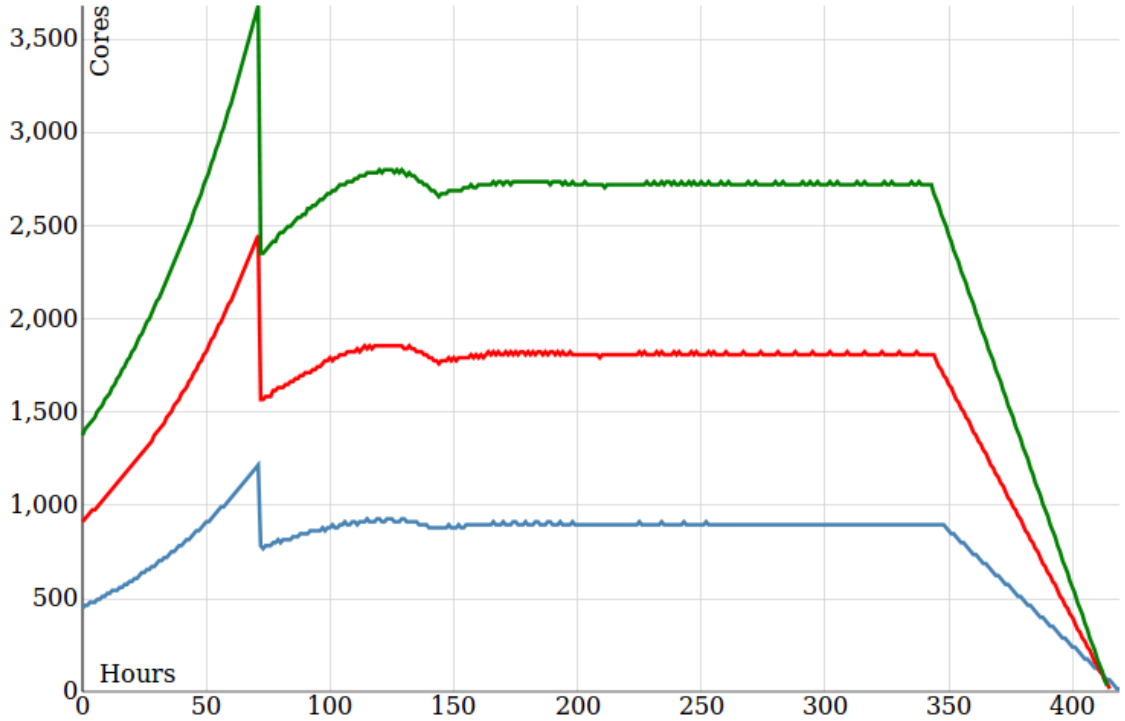
Update Chart

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.

Slurm Shares Visualizer

https://github.com/BYUHPC/slurm-fairshare-pie

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

https://github.com/BYUHPC/slurm-limits-web

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				

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: Not all code integrates with slurm

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
- Can mess up sshrc in a way that breaks X11 and other things
- Users can override $SLURM_JOB_ID
- Messy but would likely work if PermitUserRC=no *and* if the user never runs exec{l,p}e functions or changes $SLURM_JOB_ID

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 depending on settings 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 instance results in a new step. Results in many db entries

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
```

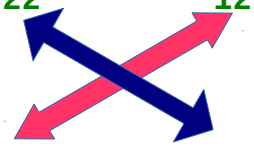
- Last line is incoming to sshd
- netstat -np # source of ssh connection

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

Solution

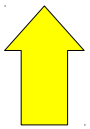
- Destination:

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



- Source:

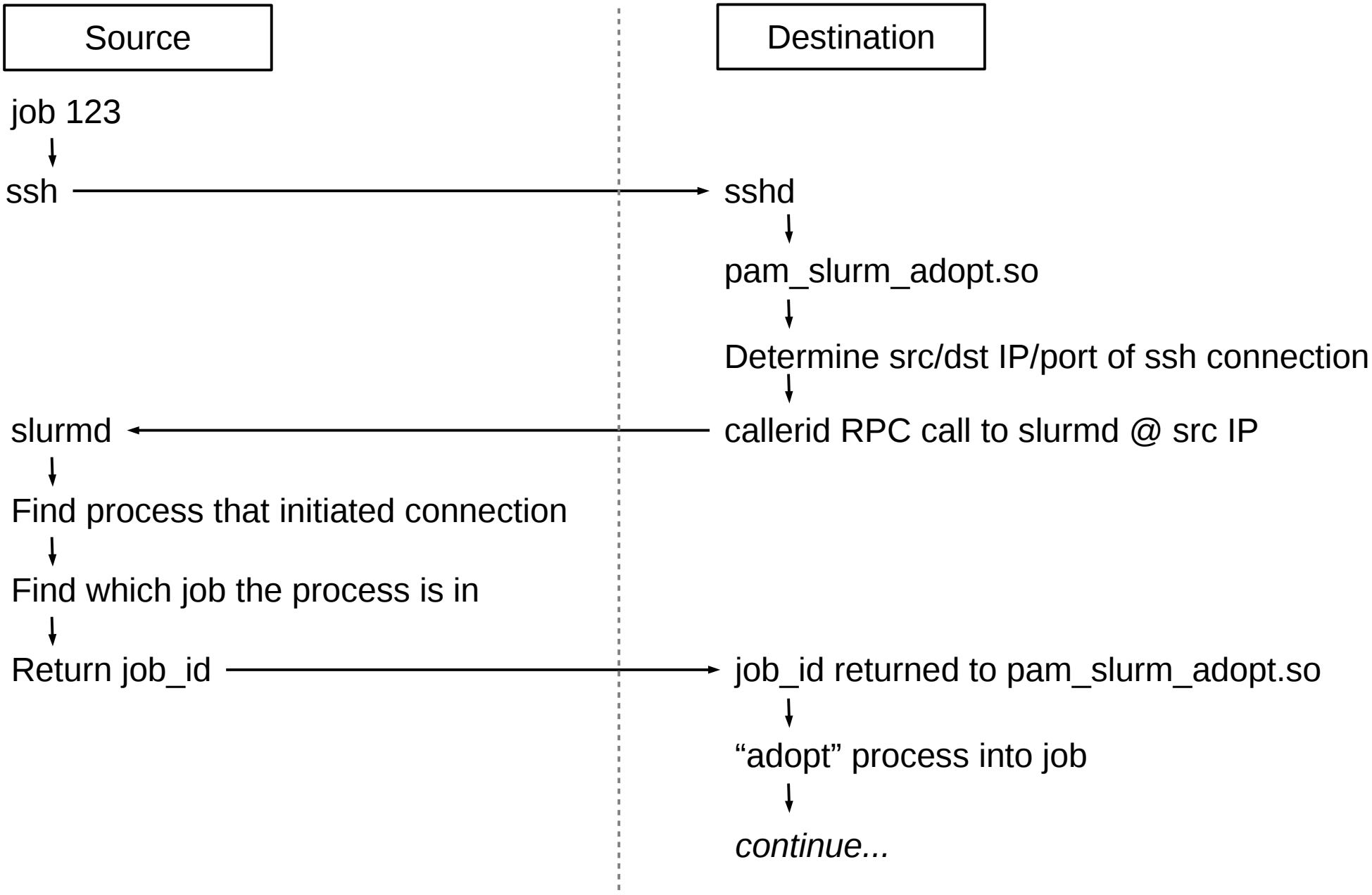
```
tcp        0      0 12.3.456.78:55777       10.1.123.106:22         ESTABLISHED 23181/ssh
```



- 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 call
  - RPC call itself is OS-agnostic
  - 5 fields: src and dst IP and port, IP version (4, 6)
- Reads /proc/net/tcp{,6}
- Response contains nodename and job_id


Source
Destination
job 123
ssh
sshd
pam_slurm_adopt.so
Determine src/dst IP/port of ssh connection
slurmd
callerid RPC call 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`

Process Adoption

- Jobs will be “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 (Thanks Moe!)
  - Enabled with PrologFlags=contain
  - The adopted process will go into this step
- Unfortunately... I have not yet finished the adoption code
  - Stubs are in place in pam_slurm_adopt.c (as of 15.08)

Job ID Indeterminate?

- What happens if the job ID can't be determined?
- Configurable in PAM module
- If user has no job on node, deny
- If user has single job, assume it's that one
- If user has multiple jobs, pick a random one (they somehow evaded our detection or came in from a login node... not my problem)
  - Alternatively, place them in generic /uid_$uid cgroup

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 call itself is OS-agnostic
- IPv6 is ready but Slurm RPC calls are IPv4 only

Status

- 15.08.0 has all necessary code except the code to adopt processes
- pam_slurm_adopt can successfully determine job_id but can't yet adopt it
  - I plan to finish pam_slurm_adopt within a few months
  - If you want to work on it before then, please do but let me know so I don't duplicate the effort.

Wishlist

- Proactive “you're using it wrong” tools for users
  - Job “efficiency” stats in job output? Actual/allocated cputime, memory, walltime, etc. Include it in job emails?
  - “GoodCitizenship” priority factors
    - TRES efficiency (CPU, memory, GPU, walltime, etc) of job allocation as a whole (don't care about steps)
    - Walltime nice but not super important; other efficiencies are really important
- More verbose error messages (sounds like pending reasons are improved in 15.08)
  - Users have a hard time with *good* explanations; terse messages are even harder
  - Maybe “scontrol show job”, et al can have a verbose explanation or “Resources”, etc
- Non-idle node state for nodes that are reserved for high priority jobs (No, userbob, those nodes are not idle; they will soon be part of important_user's job)
- When job dies due to OOM, set RSS accounting fields equal to the allocated amount (they tried to exceed it, right?) and set a job failure state “MEM_LIMIT_EXCEEDED” or similar

Questions?

http://github.com/BYUHPC