

Slurm Training Documentation



Contents

Accessing the Slurm Training Learning Environment	16
The Learning Environment	16
Attaching to the AWS Instance – Linux/Mac Method	16
Attaching to the AWS Instance – Windows 10 Method	18
Attaching to the AWS Instance – Windows 7 (PuTTY) Method	19
The /lab_scripts folder and Files.....	22
Building The Learning Environment for Yourselves	23
Submitting and Running Jobs Discussion.....	27
Architecture	27
Summary of Slurm Commands	28
Examples.....	29
Submitting Batch Jobs using the <code>sbatch</code> Command	32
Slurm Completion Support	42
Submitting Jobs using the <code>srun</code> Command.....	42
Troubleshooting jobs with <code>srun</code>	43
Best Practices, Large Job Counts	43
MPI.....	43
Job Submission Exercises.....	45
Exercise 1: Submit a job with <code>srun</code>	45
Exercise 2: Submit an interactive job with <code>salloc</code>	47
Exercise 3: Using <code>sbatch</code> , create a resource allocation and submit a job script to it.....	51
Exercise 4: Using <code>salloc</code> , create a resource allocation and submit a command to it.....	52
Submit a job using <code>salloc</code> and run <code>whereami</code> in the allocation.....	55
Exercise 5: Finding the PIDs of running jobs.....	56
Create job script that will run multiple programs on a compute node	57
Get the pids from the running job	58
Exercise 6 - Task Allocation, Placement, and Binding.....	59
Exercise 7: Use Bash completion for Slurm to Submit Jobs	63
Cleanup	67
Job Arrays Discussion.....	68
Job ID and Environment Variables.....	68
File Names	69
<code>scancel</code> Command Usage	70

squeue Command Use	70
scontrol Command Usage.....	71
Job Dependencies.....	72
Other Command Use	73
System Administration	73
Job Array Exercises	74
Exercise 1: Submit a job array of 30 sub-jobs, taking input files as data.....	74
Cleanup	77
Slurm RESTful API Discussion.....	78
Guiding Principle of REST (from the rest website).....	78
Resource:	78
Resource Methods.....	79
What is OpenAPI (aka Swagger)	79
Slurm API Methods	79
The Slurm API Models.....	80
slurmrestd.....	83
Slurmrestd Design.....	85
Examples.....	85
Security	86
Authentication with REST API.....	87
IPv4 and IPv6 Support.....	88
Example Queries.....	88
Additional Resources and Links	92
slurmrestd Lab Exercises	93
Exercise 1: OpenAPI Generation.....	93
Exercise 2: Direct Slurm REST query using JWT:	94
Exercise 3: Direct Slurm REST job submission	98
Exercise 4: Submitting a Proxy-authenticated job through slurmrestd.....	102
Exercise 5: Cancel Job using curl through slurmrestd and using auth	103
Cleanup	104
Slurm Job Management.....	105
scancel	105
Description.....	105
scancel Usage.....	105

ENVIRONMENT VARIABLES.....	107
EXAMPLES.....	107
scontrol update	108
scontrol update SPECIFICATION	108
SPECIFICATIONS FOR UPDATE COMMAND, JOBS.....	108
SPECIFICATIONS FOR UPDATE COMMAND, STEPS	115
SPECIFICATIONS FOR UPDATE COMMAND, NODES	115
scontrol show.....	117
sinfo.....	118
svview.....	123
API Changes for Slurm V23.02	124
Managing Jobs Exercises	125
Exercise 1: Modify a job using scontrol	125
Exercise 2: Retrieving job information using scontrol	125
Cleanup.....	127
Slurm Accounting and Reporting Discussion	128
sreport	128
sstat	130
sacct (end-user) and sacctmgr (administrator)	130
Accounting and Reporting Exercises.....	132
Exercise 1: Use the sstat command to view job information stored in the database:.....	132
Exercise 2: Use the sacct command to view accounting database information:.....	132
Cleanup.....	133
Topology and Application Layout Discussion	134
Three-dimension Topology.....	134
Hierarchical Networks	134
User Options	137
Environment Variables	137
Topology and Application Layout Exercises.....	138
Exercise 1: Create a topology.conf file.....	138
Exercise 2: Submit a job to a node on a switch	138
Exercise 3: Using cpu-bind to Place Tasks on Allocated Resources	139
Exercise 4: Using cpu-bind to Place Tasks on a specific Resource	140
Cleanup.....	141

Cgroups Discussion	142
Use of Cgroups in Slurm	142
Slurm Cgroups Configuration Overview	142
Currently Available Cgroup Plugins	142
PROCTRACK/CGROUP PLUGIN.....	142
TASK/CGROUP PLUGIN	143
JOBACCT_GATHER/CGROUP PLUGIN	143
Use of Cgroups for Resource Specialization	143
Organization of Slurm Cgroups.....	144
CGROUP V2 Plugin	145
Following cgroup v2 rules.....	145
TOP-DOWN CONSTRAINT	145
NO INTERNAL PROCESS CONSTRAINT	145
Following systemd rules	145
THE REAL PROBLEM: SYSTEMD + RESTARTING SLURMD.....	145
CONSEQUENCES OF NOT FOLLOWING SYSTEMD RULES	147
WHAT HAPPENS WITH LINUX DISTROS WITHOUT SYSTEMD?.....	147
cgroup/v2 overview.....	147
SLURMD STARTUP	147
SLURMD RESTART.....	148
SLURMSTEPD START	148
TERMINATION AND CLEANUP	149
HIERARCHY OVERVIEW	149
Working at the task level	150
The eBPF based devices controller	150
Running different nodes with different cgroup versions.....	151
Configuration.....	151
CGROUP PLUGIN.....	151
DEVELOPER OPTIONS	151
IGNORED PARAMETERS	152
Cgroup Exercises.....	153
Exercise 1: Restricting memory Cgroup Example	153
Edit the slurm.conf:.....	153
Cleanup	155

Exercise 2: Restricting memory: Another way Example	155
Cleanup.....	156
Quality of Service Discussion	157
Job Scheduling Priority	157
Job Preemption Introduction.....	157
Job Limits	157
Partition QOS	157
Floating Partition Example.....	158
Partition vs Job QOS	158
Other QOS Options	158
Configuration	160
QOS Examples.....	160
Preemption	162
Preemption configuration	162
Preemption Design and Operation	164
Limitations of Preemption During Backfill Scheduling.....	166
A Simple Example	166
Another Example	167
Quality of Service (QOS) Lab Exercises	169
Lab Setup-DO THIS BEFORE UNDERTAKING THE EXERCISES	169
Exercise 1: Configure for QOS-based job preemption	170
Cleanup.....	172
The Infrastructure for Building Slurm	173
Database.....	173
Slurm Versions	173
Building Slurm.....	174
Configuring and Starting Slurm.....	174
Building and Installing Slurm Exercises.....	176
Configure Method	176
RPM Method.....	179
Building The Learning Environment for Yourselves	184
You can now use the DSO environment to do the other labs	187
Node and Partition Configuration Discussion.....	188
Nodes.....	188

Sample Node Configuration.....	189
The DownNodes Parameter	189
Partitions	190
Partition-Based Preemption	193
Configuration for Preemption	194
Preemption Design and Operation	197
Limitations of Preemption During Backfill Scheduling.....	198
Slurm Dynamic Nodes.....	198
Dynamic Node Communications	198
Slurm Configuration Related to Dynamic Nodes	198
PARTITION ASSIGNMENT for Dynamic Nodes	199
Creating Nodes	199
Deleting Nodes	199
Node and Partition Exercises.....	200
Exercise 1: Group nodes into partitions	200
Cleanup.....	202
Exercise 2: Configure Slurm for Partition-Based Preemption.....	202
Cleanup	207
Daemons and Logging Discussion.....	208
Daemons.....	208
slurmctld Daemon	209
slurmdbd Daemon	209
slurmd.....	209
slurmstepd.....	210
Linux Job Launch Sequence	210
Log Files	211
Daemons and Logging Exercises.....	213
Exercise 1: Change the control daemon logging level	213
Cleanup.....	214
Slurm Scheduling Discussion	215
Scheduling Configuration.....	215
Main Scheduler	216
Backfill Scheduling	216
Consumable Resources.....	217

Using the Consumable Resource Allocation Plugin: select/cons_tres.....	218
General Comments	219
Examples of CR_Memory, CR_Socket_Memory, and CR_CPU_Memory type consumable resources	219
Example of Node Allocations Using Consumable Resource Plugin.....	220
Using Slurm's Default Node Allocation (Non-shared Mode)	221
Using a Processor Consumable Resource Approach	221
Generic Resource (GRES) Scheduling.....	223
GRES Configuration.....	223
Running GRES Jobs.....	224
GPU Management	225
MPS Management	226
MIC Management.....	228
CPU Management User and Administrator Guide.....	228
CPU Management Steps performed by Slurm.....	228
STEP 1: SELECTION OF NODES	229
STEP 2: ALLOCATION OF CPUS FROM THE SELECTED NODES.....	231
STEP 3: DISTRIBUTION OF TASKS TO THE SELECTED NODES	235
STEP 4: OPTIONAL DISTRIBUTION AND BINDING OF TASKS TO CPUS WITHIN A NODE	236
Additional Notes on CPU Management Steps	237
Getting Information about CPU usage by Jobs/Steps/Tasks	237
A NOTE ON CPU NUMBERING	238
CPU Management Examples.....	238
EXAMPLE NODE AND PARTITION CONFIGURATION	238
EXAMPLE 1: ALLOCATION OF WHOLE NODES	239
EXAMPLE 2: SIMPLE ALLOCATION OF CORES AS CONSUMABLE RESOURCES	240
EXAMPLE 3: CONSUMABLE RESOURCES WITH BALANCED ALLOCATION ACROSS NODES.....	240
EXAMPLE 4: CONSUMABLE RESOURCES WITH MINIMIZATION OF RESOURCE FRAGMENTATION	241
EXAMPLE 5: CONSUMABLE RESOURCES WITH CYCLIC DISTRIBUTION OF TASKS TO NODES	242
EXAMPLE 6: CONSUMABLE RESOURCES WITH DEFAULT ALLOCATION AND PLANE DISTRIBUTION OF TASKS TO NODES.....	242
EXAMPLE 7: CONSUMABLE RESOURCES WITH OVERCOMMITMENT OF CPUS TO TASKS.....	243
EXAMPLE 8: CONSUMABLE RESOURCES WITH RESOURCE SHARING BETWEEN JOBS	244
EXAMPLE 9: CONSUMABLE RESOURCES ON MULTITHREADED NODE, ALLOCATING ONLY ONE THREAD PER CORE	245

EXAMPLE 10: CONSUMABLE RESOURCES WITH TASK AFFINITY AND CORE BINDING	246
EXAMPLE 11: CONSUMABLE RESOURCES WITH TASK AFFINITY AND SOCKET BINDING, CASE 1	247
EXAMPLE 12: CONSUMABLE RESOURCES WITH TASK AFFINITY AND SOCKET BINDING, CASE 2	248
EXAMPLE 13: CONSUMABLE RESOURCES WITH TASK AFFINITY AND SOCKET BINDING, CASE 3	249
EXAMPLE 14: CONSUMABLE RESOURCES WITH TASK AFFINITY AND CUSTOMIZED ALLOCATION AND DISTRIBUTION	250
EXAMPLE 15: CONSUMABLE RESOURCES WITH TASK AFFINITY TO OPTIMIZE THE PERFORMANCE OF A MULTI-TASK, MULTI-THREAD JOB	251
EXAMPLE 16: CONSUMABLE RESOURCES WITH TASK CGROUP	252
SlurmctldParameters	254
scontrol reconfigure	255
Scheduling Exercises	256
Exercise 1: Modify the Backfill Scheduler Policy by changing the default depth:	256
Cleanup	258
Exercise 2: Submitting jobs to GPUs	259
Cleanup	261
Slurm Job Prioritization Discussion	262
Job Prioritization Factors	262
Age Factor	263
Job Size Factor	263
Partition Factor	264
Quality of Service (QOS) Factor	264
TRES Factors	264
Fair-share Factor	264
The sprio Utility	265
Configuration of Priority	265
Job Priority Exercises	267
Exercise 1: Assign priority by QOS	267
Configure the slurm.conf	267
Add accounts and assign priority to them	268
Submit workload to see the priority	270
Cleanup	271
Fairshare Discussion	272
NORMALIZED SHARES	273

NORMALIZED USAGE	274
SIMPLIFIED FAIR-SHARE FORMULA	275
THE FAIR-SHARE FACTOR UNDER AN ACCOUNT HIERARCHY	275
THE SLURM FAIR-SHARE FORMULA.....	276
FAIRSHARE=PARENT	277
EXAMPLE.....	277
The <code>sprio</code> utility	279
Configuration	280
PriorityType	280
PriorityDecayHalfLife	280
PriorityCalcPeriod	280
PriorityUsageResetPeriod	280
PriorityFavorSmall	280
PriorityMaxAge	280
PriorityWeightAge	280
PriorityWeightFairshare.....	280
PriorityWeightJobSize.....	281
PriorityWeightPartition	281
PriorityWeightQOS	281
PriorityWeightTRES.....	281
Configuration Example	281
Fairshare Exercises	283
Exercise 1: Create a fairshare usage paradigm based on accounts	283
Configure <code>slurm.conf</code> for fairshare	283
Create the accounts with fairshare values	284
Submit some jobs and view the change in job priorities	285
Cleanup	289
Accounting Discussion	290
Infrastructure.....	291
Slurm JobComp Configuration	292
Slurm Accounting Configuration Before Build	292
Slurm Accounting Configuration After Build	293
SlurmDBD Configuration.....	294
MySQL Configuration.....	295

Tools	297
Database Configuration	297
sacctmgr dump.....	298
Example	299
Database Maintenance.....	302
Cluster Options	303
Account Options	303
User Options.....	303
Limit Enforcement	304
Modifying Entities.....	304
Removing Entities	304
Accounting Exercises	305
Exercise 1: Use the sacct command to view status information of a running job/step:	305
Exercise 2: Use the sacctmgr command to manipulate the database information:.....	308
Cleanup.....	309
Advanced Resource Reservations Discussion	310
Reservation Creation	310
Reservation Use	313
Reservation Modification	313
Reservation Deletion	313
Overlapping Reservations.....	314
Reservations Floating Through Time	314
Reservations that Replace Allocated Resources.....	315
Reservation Purging After Last Job.....	316
Reservation Accounting	316
Prolog and Epilog	316
Reservations Lab Exercises	317
Exercise 1: Create an administrative reservation across the entire cluster	317
Exercise 2: Create a reservation on a specific node for a specific user	319
Cleanup	320
Managing Access to Compute Nodes Discussion	321
pam_slurm_adopt Module	321
Installation from Source	321
Installation by RPM.....	321

PAM Configuration	321
pam_slurm_adopt Module Options	322
Slurm Configuration.....	323
NSS_Slurm	324
Exercise 1: pam_slurm_adopt	325
Configure PAM:.....	325
Configure slurm.conf:	325
Watch it	325
Interactive session with cgroup enforcement	326
Cleanup	327
Exercise 2: Configure nss_slurm	327
Cleanup.....	328
sctrontab Discussion.....	329
Options	329
Sctrontab Options.....	330
Scrontab Examples.....	330
Examples.....	330
Exercise 1: sctrontab.....	331
Configure sctontrab:.....	331
Configure a batch job for fred:	331
Configure an scrontab entry for a fred:.....	331
Cleanup.....	332
Job Submit Plugins Discussion	333
Prolog/Epilog Discussion.....	333
FAILURE HANDLING	336
SPANK	336
SPANK Plugins.....	336
job_submit_lua plugin Discussion	337
Job Submit Plugins Lab Exercises	338
Exercise 1: Enable the throttle plugin.....	338
Cleanup.....	339
Exercise 2: Enable the require_timelimit plugin.....	339
Cleanup.....	341
Exercise 3: Create a job_submit.lua plugin.....	341

Cleanup.....	342
Exercise 4: Job Prolog Exercise	343
Cleanup.....	344
Exercise 5: SPANK Example.....	344
Exercise 6: SPANK and LUA Job Submit Example.....	346
Cleanup.....	347
Prometheus-Slurm-Exporter.....	348
Graphing Prometheus data with Grafana.....	349
Prometheus, Prometheus-slurm-exporter, and Grafana Lab Exercises	350
Exercise 1: Install Prometheus.....	350
Exercise 2: Install and Configure node_exporter.....	354
Exercise 3: Install the prometheus-slurm-exporter	356
Exercise 4: Configure Slurm for multiple accounts and generate some load	358
Exercise 5: Integrate Prometheus with Grafana.....	359
Cleanup.....	360
InfluxDB Plugin.....	361
Influx Installation	361
Integration with Slurm	361
Graphing InfluxDB data with Grafana.....	361
InfluxDB and Grafana Lab Exercises	363
Exercise 1: Verify the InfluxDB Installation.....	363
Exercise 2: Access the Grafana Dashboard.....	366
Cleanup	369
Slurm Triggers Discussion	371
Arguments:	371
Output Field Descriptions	374
Slurm System Triggers Discussion.....	374
Examples	374
Slurm Job Triggers Discussion.....	375
Examples	375
Slurm Triggers Lab Exercises.....	376
Exercise 1: Enable Node Failure Trigger	376
Exercise 2: Enable a Job Trigger.....	378
Cleanup.....	379

Break-Fix Labs	380
Break-Fix Lab Exercises.....	381
Exercise 1: Why won't my job run?	381
Cleanup.....	383
Exercise 2: Why aren't the job details accessible via sacct?.....	383
Cleanup.....	384
Exercise 3: Why aren't my user Access Control rights being honored?	384
Cleanup.....	384
Exercise 4: Why don't my jobs pack?	385
Cleanup.....	385
Running Jobs in Containers.....	386
Prerequisites.....	386
Required software:	386
Slurm, SPANK, and pyxis Container Lab Exercises	387
Exercise 1: Install enroot	387
Exercise 2: Install pyxis	387
Exercise 3: Run a job in a container	387
Cleanup	388
License Management in Slurm	389
Local Licenses	389
Remote Licenses	389
USE CASE.....	389
CONFIGURING SLURM FOR THE USE CASE	390
Dynamic Licenses.....	393
A note on job preemption with licenses.....	394
Slurm Cloud Scheduling Guide.....	395
Slurm Configuration.....	395
Operational Details.....	399
Cloud Node Lifecycle	400
Slurm Cloud Lab Exercise.....	406
Exercise: Boot the Docker Scaleout environment in "Cloud" configuration	406
Cleanup	408
Appendix.....	409
File Movement Discussion	410

sbcast.....	410
OPTIONS	410
EXAMPLE.....	411
sgather	411
OPTIONS	411
EXAMPLE.....	411
File Movement Exercises	412
Exercise 1: Using sbcast and sgather	412
Cleanup.....	413

Accessing the Slurm Training Learning Environment

The training environment is hosted in AWS. How to connect to the instance, and managing your way around the environment, is shown here.

The Learning Environment

In order to accomplish the hands-on portion of this training material, you will be given access to an IP address and an ssh key (either .pem or .ppk) to attach with.

The infrastructure for this cluster is based on Docker containers.

The Docker Containers are:

- node00-node09: 10 compute nodes
- influxdb: The influx database server
- grafana: The Grafana open source dashboard server
- xmod: The OpenXMOD server
- open-ondemand: The Open OnDemand server
- db: The MySQL database node
- es01: ElasticSearch node1
- es02: ElasticSearch node2
- es03: ElasticSearch node3
- kibana: The kibana service node
- login: The login node. (You will do most of your work on this host)
- proxy: The proxy node for JWT
- rest: The slurmrestd REST API service node
- mgmtnode: The Primary scheduler server node
- mgmtnode2: The Backup scheduler server node
- slurmdbd: The SlurmDBD node

The hostnames are defined in the */etc/hosts* file across the docker cluster. In addition, the ssh keys are captured for all users across the cluster Docker nodes, so no passwords are required

Attaching to the AWS Instance – Linux/Mac Method

In order to connect to the cluster in AWS, you will need 2 things: 1) an IP Address and 2) an ssh .pem key to attach with. You will get both from the instructor

1. Open a terminal.
2. Change directory to where your `slurm_training.pem` file resides.
3. Change the permissions on the `slurm_training.pem` key to 600:
`chmod 600 slurm_training.pem`
4. Attach to your instance like this:
`ssh -i slurm_training.pem ubuntu@<IP ADDRESS>`

NOTE: You should NOT be prompted for a password or passphrase.

You should end up at a prompt looking like this:

```
ubuntu@ip-172-31-17-131:~$
```

5. Change directories to the Docker scaleout directory and see the Docker containers:

```
cd docker-scale-out
```

6. See the docker containers:

```
docker ps --format "{{.Names}}: {{.Status}}"
```

You should see:

```
docker-scale-out_open-ondemand_1: Up 2 minutes
docker-scale-out_proxy_1: Up 2 minutes
docker-scale-out_mgmtnode2_1: Up 2 minutes
docker-scale-out_node02_1: Up 2 minutes
docker-scale-out_node03_1: Up 2 minutes
docker-scale-out_node07_1: Up 2 minutes
docker-scale-out_node05_1: Up 2 minutes
docker-scale-out_node01_1: Up 2 minutes
docker-scale-out_node08_1: Up 2 minutes
docker-scale-out_node04_1: Up 2 minutes
docker-scale-out_node00_1: Up 2 minutes
docker-scale-out_node09_1: Up 2 minutes
docker-scale-out_rest_1: Up 2 minutes
docker-scale-out_node06_1: Up 2 minutes
docker-scale-out_mgmtnode_1: Up 2 minutes
docker-scale-out_kibana_1: Up 3 minutes
docker-scale-out_slurmdbd_1: Up 3 minutes
docker-scale-out_es02_1: Up 3 minutes
docker-scale-out_login_1: Up 2 minutes
docker-scale-out_xdmod_1: Up 3 minutes
docker-scale-out_grafana_1: Up 3 minutes
docker-scale-out_es01_1: Up 3 minutes
docker-scale-out_influxdb_1: Up 3 minutes
docker-scale-out_db_1: Up 3 minutes
docker-scale-out_es03_1: Up 3 minutes
```

7. To connect to the login Docker container, use the following method:

```
docker-compose exec login bash
```

You should see:

```
[root@login ~]#
```

Attaching to the AWS Instance – Windows 10 Method

Windows 10 has a built-in ssh client that is accessible directly from the Command Prompt. In order to connect to the cluster in AWS, you will need 2 things: 1) an IP Address and 2) an ssh pem key to attach with.

NOTE: Before starting, make sure the permissions on the .pem file are “Full Control” for your user.

NOTE: If you are an administrator, then you shouldn’t need to add the permissions.

NOTE: You also may need to clear all permissions except owner (you) in Advanced Security Settings.

1. Open a Command Prompt by...
 - a. Press: **Windows key** (on your keyboard.)
 - b. Type: **command prompt**
 - c. Press: **Enter**
2. When the command prompt instance comes up, attach to your instance like this:
ssh -i slurm_training.pem ubuntu@<IP ADDRESS>

You should NOT be prompted for a password or passphrase. You should end up at a prompt looking like this:

```
ubuntu@ip-172-31-17-
```

3. Change directories to the Docker scaleout directory:
cd docker-scale-out
4. See the Docker containers:
docker ps --format "{{.Names}}: {{.Status}}

You should see:

```
docker-scale-out_open-ondemand_1: Up 2 minutes
docker-scale-out_proxy_1: Up 2 minutes
docker-scale-out_mgmtnode2_1: Up 2 minutes
docker-scale-out_node02_1: Up 2 minutes
docker-scale-out_node03_1: Up 2 minutes
docker-scale-out_node07_1: Up 2 minutes
docker-scale-out_node05_1: Up 2 minutes
docker-scale-out_node01_1: Up 2 minutes
docker-scale-out_node08_1: Up 2 minutes
docker-scale-out_node04_1: Up 2 minutes
docker-scale-out_node00_1: Up 2 minutes
docker-scale-out_node09_1: Up 2 minutes
docker-scale-out_rest_1: Up 2 minutes
docker-scale-out_node06_1: Up 2 minutes
docker-scale-out_mgmtnode_1: Up 2 minutes
docker-scale-out_kibana_1: Up 3 minutes
docker-scale-out_slurmdbd_1: Up 3 minutes
docker-scale-out_es02_1: Up 3 minutes
docker-scale-out_login_1: Up 2 minutes
docker-scale-out_xdmod_1: Up 3 minutes
docker-scale-out_grafana_1: Up 3 minutes
docker-scale-out_es01_1: Up 3 minutes
docker-scale-out_influxdb_1: Up 3 minutes
docker-scale-out_db_1: Up 3 minutes
docker-scale-out_es03_1: Up 3 minutes
```

5. To connect to the login Docker container, use the following method:

docker-compose exec login bash

You should see:

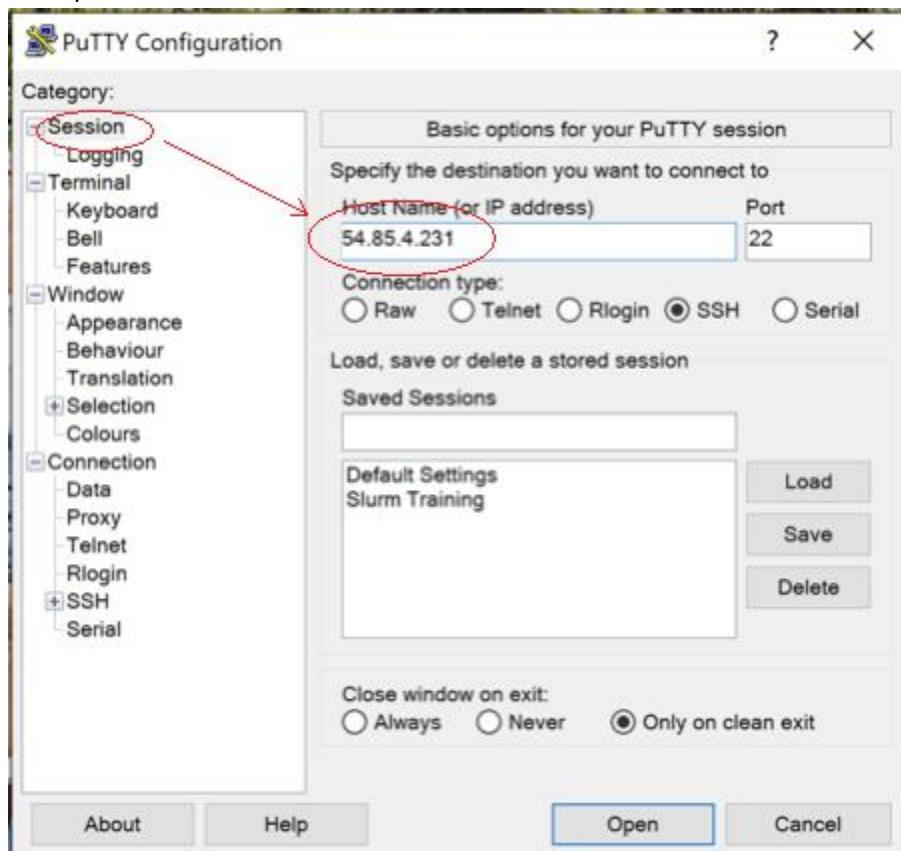
```
[root@login
```

Attaching to the AWS Instance – Windows 7 (PuTTY) Method

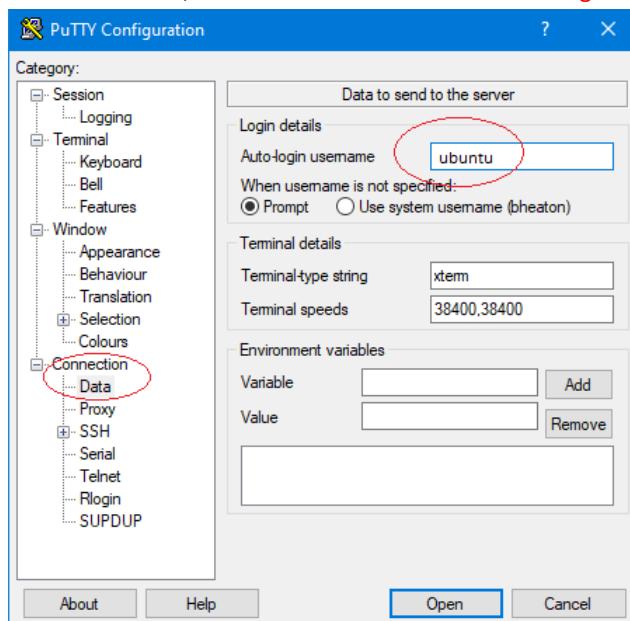
Windows 7 does not have a built-in ssh client. So in order to connect to the cluster in AWS, you will need an ssh client such as PuTTY. Once you have installed PuTTY, you will need 2 more things: 1) an IP Address and 2) an ssh .ppk key to attach with.

1. Once you have obtained the key and IP Address, [launch PuTTY](#).

2. Enter your IP address in the Host Name field.

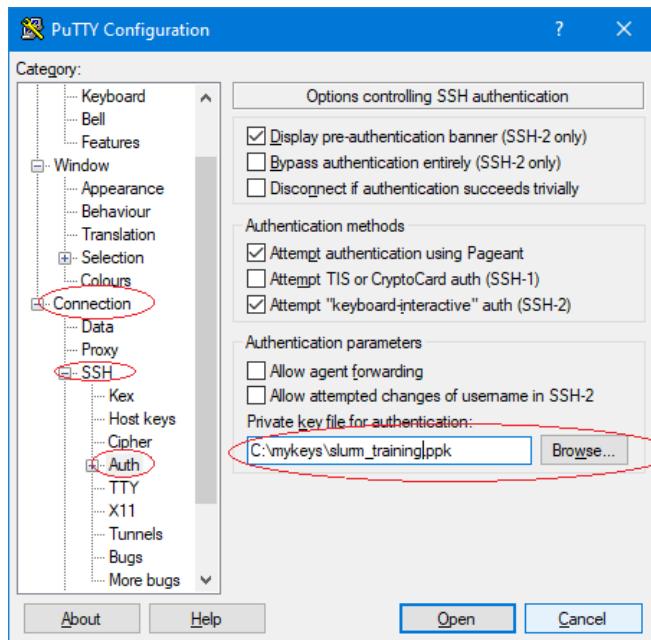


3. In the left frame, select: Connection > Data > Auto-login username:



4. Enter **ubuntu** as the username:

5. Next, in the left frame, expand **Connection > SSH > Auth** > Private key file for authentication:



6. Browse to the .ppk file, provided by the instructor.
7. Select the .ppk file and click the Open button.
8. (Save you session if you wish to make repeating these steps easy.)
9. Click Open and you should attach to your AWS instance.

You should NOT be prompted for a password or passphrase. You should end up at a prompt looking like this:

```
ubuntu@ip-172-31-17-131:~$
```

10. Change directories to the Docker scaleout directory and see the Docker containers:

```
cd docker-scale-out
docker ps --format "{{.Names}}: {{.Status}}
```

11. You should see:

```
docker-scale-out_open-ondemand_1: Up 2 minutes
docker-scale-out_proxy_1: Up 2 minutes
docker-scale-out_mgmtnode2_1: Up 2 minutes
docker-scale-out_node02_1: Up 2 minutes
docker-scale-out_node03_1: Up 2 minutes
docker-scale-out_node07_1: Up 2 minutes
docker-scale-out_node05_1: Up 2 minutes
docker-scale-out_node01_1: Up 2 minutes
docker-scale-out_node08_1: Up 2 minutes
docker-scale-out_node04_1: Up 2 minutes
docker-scale-out_node00_1: Up 2 minutes
docker-scale-out_node09_1: Up 2 minutes
docker-scale-out_rest_1: Up 2 minutes
docker-scale-out_node06_1: Up 2 minutes
docker-scale-out_mgmtnode_1: Up 2 minutes
docker-scale-out_kibana_1: Up 3 minutes
docker-scale-out_slurmdbd_1: Up 3 minutes
docker-scale-out_es02_1: Up 3 minutes
docker-scale-out_login_1: Up 2 minutes
docker-scale-out_xdmod_1: Up 3 minutes
docker-scale-out_grafana_1: Up 3 minutes
docker-scale-out_es01_1: Up 3 minutes
docker-scale-out_influxdb_1: Up 3 minutes
docker-scale-out_db_1: Up 3 minutes
docker-scale-out_es03_1: Up 3 minutes
```

12. To connect to the login Docker container, use the following method:

```
docker-compose exec login bash
```

You should see:

```
[root@login ~]#
```

The /lab_scripts folder and Files

This course contains many labs which reinforce the concepts taught in each section. Many of the lab require the creation of files. Rather than type (or copy/paste from the manual) the files out, I have included the lab scripts in the login Docker container. They are in `/lab_scripts` in the root of the filesystem.

1. To see the lab scripts, in the login Docker container, type:

```
ls -l /lab_scripts
```

2. You should see:

```
total 132
drwxrwxrwx 1 root root 4096 Sep  9 16:09 .
drwxr-xr-x 1 root root 4096 Sep  9 16:16 ..
-rwxrwxr-x 1 root root  212 Sep  9 15:39 PrologSlurmctld.sh
-rw-rw-r-- 1 root root  973 Sep  9 15:39 alloc.c
-rwxrwxr-x 1 root root 1283 Sep  9 15:39 arrayjob.py
-rwxrwxr-x 1 root root  931 Sep  9 15:39 canceljob.py
-rw-rw-r-- 1 root root  129 Sep  9 15:39 cgroup.conf
-rw-rw-r-- 1 root root 2628 Sep  9 15:39 cluster.cfg
-rwxrwxr-x 1 root root  192 Sep  9 15:39 dataprocessingjob.sh
-rwxrwxr-x 1 root root   50 Sep  9 15:39 dataprocessingprogram.sh
-rwxrwxr-x 1 root root 1307 Sep  9 15:39 depjob.py
-rwxrwxr-x 1 root root  724 Sep  9 15:39 gen_jwt.py
-rwxrwxr-x 1 root root  419 Sep  9 15:39 gethostname.sh
-rwxrwxr-x 1 root root 1453 Sep  9 15:39 hetjob.py
-rw-rw-r-- 1 root root  466 Sep  9 15:39 job.json
-rwxrwxr-x 1 root root 1297 Sep  9 15:39 job.py
-rw-rw-r-- 1 root root  562 Sep  9 15:39 job_submit.lua
-rw-rw-r-- 1 root root  516 Sep  9 15:39 job_submit_spank.lua
-rw-rw-r-- 1 root root   97 Sep  9 15:39 memalloc.c
-rw-rw-r-- 1 root root  306 Sep  9 15:39 myprogram.c
-rwxrwxr-x 1 root root  273 Sep  9 15:39 myslurmarray.sh
-rw-rw-r-- 1 root root 1565 Sep  9 15:39 pi.c
-rw-rw-r-- 1 root root  134 Sep  9 15:39 plugstack.conf
-rwxrwxr-x 1 root root  418 Sep  9 15:39 prolog.sh
-rwxrwxr-x 1 root root  865 Sep  9 15:39 qos.sh
-rw-rw-r-- 1 root root 3347 Sep  9 15:39 renice.c
-rwxrwxr-x 1 root root  912 Sep  9 15:39 sacct.py
-rwxrwxr-x 1 root root  748 Sep  9 15:39 sdiag.py
-rwxrwxr-x 1 root root  913 Sep  9 15:39 showjob.py
-rw-rw-r-- 1 root root  160 Sep  9 15:39 testping.sh.fred
-rw-rw-r-- 1 root root  151 Sep  9 15:39 testping.sh.pebbles
-rw-rw-r-- 1 root root  126 Sep  9 15:39 topology.conf
-rwxrwxr-x 1 root root  554 Sep  9 15:39 verify_jwt.py
-rw-rw-r-- 1 root root 2711 Sep  9 15:39 whereami.c
```

Building The Learning Environment for Yourselves

The Slurm cluster and accompanying supporting software can be installed in your own environment. Either on a stand-alone computer or in a Virtual Machine. The requirements are:

- **Ubuntu 20.04 LTS, 4 CPUs, 16GB, 25GB disk, User=ubuntu, Password=ubuntu**

- Once you have your Ubuntu VM/Computer installed, log in as user: **root**
- Open a terminal, and enter the following (**all on one line**):
`sudo apt -y update && apt -y upgrade && apt -y install curl gcc make
wget jq`
- Next, the following command (**all on one line**), to begin setting up the docker-compose environment:
`sudo curl -L sudo curl -SL
https://github.com/docker/compose/releases/download/v2.14.2/docker-
compose-linux-x86_64 -o /usr/local/bin/docker-compose`

- Now, enter all of the following command (**in a series**):

```
curl -sSL https://get.docker.com/ | sudo bash  
  
rm -rf /usr/bin/docker-compose  
  
ln -s /usr/local/bin/docker-compose /usr/bin/docker-compose  
  
chmod +x /usr/local/bin/docker-compose  
  
usermod -a -G docker ubuntu  
  
reboot
```

- After **reboot**, login as:

User: **ubuntu**
Password: **ubuntu**

- Switch to the root user:

```
sudo su -
```

- As **root**, enter the following (as all one command -- the quotes will allow everything to be entered at the same time -- copy/paste the whole block or type it exactly as it appears [quotes are in **blue** to make them easy to see exactly where they begin and end]):

```
echo "  
net.ipv4.tcp_max_syn_backlog=4096  
net.core.netdev_max_backlog=1000  
net.core.somaxconn=15000  
fs.file-max=992832  
  
# Force gc to clean-up quickly  
net.ipv4.neigh.default.gc_interval = 3600  
  
# Set ARP cache entry timeout  
net.ipv4.neigh.default.gc_stale_time = 3600  
  
# Setup DNS threshold for arp  
net.ipv4.neigh.default.gc_thresh3 = 8096
```

```
net.ipv4.neigh.default.gc_thresh2 = 4048
net.ipv4.neigh.default.gc_thresh1 = 1024

# Increase map count for elasticsearch
vm.max_map_count=262144

# Avoid running out of file descriptors
fs.file-max=10000000
fs.inotify.max_user_instances=65535
fs.inotify.max_user_watches=1048576
" >> /etc/sysctl.conf
```

8. Update sysctl:

```
sysctl --system
```

9. Exit from root, back to **ubuntu** user.

10. Enter the following as **all one line**:

```
git clone --depth 1 --single-branch -b master
https://gitlab.com/SchedMD/training/docker-scale-out.git
```

11. Switch to the docker-scale-out directory:

```
cd docker-scale-out
```

12. Change the version of Slurm you want to build

```
export SLURM_RELEASE=slurm-23-02-4-1
```

13. Build the docker containers using docker-compose and make:

```
make build
```

NOTE: This takes a minute or two. (Or 20?). Once the command line prompt re-appears, the docker scale out (DSO) environment should be all done and ready to use.

13. See the Docker containers:

```
cd docker-scale-out
docker ps --format "{{.Names}}: {{.Status}}" | column -t
```

You should see:

docker-scale-out-proxy-1:	Up	About	an	hour
docker-scale-out-node08-1:	Up	About	an	hour
docker-scale-out-node09-1:	Up	About	an	hour
docker-scale-out-node05-1:	Up	About	an	hour
docker-scale-out-node01-1:	Up	About	an	hour
docker-scale-out-node02-1:	Up	About	an	hour
docker-scale-out-node06-1:	Up	About	an	hour
docker-scale-out-node00-1:	Up	About	an	hour
docker-scale-out-mgmtnode2-1:	Up	About	an	hour
docker-scale-out-node07-1:	Up	About	an	hour
docker-scale-out-rest-1:	Up	About	an	hour
docker-scale-out-node03-1:	Up	About	an	hour
docker-scale-out-node04-1:	Up	About	an	hour
docker-scale-out-mgmtnode-1:	Up	About	an	hour
docker-scale-out-open-on-demand-1:	Up	About	an	hour
docker-scale-out-kibana-1:	Up	About	an	hour
docker-scale-out-slurmdbd-1:	Up	About	an	hour
docker-scale-out-es03-1:	Up	About	an	hour
docker-scale-out-influxdb-1:	Up	About	an	hour
docker-scale-out-login-1:	Up	About	an	hour
docker-scale-out-grafana-1:	Up	About	an	hour
docker-scale-out-es02-1:	Up	About	an	hour
docker-scale-out-es01-1:	Up	About	an	hour
docker-scale-out-db-1:	Up	About	an	hour
docker-scale-out-xdmod-1:	Up	About	an	hour

14. To connect to the login Docker container, use the following method:

`docker-compose exec login bash`

You should see:

[root@login ~]#

You can now use the DSO environment to do the other labs

Submitting and Running Jobs Discussion

Slurm is an open source, fault-tolerant, and highly scalable cluster management and job scheduling system for large and small Linux clusters. Slurm requires no kernel modifications for its operation and is relatively self-contained. As a cluster workload manager, Slurm has three key functions. First, it allocates exclusive and/or non-exclusive access to resources (compute nodes) to users for some duration of time so they can perform work. Second, it provides a framework for starting, executing, and monitoring work (normally a parallel job) on the set of allocated nodes. Finally, it arbitrates contention for resources by managing a queue of pending work.

Architecture

As depicted in Figure 1, Slurm consists of a slurmd daemon running on each compute node and a central slurmctld daemon running on a management node (with optional fail-over twin). The slurmd daemons provide fault-tolerant hierarchical communications. The user commands include: sacct, salloc, sattach, sbatch, sbcast, scancel, scontrol, sinfo, smap, squeue, srun, strigger and sview. All of the commands can run anywhere in the cluster.

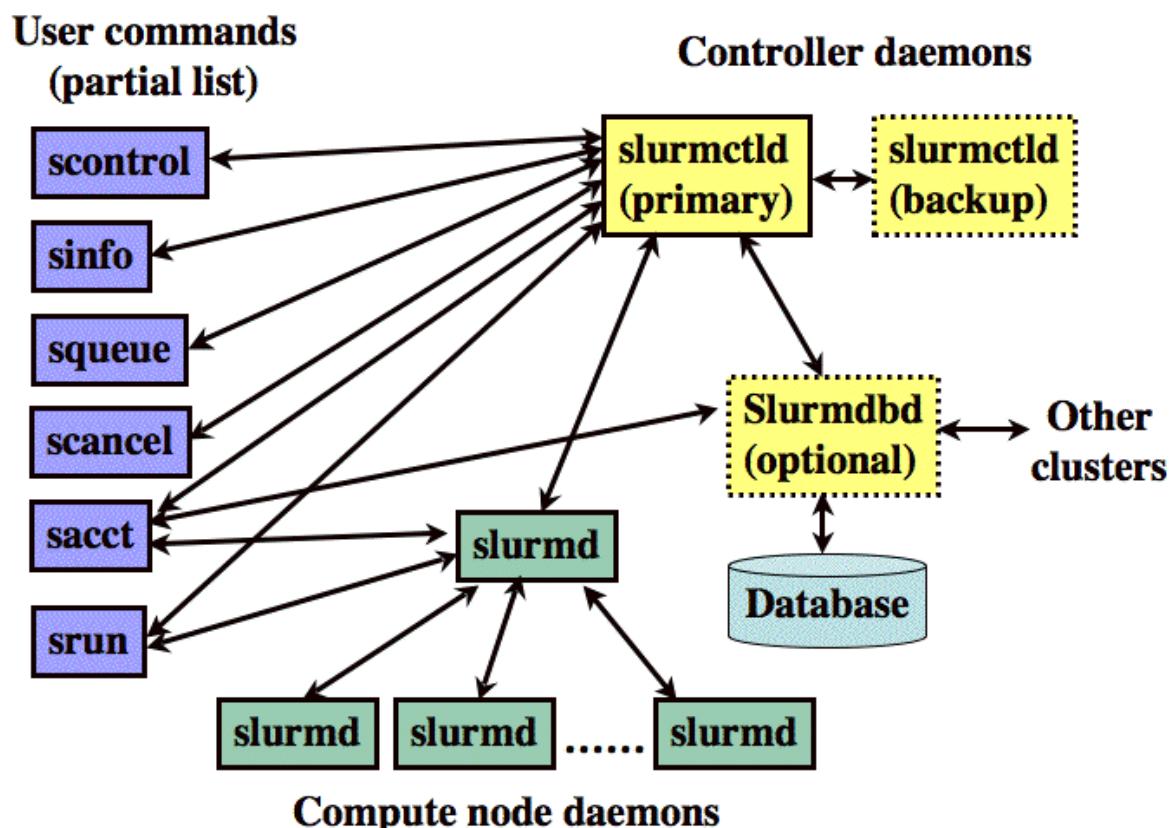


Figure 1. Slurm components

The entities managed by these Slurm daemons, shown in Figure 2, include nodes, the compute resource in Slurm, partitions, which group nodes into logical (possibly overlapping) sets, jobs, or allocations of resources assigned to a user for a specified amount of time, and job steps, which are sets of (possibly parallel) tasks within a job. The partitions can be considered job queues, each of which has an assortment of constraints such as job size limit, job time limit, users permitted to use it, etc. Priority-ordered jobs are allocated nodes within a partition until the resources (nodes, processors, memory, etc.) within that partition are exhausted. Once a job is assigned a set of nodes, the user is able to initiate parallel work in the form of job steps in any configuration within the allocation. For instance, a single job step may be started that utilizes all nodes allocated to the job, or several job steps may independently use a portion of the allocation.

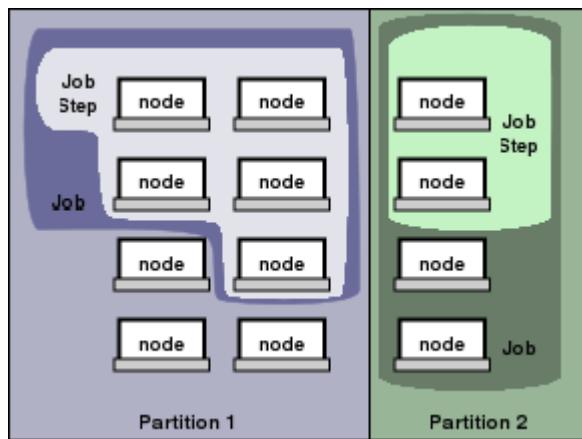


Figure 2. Slurm entities

Summary of Slurm Commands

Man pages exist for all Slurm daemons, commands, and API functions. The command option --help also provides a brief summary of options. **Note** that the command options are all case sensitive.

- sacct** is used to report job or job step accounting information about active or completed jobs.
- salloc** is used to allocate resources for a job in real time. Typically this is used to allocate resources and spawn a shell. The shell is then used to execute `srun` commands to launch parallel tasks.
- sattach** is used to attach standard input, output, and error plus signal capabilities to a currently running job or job step. One can attach to and detach from jobs multiple times.
- sbatch** is used to submit a job script for later execution. The script will typically contain one or more `srun` commands to launch parallel tasks.
- sbcast** is used to transfer a file from local disk to local disk on the nodes allocated to a job. This can be used to effectively use diskless compute nodes or provide improved performance relative to a shared file system.
- scancel** is used to cancel a pending or running job or job step. It can also be used to send an arbitrary signal to all processes associated with a running job or job step.
- scontrol** is the administrative tool used to view and/or modify Slurm state. **Note** that many `scontrol` commands can only be executed as user root.

- sinfo** reports the state of partitions and nodes managed by Slurm. It has a wide variety of filtering, sorting, and formatting options.
- squeue** reports the state of jobs or job steps. It has a wide variety of filtering, sorting, and formatting options. By default, it reports the running jobs in priority order and then the pending jobs in priority order.
- srun** is used to submit a job for execution or initiate job steps in real time. **srun** has a wide variety of options to specify resource requirements, including: minimum and maximum node count, processor count, specific nodes to use or not use, and specific node characteristics (so much memory, disk space, certain required features, etc.). A job can contain multiple job steps executing sequentially or in parallel on independent or shared resources within the job's node allocation.
- triggerer** is used to set, get or view event triggers. Event triggers include things such as nodes going down or jobs approaching their time limit.
- Sview** is a graphical user interface to get and update state information for jobs, partitions, and nodes managed by Slurm.

Examples

In this first example, we determine what partitions exist on the system, what nodes they include, and general system state. This information is provided by the **sinfo** command. In the example below we find there are two partitions: debug and batch. The * following the name debug indicates this is the default partition for submitted jobs. We see that both partitions are in an UP state. Some configurations may include partitions for larger jobs that are DOWN except on weekends or at night. The information about each partition may be split over more than one line so that nodes in different states can be identified. In this case, the two nodes adev[1-2] are down. The * following the state down indicate the nodes are not responding. **Note** the use of a concise expression for node name specification with a common prefix adev and numeric ranges or specific numbers identified. This format allows for very clusters to be easily managed. The **sinfo** command has many options to easily let you view the information of interest to you in whatever format you prefer. See the man page for more information:

```
$ sinfo
PARTITION AVAIL  TIMELIMIT NODES  STATE NODELIST
debug*      up      30:00     2  down* adev[1-2]
debug*      up      30:00     3    idle adev[3-5]
batch       up      30:00     3  down* adev[6,13,15]
batch       up      30:00     3  alloc  adev[7-8,14]
batch       up      30:00     4    idle adev[9-12]
```

Next we determine what jobs exist on the system using the **squeue** command. The ST field is the job state. Two jobs are in a running state (R is an abbreviation for Running) while one job is in a pending state (PD is an abbreviation for Pending). The TIME field shows how long the jobs have run for using the format days-hours:minutes:seconds. The NODELIST(REASON) field indicates where the job is running or the reason it is still pending. Typical reasons for pending jobs are Resources (waiting for resources to become available) and Priority (queued behind a higher priority job). The **squeue** command has many options to easily let you view the information of interest to you in whatever format you prefer. See the man page for more information.

```
$ squeue
```

```

JOBID PARTITION NAME USER ST TIME NODES NODELIST(REASON)
65646 batch chem mike R 24:19      2 adev[7-8]
65647 batch bio joan R 0:09       1 adev14
65648 batch math phil PD 0:00      6 (Resources)

```

The `scontrol` command can be used to report more detailed information about nodes, partitions, jobs, job steps, and configuration. It can also be used by system administrators to make configuration changes. A couple of examples are shown below. See the man page for more information.

```

$ scontrol show partition
PartitionName=debug TotalNodes=5 TotalCPUs=40 RootOnly=NO
  Default=YES OverSubscribe=FORCE:4 PriorityTier=1 State=UP
  MaxTime=00:30:00 Hidden=NO
  MinNodes=1 MaxNodes=26 DisableRootJobs=NO AllowGroups=ALL
  Nodes=adev[1-5] NodeIndices=0-4

PartitionName=batch TotalNodes=10 TotalCPUs=80 RootOnly=NO
  Default=NO OverSubscribe=FORCE:4 PriorityTier=1 State=UP
  MaxTime=16:00:00 Hidden=NO
  MinNodes=1 MaxNodes=26 DisableRootJobs=NO AllowGroups=ALL
  Nodes=adev[6-15] NodeIndices=5-14

$ scontrol show node adev1
NodeName=adev1 State=DOWN* CPUs=8 AllocCPUs=0
  RealMemory=4000 TmpDisk=0
  Sockets=2 Cores=4 Threads=1 Weight=1 Features=intel
  Reason=Not responding [slurm@06/02-14:01:24]

65648      batch math phil PD 0:00      6 (Resources)

$ scontrol show job
JobId=65672 UserId=phil(5136) GroupId=phil(5136)
  Name=math
  Priority=4294901603 Partition=batch BatchFlag=1
  AllocNode:Sid=adev0:16726 TimeLimit=00:10:00 ExitCode=0:0
  StartTime=06/02-15:27:11 EndTime=06/02-15:37:11
  JobState=PENDING NodeList=(null) NodeListIndices=
  NumCPUs=24 ReqNodes=1 ReqS:C:T=1-65535:1-65535:1-65535
  OverSubscribe=1 Contiguous=0 CPUs/task=0 Licenses=(null)
  MinCPUs=1 MinSockets=1 MinCores=1 MinThreads=1
  MinMemory=0 MinTmpDisk=0 Features=(null)
  Dependency=(null) Account=(null) Requeue=1
  Reason=None Network=(null)
  ReqNodeList=(null) ReqNodeListIndices=
  ExcNodeList=(null) ExcNodeListIndices=
  SubmitTime=06/02-15:27:11 SuspendTime=None PreSusTime=0
  Command=/home/phil/math
  WorkDir=/home/phil

```

It is possible to create a resource allocation and launch the tasks for a job step in a single command line using the `srun` command. Depending upon the MPI implementation used, MPI jobs may also be launched in this manner. See the MPI training module for more MPI-specific information. In this example we execute `/bin/hostname` on three nodes (-N3) and include task numbers on the output (-l). The default partition will be used. One task per

node will be used by default. **Note** that the `srun` command has many options available to control what resource are allocated and how tasks are distributed across those resources.

```
$ srun -N3 -l /bin/hostname
0: adev3
1: adev4
2: adev5
```

This variation on the previous example executes `/bin/hostname` in four tasks (`-n4`). One processor per task will be used by default (**note** that we don't specify a node count).

```
$ srun -n4 -l /bin/hostname
0: adev3
1: adev3
2: adev3
3: adev3
```

One common mode of operation is to submit a script for later execution. In this example the script name is `my.script` and we explicitly use the nodes `adev9` and `adev10` (`-w "adev[9-10]`", **note** the use of a node range expression). We also explicitly state that the subsequent job steps will spawn four tasks each, which will ensure that our allocation contains at least four processors (one processor per task to be launched). The output will appear in the file `my.stdout` ("`-o my.stdout`"). This script contains a timelimit for the job embedded within itself. Other options can be supplied as desired by using a prefix of "#SBATCH" followed by the option at the beginning of the script (before any commands to be executed in the script). Options supplied on the command line would override any options specified within the script. **Note** that `my.script` contains the command `/bin/hostname` that is executed on the first node in the allocation (where the script runs) plus two job steps initiated using the `srun` command and executed sequentially.

```
$ cat my.script
#!/bin/sh
#SBATCH --time=1
/bin/hostname
srun -l /bin/hostname
srun -l /bin/pwd

$ sbatch -n4 -w "adev[9-10]" -o my.stdout my.script
sbatch: Submitted batch job 469

$ cat my.stdout
adev9
0: adev9
1: adev9
2: adev10
3: adev10
0: /home/jette
1: /home/jette
2: /home/jette
```

```
3: /home/jette
```

The final mode of operation is to create a resource allocation and spawn job steps within that allocation. The `salloc` command is used to create a resource allocation and typically start a shell within that allocation. One or more job steps would typically be executed within that allocation using the `srun` command to launch the tasks (depending upon the type of MPI being used, the launch mechanism may differ, see MPI details below). Finally the shell created by `salloc` would be terminated using the `exit` command. Slurm does not automatically migrate executable or data files to the nodes allocated to a job. Either the files must exist on local disk or in some global file system (e.g. NFS or Lustre). We provide the tool `sbcast` to transfer files to local storage on allocated nodes using Slurm's hierarchical communications. In this example we use `sbcast` to transfer the executable program `a.out` to `/tmp/joe.a.out` on local storage of the allocated nodes. After executing the program, we delete it from local storage

```
$ salloc -N1024 bash
$ sbcast a.out /tmp/joe.a.out
Granted job allocation 471

$ srun /tmp/joe.a.out
Result is 3.14159

$ srun rm /tmp/joe.a.out
$ exit
salloc: Relinquishing job allocation 471"
```

In this example, we submit a batch job, get its status, and cancel it.

```
$ sbatch test
srun: jobid 473 submitted

$ squeue
JOBID PARTITION NAME USER ST TIME NODES NODELIST (REASON)
 473 batch    test jill R 00:00 1      adev9

$ scancel 473

$ squeue
JOBID PARTITION NAME USER ST TIME NODES NODELIST (REASON)
```

Submitting Batch Jobs using the `sbatch` Command

The typical method for submitting serial workload to a cluster is with the `sbatch` command. It can be as simple as:

```
sbatch script.sh
```

The command specified in script.sh file will then be evaluated by the scheduler and run on the first available compute node(s) that fits the resources requested in the script. The command `sbatch` can then be used to see your job in the queue, regardless of its state (unless it's completed, then it goes off the queue).

The structure of a job submit script contains zero or more directives or commands to tell the job scheduler what to do. A convenient usage output can be gathered using `sbatch --usage`:

```
[slurm@slurm ~]$ sbatch --usage
Usage: sbatch [-N nnodes] [-n ntasks]
              [-c ncpus] [-r n] [-p partition] [--hold] [--parsable] [-t minutes]
              [-D path] [--no-kill] [--overcommit]
              [--input file] [--output file] [--error file]
              [--time-min=minutes] [--licenses=names] [--clusters=cluster_names]
              [--chdir=directory] [--oversubscribe] [-m dist] [-J jobname]
              [--verbose] [--gid=group] [--uid=user]
              [--contiguous] [--mincpus=n] [--mem=MB] [--tmp=MB] [-C list]
              [--account=name] [--dependency=type:jobid[:time]] [--comment=name]
              [--mail-type=type] [--mail-user=user] [--nice[=value]] [--wait]
              [--requeue] [--no-requeue] [--ntasks-per-node=n] [--propagate]
              [--nodefile=file] [--nodelist=hosts] [--exclude=hosts]
              [--network=type] [--mem-per-cpu=MB] [--qos=qos] [--gres=list]
              [--mem-bind=...] [--reservation=name] [--mcs-label=mcs]
              [--cpu-freq=min[-max[:gov]]] [--power=flags] [--gres-flags=opts]
              [--switches=max-switches{@max-time-to-wait}] [--reboot]
              [--core-spec=cores] [--thread-spec=threads]
              [--bb=burst_buffer_spec] [--bbf=burst_buffer_file]
              [--array=index_values] [--profile=...] [--ignore-pbs] [--spread-job]
              [--export[=names]] [--export-file=file|fd] [--delay-boot=mins]
              [--use-min-nodes]
              [--cpus-per-gpu=n] [--gpus=n] [--gpu-bind=...] [--gpu-freq=...]
              [--gpus-per-node=n] [--gpus-per-socket=n] [--gpus-per-task=n]
              [--mem-per-gpu=MB] [--tres-per-task=list]
              executable [args...]
```

The short summary of all the options are listed below (for a complete description, see the man page):

Option	Description
<code>-A, --account=<account></code>	Charge resources used by this job to specified account.
<code>--acctg-freq</code>	Define the job accounting and profiling sampling intervals.
<code>-a, --array=<indexes></code>	Submit a job array, multiple jobs to be executed with identical parameters.
<code>--batch=<list></code>	Nodes can have features assigned to them by the Slurm administrator.
<code>--bb=<spec></code>	Burst buffer specification.

--bbf=<file_name>	Path of file containing burst buffer specification.
-b --begin=<time>	Submit the batch script to the Slurm controller immediately, like normal, but tell the controller to defer the allocation of the job until the specified time.
-D, --chdir=<directory>	Set the working directory of the batch script to directory before it is executed. The path can be specified as full path or relative path to the directory where the command is executed.
--cluster-constraint=[!]<list>	Specifies features that a federated cluster must have to have a sibling job submitted to it.
-M, --clusters=<string>	Clusters to issue commands to.
--comment=<string>	An arbitrary comment enclosed in double quotes if using spaces or some special characters.
-C, --constraint=<list>	Nodes can have features assigned to them by the Slurm administrator. Users can specify which of these features are required by their job using the constraint option. node_features/helpers have been updated in Slurm 23.02 to add support for for the OR and parentheses operators in a --constraint expression
--container=<path_to_container>	Absolute path to OCI container bundle.
--contiguous	If set, then the allocated nodes must form a contiguous set.
-S, --core-spec=<num>	Count of specialized cores per node reserved by the job for system operations and not used by the application.
--cores-per-socket=<cores>	Restrict node selection to nodes with at least the specified number of cores per socket.
--cpu-freq =<p1[-p2[:p3]]>	Request that job steps initiated by srun commands inside this sbatch script be run at some requested frequency if possible, on the CPUs selected for the step on the compute node(s).
--cpus-per-gpu=<ncpus>	Advise Slurm that ensuing job steps will require ncpus processors per allocated GPU
-c, --cpus-per-task=<ncpus>	Advise the Slurm controller that ensuing job steps will require ncpus number of processors per task. Without this option, the controller will just try to allocate one processor per task.

--deadline=<OPT>	remove the job if no ending is possible before this deadline (start > (deadline - time[-min])).
--delay-boot=<minutes>	Do not reboot nodes in order to satisfied this job's feature specification if the job has been eligible to run for less than this time period.
-d, --dependency=<dependency_list>	Defer the start of this job until the specified dependencies have been satisfied completed. <dependency_list> is of the form <type:job_id[:job_id][,type:job_id[:job_id]]> or <type:job_id[:job_id][?type:job_id[:job_id]]>.
-m, --distribution=arbitrary <block cyclic plane=<options>[:block cyclic fcyclic]>	Specify alternate distribution methods for remote processes.
-e, --error=<filename pattern>	Instruct Slurm to connect the batch script's standard error directly to the file name specified in the "filename pattern". In Slurm V23.02, we now will automatically create the directory and filename if they do not already exist.
-x, --exclude=<node name list>	Explicitly exclude certain nodes from the resources granted to the job.
--exclusive[=user mcs]	The job allocation can not share nodes with other running jobs (or just other users with the "=user" option or with the "=mcs" option).
--export=<environment variables [ALL] NONE>	Identify which environment variables from the submission environment are propagated to the launched application.
--export-file=<filename fd>	If a number between 3 and OPEN_MAX is specified as the argument to this option, a readable file descriptor will be assumed (STDIN and STDOUT are not supported as valid arguments).
-B --extra-node-info=<sockets[:cores[:threads]]>	Restrict node selection to nodes with at least the specified number of sockets, cores per socket and/or threads per core.
--get-user-env[=timeout] [mode]	This option will tell sbatch to retrieve the login environment variables for the user specified in the --uid option.
--gid=<group>	If sbatch is run as root, and the --gid option is used, submit the job with group's group access permissions.
--gpu-bind=[verbose,]<type>	Bind tasks to specific GPUs

--gpu-freq=[<type=value>]	Request that GPUs allocated to the job are configured with specific frequency values.
-G, --gpus=[type=:]<number>	Specify the total number of GPUs required for the job
--gpus-per-node=[type:]<number>	Specify the number of GPUs required for the job on each node included in the job's resource allocation
--gpus-per-socket=[type:]<number>	Specify the number of GPUs required for the job on each socket included in the job's resource allocation.
--gpus-per-task=[type:]<number>	Specify the number of GPUs required for the job on each task to be spawned in the job's resource allocation
--gres=<list>	Specifies a comma delimited list of generic consumable resources.
--gres-flags=<type>	Specify generic resource task binding options.
-h, --help	Display help information and exit.
--hint=<type>	Bind tasks according to application hints.
-H, --hold	Specify the job is to be submitted in a held state (priority of zero).
--ignore-pbs	Ignore any "#PBS" options specified in the batch script.
-i, --input=<filename pattern>	Instruct Slurm to connect the batch script's standard input directly to the file name specified in the "filename pattern".
-J, --job-name=<jobname>	Specify a name for the job allocation.
--kill-on-invalid-dep=<yes no>	If a job has an invalid dependency and it can never run this parameter tells Slurm to terminate it or not.
-L, --licenses=<license>	Specification of licenses (or other resources available on all nodes of the cluster) which must be allocated to this job.
--mail-type=<type>	Notify user by email when certain event types occur.
--mail-user=<user>	User to receive email notification of state changes as defined by --mail-type.
--mcs-label=<mcs>	This parameter is a group among the groups of the user.

--mem=<size[units]>	Specify the real memory required per node.
--mem-bind=[{quiet,verbose},]t ype	Bind tasks to memory.
--mem-per-cpu=<size[units]>	Minimum memory required per allocated CPU.
--mem-per-gpu=<size[units]>	Minimum memory required per allocated GPU.
--mincpus=<n>	Specify a minimum number of logical cpus/processors per node.
--network=<type>	Specify information pertaining to the switch or network.
--nice[=adjustment]	Run the job with an adjusted scheduling priority within Slurm.
-k, --no-kill	Do not automatically terminate a job if one of the nodes it has been allocated fails.
--no-requeue	Specifies that the batch job should never be requeued under any circumstances.
-F, --nodefile=<node file>	Much like --nodelist, but the list is contained in a file of name node file.
-w, --nodelist=<node name list>	Request a specific list of hosts.
-N, --nodes=<minnodes[-maxnodes]>	<p>Request that a minimum of minnodes nodes be allocated to this job.</p> <p>In Slurm V23.02, extended the the --nodes syntax to allow for a list of valid node counts to be allocated to the job. This also supports a "step count" value (e.g., --nodes=20-100:20 is equivalent to --nodes=20,40,60,80,100) which can simplify the syntax when the job needs to scale by a certain "chunk" size.</p>
-n, --ntasks=<number>	This option advises the Slurm controller that job steps run within the allocation will launch a maximum of number tasks and to provide for sufficient resources.
--ntasks-per-core=<ntasks>	Request the maximum ntasks be invoked on each core.
--ntasks-per-gpu=<ntasks>	Request that there are ntasks tasks invoked for every GPU

--ntasks-per-node=<ntasks>	Request that ntasks be invoked on each node.
--ntasks-per-socket=<ntasks>	Request the maximum ntasks be invoked on each socket.
--open-mode=append truncate	Open the output and error files using append or truncate mode as specified.
-o, --output=<filename pattern>	Instruct Slurm to connect the batch script's standard output directly to the file name specified in the "filename pattern". In Slurm V23.02, we now will automatically create the directory and filename if they do not already exist.
-O, --overcommit	Overcommit resources.
-s, --oversubscribe	The job allocation can over-subscribe resources with other running jobs.
--parsable	Outputs only the job id number and the cluster name if present.
-p, --partition=<partition_names>	Request a specific partition for the resource allocation.
--power=<flags>	Comma separated list of power management plugin options.
--prefer=<list>	Nodes can have features assigned to them by the Slurm administrator. Users can specify which of these features are desired but not required by their job using the prefer option.
--priority=<value>	Request a specific job priority.
--profile=<all none [energy[, task[, lustre[, network]]]]>	Enables detailed data collection by the acct_gather_profile plugin.
--propagate[=rlimit[,rlimit...]]	Allows users to specify which of the modifiable (soft) resource limits to propagate to the compute nodes and apply to their jobs.
-q, --qos=<qos>	Request a quality of service for the job.
-Q, --quiet	Suppress informational messages from sbatch.
--reboot	Force the allocated nodes to reboot before starting the job.
--requeue	Specifies that the batch job should eligible to being requeue.

--reservation=<name>	Allocate resources for the job from the named reservation.
--signal=[B:]<sig_num>[@<sig_time>]	When a job is within sig_time seconds of its end time, send it the signal sig_num.
--sockets-per-node=<sockets>	Restrict node selection to nodes with at least the specified number of sockets.
--spread-job	Spread the job allocation over as many nodes as possible and attempt to evenly distribute tasks across the allocated nodes.
--switches=<count>[@<max-time>]	When a tree topology is used, this defines the maximum count of switches desired for the job allocation and optionally the maximum time to wait for that number of switches.
--test-only	Validate the batch script and return an estimate of when a job would be scheduled to run given the current job queue and all the other arguments specifying the job requirements.
--thread-spec=<num>	Count of specialized threads per node reserved by the job for system operations and not used by the application.
--threads-per-core=<threads>	Restrict node selection to nodes with at least the specified number of threads per core.
-t, --time=<time>	Set a limit on the total run time of the job allocation.
--time-min=<time>	Set a minimum time limit on the job allocation.
--tmp=<size[units]>	Specify a minimum amount of temporary disk space per node.
--tres-per-task=<list>	<p>Specifies a comma-delimited list of trackable resources required for the job on each task to be spawned in the job's resource allocation. The format of each entry on the list is "name[:type]:count)". The name is that of the trackable resource. The count is the number of those resources with a default value of 1.</p> <p>The count can have a suffix of "k" or "K" (multiple of 1024), "m" or "M" (multiple of 1024 x 1024), "g" or "G" (multiple of 1024 x 1024 x 1024), "t" or "T" (multiple of 1024 x 1024 x 1024 x 1024), "p" or "P" (multiple of 1024 x 1024 x 1024 x 1024 x 1024).</p>

	The specified resources will be allocated to the job on each node. The available trackable resources are configurable by the system administrator. NOTE: Invalid TRES for --tres-per-task include bb,billing,energy,fs,mem,node,pages,vmem.
--uid=<user>	Attempt to submit and/or run a job as user instead of the invoking user id.
--usage	Display brief help message and exit.
--use-min-nodes	If a range of node counts is given, prefer the smaller count.
-v, --verbose	Increase the verbosity of sbatch's informational messages.
-V, --version	Display version information and exit.
-W, --wait	Do not exit until the submitted job terminates.
--wait-all-nodes=<value>	Controls when the execution of the command begins.
--wckey=<wckey>	Specify wckey to be used with job.
--wrap=<command string>	Sbatch will wrap the specified command string in a simple "sh" shell script, and submit that script to the slurm controller.

These values can not only be used on the `sbatch` command line, but can also be put in a job submit script using the `#SBATCH` syntax:

```
#!/bin/bash
#SBATCH -n 1          # Number of cores
#SBATCH -N 1          # Ensure that all cores are on one machine
#SBATCH -t 0-00:10     # Runtime in D-HH:MM, minimum of 10 minutes
#SBATCH -p debug       # Partition to submit to
#SBATCH --mem=100       # Memory pool for all cores (see also --mem-per-cpu)
#SBATCH -o myoutput_%j.out # File to which STDOUT will be written, %j inserts jobid
#SBATCH -e myerrors_%j.err # File to which STDERR will be written, %j inserts jobid

perl -e 'print "Hi there. \n"'
echo "This was run on $SLURM_JOB_NODELIST"
```

So, assuming the job script is called `hithere.bash`, when you submit the job as follows:

```
$ sbatch hithere.bash
```

The output file (called `myoutput_<JOBID>.out`) would look like this:

```
Hi there.  
This was run on node00
```

If there were any errors, they would appear in myerrors_<JOBID>.err.

A brief explanation of the job script entries:

In general, the script is composed of 4 parts:

- the `#!/bin/bash` line allows the script to be run as a bash script
- the `#SBATCH` lines are technically bash comments, but they set various parameters for the SLURM scheduler
- the command line itself, in this case calling perl and having it print a message, then echoing the SLURM Variable called `SLURM_JOB_NODELIST`, which we will talk about later

The `#SBATCH` lines shown above set key parameters. The Slurm controller copies many environment variables from your current session to the compute host where the script is run including PATH and your current working directory. As a result, you can specify files relative to your current location (e.g. `./home/fred/myscripts`).

#SBATCH -n 1

This line sets the number of cores that you need to run the program. If this parameter is omitted, Slurm defaults to `-n 1`.

#SBATCH -N 1

This line requests that the cores be assigned all on node. If you know your code uses a message passing protocol like MPI, you can adjust this number to the total number of nodes you request for your MPI job. Slurm makes no default assumptions on this parameter -- if you request more than one core (`-n > 1`) and you forget this parameter, your job may be scheduled across nodes; and unless your job is MPI (multinode) aware, your job will probably run slowly, as it is oversubscribed on the master node and wasting resources on the other(s).

#SBATCH -t 0-01:00

This line specifies the running time for the job in minutes. Other acceptable time formats include "minutes", "minutes:seconds", "hours:minutes:seconds", "days-hours", "days-hours:minutes" and "days-hours:minutes:seconds". If your job runs longer than the value you specify here, it will be canceled by a SIGHUP/SIGKILL by the slurmd on the computer node(s). The default time values "Out of the box" is INFINITY. The Slurm administrator can assign a default time, however, so if you forget to assign one, the system can assign one for you

#SBATCH -p batch

This line specifies the Slurm partition (AKA queue) under which the script will be run. The Slurm administrator may have created multiple partitions other than the debug (default) partition. Most often, a partition (or queue) is a collection of compute nodes, usually grouped together by type, location in the racks, or feature..

#SBATCH --mem=100

Most clusters require that you specify the amount of memory (in MB) that you will be using for your job. Accurate specifications allow jobs to be run with maximum efficiency on the system. There are two main options, --mem-per-cpu and --mem. The --mem option specifies the total memory pool for one or more cores, and is the recommended option to use. If you must do work across multiple compute nodes (e.g. MPI code), then you must use the --mem-per-cpu option, as this will allocate the amount specified for each of the cores you're requesting, whether it is on one node or multiple nodes. If this parameter is omitted, then the Slurm administrator can configure a default for you.

```
#SBATCH -o myoutput_%j.out
```

This line specifies the file to which standard out will be appended. If a relative file name is used, it will be relative to your current working directory. The %j in the filename will be substituted by the JobID at runtime. If this parameter is omitted, any output will be directed to a file named slurm-JOBID.out in the current directory.

```
#SBATCH -e myerrors_%j.err
```

This line specifies the file to which standard error will be appended. Slurm submission and processing errors will also appear in the file. The %j in the filename will be substituted by the JobID at runtime. If this parameter is omitted, any output will be directed to a file named slurm-JOBID.err in the current directory.

```
#SBATCH --test-only
```

While not shown in the above job script, adding this option to your script will tell the scheduler to return information on what would happen if you submit this job. This is a practical way to determine if your script is viable as well as give a rough estimate of how long it would take to schedule in the current queue load.

```
#SBATCH --account=some_lab
```

If you belong to more than one account, specify it on this line. When you submit with the -A (or --account=), Slurm will make sure the job is assigned to a particular charging entity. Also, if Fairshare is implemented, it can be used to assign priority values to your jobs.d

Slurm Completion Support

Slurm also has completion help files included in the source code. There is auto-completion help for the Slurm submission (and other) commands at the Bash command line, as well as for the Vim syntax hi-lighting in the writing of batch files. See the "[Slurm Completion Discussion](#)" section of the Appendix for details.

Submitting Jobs using the `srun` Command

Although batch submission is the most used way to take full advantage of scheduled compute resources, you can also submit foreground/interactive jobs to the cluster. These can be useful for things like:

- Iterative data exploration at the command line (not recommended anymore for interactive workload. Instead, should use salloc)
- RAM intensive graphical applications like MATLAB or SAS
- Interactive "console tools" like R and iPython
- Significant software development and compiling efforts

An interactive job differs from a batch job in one important aspect: jobs should be initiated with the `srun` command instead of `sbatch`. This command:

```
$ srun -p debug --pty --mem 500 -t 0-06:00 /bin/bash
```

will start a command line shell (/bin/bash) on the debug queue with 500 MB of RAM for 6 hours; 1 core on 1 node is assumed as these parameters (-n 1 -N 1) were left out. When the interactive session starts, you will notice that you are no longer on a login node, but rather one of the compute nodes dedicated to this queue. The --pty option allows the session to act like a standard terminal.

```
$ srun -p debug --pty --x11=first --mem 4000 -t 0-06:00 /bin/bash
```

In this case, we've asked for more memory because we plan to run MATLAB which requires a larger memory footprint. The --x11-first option allows XWindows to operate between the login and compute nodes.

Troubleshooting jobs with srun

When diagnosing jobs that do not run, an option is to use the –slurmd-debug=<level> option:

Specify a debug level for slurmd(8). The level may be specified either as an integer value between 2 [error] and 6 [debug2], or as one of the SlurmdDebug tags.

error	Log only errors
info	Log errors and general informational messages
verbose	Log errors and verbose informational messages
debug	Log errors and verbose informational messages and debugging messages
debug2	Log errors and verbose informational messages and more debugging messages

The slurmd debug information is copied onto the stderr of the job. By default only errors are displayed. This option applies to job and step allocations.

Best Practices, Large Job Counts

Consider putting related work into a single Slurm job with multiple job steps both for performance reasons and ease of management. Each Slurm job can contain a multitude of job steps and the overhead in Slurm for managing job steps is much lower than that of individual jobs.

Job arrays are an efficient mechanism of managing a collection of batch jobs with identical resource requirements. Most Slurm commands can manage job arrays either as individual elements (tasks) or as a single entity (e.g. delete an entire job array in a single command).

MPI

MPI use depends upon the type of MPI being used. There are three fundamentally different modes of operation used by these various MPI implementation.

1. Slurm directly launches the tasks and performs initialization of communications through the PMI2 or PMIx APIs. (Supported by most modern MPI implementations.)
2. Slurm creates a resource allocation for the job and then mpirun launches tasks using Slurm's infrastructure (older versions of OpenMPI).

3. Slurm creates a resource allocation for the job and then mpirun launches tasks using some mechanism other than Slurm, such as SSH or RSH. These tasks initiated outside of Slurm's monitoring or control. Slurm's epilog should be configured to purge these tasks when the job's allocation is relinquished. The use of pam_slurm_adopt is also strongly recommended.

Job Submission Exercises

In this set of labs, you will learn how to submit jobs to a cluster in Slurm using `srun`, `salloc`, and `sbatch` commands

Exercise 1: Submit a job with `srun`

- As `fred`, view the nodes in the cluster, their configuration, and their state for this training cluster:

```
su - fred  
sinfo -Nel -pdebug
```

Note:

- `-N` means print information in a node-oriented format with one line per node and partition
- `-e` means exact node configuration, not group'd together
- `-l` means long output, print more detailed information
- `-p` means show the nodes in the partition called “debug”

You should see:

```
Fri Dec 30 13:21:34 2022  
NODELIST NODES PARTITION STATE CPUS S:C:T MEMORY TMP_DISK WEIGHT AVAIL_FE REASON  
node00 1 debug* idle 4 1:4:1 15967 0 1 (null) none  
node01 1 debug* idle 4 1:4:1 15967 0 1 (null) none  
node02 1 debug* idle 4 1:4:1 15967 0 1 (null) none  
node03 1 debug* idle 4 1:4:1 15967 0 1 (null) none  
node04 1 debug* idle 4 1:4:1 15967 0 1 (null) none  
node05 1 debug* idle 4 1:4:1 15967 0 1 (null) none  
node06 1 debug* idle 4 1:4:1 15967 0 1 (null) none  
node07 1 debug* idle 4 1:4:1 15967 0 1 (null) none  
node08 1 debug* idle 4 1:4:1 15967 0 1 (null) none  
node09 1 debug* idle 4 1:4:1 15967 0 1 (null) none
```

NOTE: Your MEMORY amount may differ depending on which Cloud Platform this training course is being hosted on

Basically there are 10-nodes in the “debug” partition in this cluster. Each node has 4 cores and 16GB RAM (Memory reported changes depending on how much is discovered during installation. So your mileage may vary on the MEMORY amount). They are all currently in the idle state. The asterisk (*) indicates that debug is the default partition

- Using the `srun` command, submit a job that will run on 5 of the cluster nodes launching the Linux `hostname` command:

```
srun -N5 hostname
```

It should return the following:

```
node01  
node04  
node00  
node02  
node03
```

3. Again using the `srun` command, submit a job that will run on 10 cores (of the 40 installed in this cluster) launching the Linux `hostname` command:

```
srun -n10 hostname
```

It should return the following:

```
node02  
node02  
node01  
node01  
node01  
node01  
node00  
node00  
node00  
node00
```

Note the distribution. On each of the first 2 nodes, all 4 of the cores were allocated. Since we asked for 10 tasks with the `-n` switch (a task is a core by default unless modified by policy or command), the first 2 nodes filled up and the last 2 `hostname` commands ran on the third node (node02).

4. In some cases, jobs may want more than one cpu per task. In this case, you can use the `-c` switch. In essence, asking for X number of cores per task. It's easy to see the allocation and distribution using the `--cpu-bind=verbose` switch:

```
srun -n1 -c2 --cpu-bind=verbose hostname 1>/dev/null
```

You should see:

```
cpu-bind=MASK - node00, task 0 0 [375]: mask 0x3 set
```

NOTE: The CPU mask of 0x3 means cpu's 0 and 1 are invoked. So for task 0, CPUs 0 and 1 were engaged in running the `hostname` command.

5. Of course, in addition to cores/CPUs requested, you can request other allocation units, such as memory or GPUs. Try submitting a 2-GPU and 1G job with the 2 CPUs/task as you did in the previous step. But this time, look for the allocated units in the `scontrol` command that accompany the job request

```
srun -n1 -c2 --gres=gpu:2 --mem=1000 hostname  
scontrol show job
```

Should return:

```

JobId=4 JobName=hostname
  UserId=fred(1010) GroupId=users(100) MCS_label=N/A
  Priority=4294901756 Nice=0 Account=bedrock QOS=normal
  JobState=COMPLETED Reason=None Dependency=(null)
  Requeue=1 Restarts=0 BatchFlag=0 Reboot=0 ExitCode=0:0
  RunTime=00:00:01 TimeLimit=UNLIMITED TimeMin=N/A
  SubmitTime=2023-01-03T08:20:55 EligibleTime=2023-01-03T08:20:55
  AccrueTime=Unknown
  StartTime=2023-01-03T08:20:55 EndTime=2023-01-03T08:20:56
  Deadline=N/A
  SuspendTime=None SecsPreSuspend=0 LastSchedEval=2023-01-
  03T08:20:55 Scheduler=Main
    Partition=debug AllocNode:Sid=docker-scale-out-login-1:172
    ReqNodeList=(null) ExcNodeList=(null)
    NodeList=node00
    BatchHost=node00
    NumNodes=1 NumCPUs=2 NumTasks=1 CPUs/Task=2 ReqB:S:C:T=0:0:2:2
    TRES=cpu=2,mem=1000M,node=1,billing=2,gres/gpu=2,gres/gpu:gtx=2
    Socks/Node=* NtasksPerN:B:S:C=0:0:2:2 CoreSpec=*
    MinCPUSNode=2 MinMemoryNode=1000M MinTmpDiskNode=0
    Features=(null) DelayBoot=00:00:00
    OverSubscribe=OK Contiguous=0 Licenses=(null) Network=(null)
    Command=hostname
    WorkDir=/home/fred
    Power=
TresPerNode=gres:gpu:2

```

Note: Look for the **MinMemoryNode=1000M** value, and **TresPerNode=gres:gpu:2** values

6. This information is also stored in the Slurm database. Using the `sacct` command, query the database to see the allocated units as stored in the job history record:

sacct --format=jobid,account,user,allocGRES%70

You should now see:

JobID	Account	User	AllocTRES
<hr/>			
4	bedrock	fred	billing=2,cpu=2,gres/gpu:gtx=2,gres/gpu=2,mem=1000M,node=1
4.extern	bedrock		billing=2,cpu=2,gres/gpu:gtx=2,gres/gpu=2,mem=1000M,node=1
4.0	bedrock		cpu=2,gres/gpu:gtx=2,gres/gpu=2,mem=1000M,node=1

Note: The **mem=1000M** is the 1GB memory request, and **gpu=2** is the GPU request

Exercise 2: Submit an interactive job with salloc

1. Before submitting an interactive job, as root user, change a policy in `/etc/slurm/slurm.conf`: If not already changed, modify:

```
LaunchParameters=enable_nss_slurm  
To:  
LaunchParameters=enable_nss_slurm,use_interactive_step
```

Restart the scheduler:

```
/lab_scripts/restart.sh
```

2. Now as fred, submit a one-node, 1GB, 2GPU, 10-minute interactive job to node03:

```
salloc -w node03 --mem=1000 --gres=gpu:2 -t10
```

NOTE: This allocation shell has environment variables set which will impact the execution of `srun`. You can see the environment (exported) variables of the current shell by using the bash shell builtin command `env`.

3. View the Slurm environment variables of the allocation shell:

```
env | grep ^SLURM | sort
```

Should return:

```
SLURMD_NODENAME=node03
SLURM_CLUSTER_NAME=cluster
SLURM_CONF=/etc/slurm/slurm.conf
SLURM_CPUS_ON_NODE=1
SLURM_GPUS_ON_NODE=2
SLURM_GTIDS=0
SLURM_JOBID=7
SLURM_JOB_ACCOUNT=bedrock
SLURM_JOB_CPUS_PER_NODE=1
SLURM_JOB_CPUS_PER_NODE_PACK_GROUP_0=1
SLURM_JOB_GID=100
SLURM_JOB_GPUS=0
SLURM_JOB_ID=7
SLURM_JOB_NAME=interactive
SLURM_JOB_NODELIST=node03
SLURM_JOB_NUM_NODES=1
SLURM_JOB_PARTITION=debug
SLURM_JOB_QOS=normal
SLURM_JOB_UID=1010
SLURM_JOB_USER=fred
SLURM_LAUNCH_NODE_IPADDR=2001:db8:1:1::1:5
SLURM_LOCALID=0
SLURM_MEM_PER_NODE=1000
SLURM_MPI_TYPE=pmix
SLURM_NNODES=1
SLURM_NODEID=0
SLURM_NODELIST=node03
SLURM_NODE_ALIASES=(null)
SLURM_PMIXP_ABORT_AGENT_PORT=44429
SLURM_PMIX_MAPPING_SERV=(vector,(0,1,1))
SLURM_PRIO_PROCESS=0
SLURM_PROCID=0
SLURM_PTY_PORT=46105
SLURM_PTY_WIN_COL=205
SLURM_PTY_WIN_ROW=32
SLURM_SRUN_COMM_HOST=2001:db8:1:1::1:5
SLURM_SRUN_COMM_PORT=44685
SLURM_STEPID=4294967290
SLURM_STEP_ID=4294967290
SLURM_STEP_LAUNCHER_PORT=44685
SLURM_STEP_NODELIST=node03
SLURM_STEP_NUM_NODES=1
SLURM_STEP_NUM_TASKS=1
SLURM_STEP_TASKS_PER_NODE=1
SLURM_SUBMIT_DIR=/home/fred
SLURM_SUBMIT_HOST=login
SLURM_TASKS_PER_NODE=1
SLURM_TASK_PID=515
SLURM_TOPOLOGY_ADDR=node03
SLURM_TOPOLOGY_ADDR_PATTERN=node
SLURM_WORKING_CLUSTER=cluster:mgmtnode:6817:9728:109
```

4. Use `srun` to launch a command in the allocation:

```
srun ping -4 -c 10 google.com
```

The Linux ping command should run through 10 iterations.

5. Issue a hostname command:

```
hostname
```

You should see:

```
node03
```

6. Confirm you have the SLURM environment variable for NODELIST on the node03 node:

```
env|grep SLURM_Nodelist
```

Should return:

```
SLURM_Nodelist=node03
```

7. View the job in the queue:

```
squeue
```

Should return:

JOBID	PARTITION	NAME	USER	ST	TIME	NODES	NODELIST (REASON)
1	debug	interact	fred	R	3:10	1	node03

8. Verify a slurmstepd is running that shepherds the bash task:

```
pgrep -afl stepd
```

Should return something similar to:

```
230 slurmstepd: [1.extern]
237 slurmstepd: [1.interactive]
```

NOTE: The “extern” step accounts for all resources usage by that job outside of slurm. An example of a process belonging to a slurm job, but not directly controlled by slurm are ssh sessions. If you ssh into a node where one of your jobs runs, your session will be placed into the context of the job (and you will be limited to your available resources by cgroups, if that is set up). And all calculations you do in that ssh session will be accounted for in the .extern job step. The 1.0 step refers to the slurmstepd tracking the step spawned by srun. You will see a stapID with each srun invocation.

9. End the job by exiting the interactive shell on node03, as well as the allocation shell:

```
exit
```

Should return something similar to:

```
exit  
salloc: Relinquishing job allocation 1
```

Exercise 3: Using sbatch, create a resource allocation and submit a job script to it

In this exercise, you will request an allocation through `sbatch`, and once granted, the script will run on the allocated resources.

1. **As the Fred user**, create a file in fred's home directory called `gethostname.sh`. Add the following to it (It is also found in the `/lab_scripts` folder so you can just copy it to Fred's home directory):

```
#!/bin/bash

#SBATCH -N10  # Run this across 10 nodes
#SBATCH -n20  # Run 20 tasks
#SBATCH -pdebug #Run in the debug partition
#SBATCH --mail-user=fred@localhost # Send mail to fred
#SBATCH --mail-type=BEGIN,END,FAIL # Send mail on begin, end, fail
#SBATCH -t 1  # Schedule 1 minute wallclock
#SBATCH -o gethostname_%j.out # output goes to gethostname_<JOBID>.out
#SBATCH -e gethostname_%j.err # error goes to gethostname_<JOBID>.err

srun -l hostname | sort -h
```

2. Submit the job using `sbatch`:

`sbatch gethostname.sh`

3. Record the JOBID.

4. Look at the output and error files:

`cat gethostname_<JOBID>.out | sort -h`

Should look like this (My JOBID is 2):

```
0: node00
1: node00
2: node00
3: node00
4: node01
5: node01
6: node01
7: node01
8: node02
9: node02
10: node02
11: node02
12: node03
13: node03
14: node04
15: node05
16: node06
17: node07
18: node08
19: node09
```

This job ran across all 10 nodes and utilized a total of 20 tasks: 2 cores per node * 10 nodes = 20 tasks.

NOTE: Your output may differ, depending on how the allocation was made. I.e, some tasks could be assigned to some nodes and not others.

5. If you look at Fred's mail (using the `mail` command as **Fred**) you will see the begin and end entries:
mail

You should see something like this:

```
Heirloom Mail version 12.5 7/5/10. Type ? for help.
"/var/spool/mail/fred": 2 messages 2 new
>N 1 slurm@login.local Thu Mar 26 16:55 18/672 "Slurm Job_id=2 Name=gethostname.sh Began, Queued
time 00:00:01"
N 2 slurm@login.local Thu Mar 26 16:55 19/693 "Slurm Job_id=2 Name=gethostname.sh Ended, Run
time 00:00:00, COMPLETED, ExitCode 0"
&
```

Exercise 4: Using `salloc`, create a resource allocation and submit a command to it

Using `salloc`, create an allocation and when assigned the allocation, submit a command that will run in the allocated resources.

This lab requires a file called `whereami.c`. It is found in the `/lab_scripts` folder on the Login Docker container. As root, change directory to `/lab_scripts` and compile it.

NOTE: If you are copying and pasting from this lab manual, and using vi or vim, use “:set paste” to preserve formatting. Compare this document with what you have in your .c file to make sure it is appropriately formatted

If you want, you can copy/paste `whereami.c` from this document (or, you can find this in `/lab_scripts` folder in the learning environment -- code begins on next page):

```

/***** test1.91.prog.c - Simple test program for SLURM regression test1.91.
* Reports SLURM task ID and the CPU mask,
* similar functionality to "taskset" command
*****
* Copyright (C) 2005 The Regents of the University of California.
* Produced at Lawrence Livermore National Laboratory (cf, DISCLAIMER).
* Written by Morris Jette <jette1@llnl.gov>
* CODE-OCEC-09-009. All rights reserved.
*
* This file is part of SLURM, a resource management program.
* For details, see <http://slurm.schedmd.com/>.
* Please also read the included file: DISCLAIMER.
*
* SLURM is free software; you can redistribute it and/or modify it under
* the terms of the GNU General Public License as published by the Free
* Software Foundation; either version 2 of the License, or (at your option)
* any later version.
*
* SLURM is distributed in the hope that it will be useful, but WITHOUT ANY
* WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS
* FOR A PARTICULAR PURPOSE. See the GNU General Public License for more
* details.
*
* You should have received a copy of the GNU General Public License along
* with SLURM; if not, write to the Free Software Foundation, Inc.,
* 51 Franklin Street, Fifth Floor, Boston, MA 02110-1301 USA.
\****

#define _GNU_SOURCE
#define __USE_GNU
#include <errno.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>

static char *_get_cpu_bindings()
{
    FILE *cpuinfo = fopen("/proc/self/status", "rb");
    char *line = 0;
    size_t size = 0;
    char *cpus = calloc(1024, sizeof(char));
    while(getdelim(&line, &size, '\n', cpuinfo) != -1) {
        if (strstr(line, "Cpus_")) {
            char *end = strstr(line, "\n");
            if (end)
                *end = '\0';
            sprintf(cpus + strlen(cpus), "%s%s", line, (cpus[0]) ? "" : "\t");
        }
    }
    free(line);
    fclose(cpuinfo);
    return cpus;
}

int main (int argc, char **argv)
{
    char *task_str;
    char *node_name;
    int task_id;

    /* On POE systems, MP_CHILD is equivalent to SLURM_PROCID */
    if (((task_str = getenv("SLURM_PROCID")) == NULL) &&
        ((task_str = getenv("MP_CHILD")) == NULL)) {
        fprintf(stderr, "ERROR: getenv(SLURM_PROCID) failed\n");
        exit(1);
    }
    node_name = getenv("SLURMD_NODENAME");
    task_id = atoi(task_str);
    printf("%d %s - %s\n", task_id, node_name, _get_cpu_bindings());

    if (argc > 1) {
        int sleep_time = strtol(argv[1], 0, 10);
        //printf("sleeping %d seconds\n", sleep_time);
        fflush(stdout);
        sleep(sleep_time);
    }
    exit(0);
}

```

1. Make sure you have completed the setup steps, in the summary, above.
2. As root, in the `/lab_scripts` folder, compile the program:
`gcc whereami.c -o whereami`
3. Copy it to the Slurm executable to `/usr/local/bin`:
`cp whereami /usr/local/bin`
4. Send it out to the rest of the compute nodes:
`pdcp whereami /usr/local/bin`

Submit a job using `salloc` and run `whereami` in the allocation

1. As fred, request an allocation of all 10 nodes and all 40 cores (each node has 4 cores or threads):
`salloc -N10 -n40 -pdebug`

When the allocation has been granted, you should see this:

```
salloc: Granted job allocation 3
salloc: Waiting for resource configuration
salloc: Nodes node[00-09] are ready for job
```

2. Validate that the allocation is what you asked for by running env:
`env | grep SLURM`

The output should show this:

```
[fred@mgmtnode ~]$ env|grep SLURM
SLURM_SUBMIT_DIR=/home/fred
SLURM_TASKS_PER_NODE=4 (x10)
SLURM_NNODES=10
SLURM_JOB_NODELIST=node[00-09]
SLURM_CLUSTER_NAME=cluster
SLURM_NODELIST=node[00-09]
SLURM_NTASKS=40
SLURM_JOB_CPUS_PER_NODE=4 (x10)
SLURM_JOB_NAME=interactive
SLURM_JOBID=3
SLURM_CONF=/etc/slurm/slurm.conf
SLURM_NODE_ALIASES=(null)
SLURM_JOB_QOS=normal
SLURM_JOB_NUM_NODES=10
SLURM_JOB_PARTITION=debug
SLURM_NPROCS=40
SLURM_SUBMIT_HOST=mgmtnode
SLURM_JOB_ACCOUNT=bedrock
SLURM_JOB_ID=3
```

Meaning you now have a 10-node, 40-core allocation (SLURM_NNODES=10, SLURM_NPROCS=40, etc)

- Now that the allocation has been made, you can **srun** the **whereami** command to get the distribution location:

```
srun -l whereami | sort -h
```

Note: the **-l** is the “list” switch for srun. It will list the tasks’ number in the left column. Then, it’s easy to sort the output

It should show this:

```
[fred@login ~]$ srun -l whereami | sort -h
0: 0 node00 - Cpus_allowed: 00000000,00000000,00000000,00000001 Cpus_allowed_list: 0
1: 1 node00 - Cpus_allowed: 00000000,00000000,00000000,00000002 Cpus_allowed_list: 1
2: 2 node00 - Cpus_allowed: 00000000,00000000,00000000,00000004 Cpus_allowed_list: 2
3: 3 node00 - Cpus_allowed: 00000000,00000000,00000000,00000008 Cpus_allowed_list: 3
4: 4 node01 - Cpus_allowed: 00000000,00000000,00000000,00000001 Cpus_allowed_list: 0
5: 5 node01 - Cpus_allowed: 00000000,00000000,00000000,00000002 Cpus_allowed_list: 1
6: 6 node01 - Cpus_allowed: 00000000,00000000,00000000,00000004 Cpus_allowed_list: 2
7: 7 node01 - Cpus_allowed: 00000000,00000000,00000000,00000008 Cpus_allowed_list: 3
8: 8 node02 - Cpus_allowed: 00000000,00000000,00000000,00000001 Cpus_allowed_list: 0
9: 9 node02 - Cpus_allowed: 00000000,00000000,00000000,00000002 Cpus_allowed_list: 1
10: 10 node02 - Cpus_allowed: 00000000,00000000,00000000,00000004 Cpus_allowed_list: 2
11: 11 node02 - Cpus_allowed: 00000000,00000000,00000000,00000008 Cpus_allowed_list: 3
12: 12 node03 - Cpus_allowed: 00000000,00000000,00000000,00000001 Cpus_allowed_list: 0
13: 13 node03 - Cpus_allowed: 00000000,00000000,00000000,00000002 Cpus_allowed_list: 1
14: 14 node03 - Cpus_allowed: 00000000,00000000,00000000,00000004 Cpus_allowed_list: 2
15: 15 node03 - Cpus_allowed: 00000000,00000000,00000000,00000008 Cpus_allowed_list: 3
16: 16 node04 - Cpus_allowed: 00000000,00000000,00000000,00000001 Cpus_allowed_list: 0
17: 17 node04 - Cpus_allowed: 00000000,00000000,00000000,00000002 Cpus_allowed_list: 1
18: 18 node04 - Cpus_allowed: 00000000,00000000,00000000,00000004 Cpus_allowed_list: 2
19: 19 node04 - Cpus_allowed: 00000000,00000000,00000000,00000008 Cpus_allowed_list: 3
20: 20 node05 - Cpus_allowed: 00000000,00000000,00000000,00000001 Cpus_allowed_list: 0
21: 21 node05 - Cpus_allowed: 00000000,00000000,00000000,00000002 Cpus_allowed_list: 1
22: 22 node05 - Cpus_allowed: 00000000,00000000,00000000,00000004 Cpus_allowed_list: 2
23: 23 node05 - Cpus_allowed: 00000000,00000000,00000000,00000008 Cpus_allowed_list: 3
24: 24 node06 - Cpus_allowed: 00000000,00000000,00000000,00000001 Cpus_allowed_list: 0
25: 25 node06 - Cpus_allowed: 00000000,00000000,00000000,00000002 Cpus_allowed_list: 1
26: 26 node06 - Cpus_allowed: 00000000,00000000,00000000,00000004 Cpus_allowed_list: 2
27: 27 node06 - Cpus_allowed: 00000000,00000000,00000000,00000008 Cpus_allowed_list: 3
28: 28 node07 - Cpus_allowed: 00000000,00000000,00000000,00000001 Cpus_allowed_list: 0
29: 29 node07 - Cpus_allowed: 00000000,00000000,00000000,00000002 Cpus_allowed_list: 1
30: 30 node07 - Cpus_allowed: 00000000,00000000,00000000,00000004 Cpus_allowed_list: 2
31: 31 node07 - Cpus_allowed: 00000000,00000000,00000000,00000008 Cpus_allowed_list: 3
32: 32 node08 - Cpus_allowed: 00000000,00000000,00000000,00000001 Cpus_allowed_list: 0
33: 33 node08 - Cpus_allowed: 00000000,00000000,00000000,00000002 Cpus_allowed_list: 1
34: 34 node08 - Cpus_allowed: 00000000,00000000,00000000,00000004 Cpus_allowed_list: 2
35: 35 node08 - Cpus_allowed: 00000000,00000000,00000000,00000008 Cpus_allowed_list: 3
36: 36 node09 - Cpus_allowed: 00000000,00000000,00000000,00000001 Cpus_allowed_list: 0
37: 37 node09 - Cpus_allowed: 00000000,00000000,00000000,00000002 Cpus_allowed_list: 1
38: 38 node09 - Cpus_allowed: 00000000,00000000,00000000,00000004 Cpus_allowed_list: 2
39: 39 node09 - Cpus_allowed: 00000000,00000000,00000000,00000008 Cpus_allowed_list: 3
```

Note: the MASK location is the bitmask for the core: 0x1 i core0 on CPU0 and 0x2 is core1 on CPU0, etc.

- Exit the allocation and end the job:

```
exit
```

Exercise 5: Finding the PIDs of running jobs

Often times, one needs to know the Process IDs of the applications running on the cluster. This could be for any reasons: Needing to clean up leftover failed processes, debugging using strace against the PID, etc.

In this exercise, you will use the scontrol command to determine the PIDs of your running job

Create job script that will run multiple programs on a compute node

1. As fred user, create a file called `test.sh` in his home directory, add the following to it:

```
#!/bin/bash

#SBATCH -N1
#SBATCH -n4
#SBATCH -wnode00
#SBATCH --mem=1000
#SBATCH -t10

srun -c1 --mem=250 -n1 --exclusive /bin/ping -c 120 -4 google.com&
srun -c1 --mem=250 -n1 --exclusive /bin/ping -c 120 -4 google.com&
srun -c1 --mem=250 -n1 --exclusive /bin/ping -c 120 -4 google.com&
srun -c1 --mem=250 -n1 --exclusive /bin/ping -c 120 -4 google.com&
wait
```

2. Submit the job:

`sbatch test.sh`

3. Find the JOBID using squeue:

`squeue`

You should see:

JOBID	PARTITION	NAME	USER	ST	TIME	NODES	NODELIST (REASON)
5	debug	test.sh	fred	R	4:02	1	node00

4. To see the individual job steps (each srun spawns a slurmstepd) using squeue:

`squeue -las`

You should see:

Thu Sep 23 18:50:06 2021						
STEPID	NAME	PARTITION	USER	TIME	NODELIST	
5.0	ping	debug	fred	0:04	node00	
5.1	ping	debug	fred	0:04	node00	
5.2	ping	debug	fred	0:04	node00	
5.3	ping	debug	fred	0:04	node00	
5.batch	batch	debug	fred	0:04	node00	
5.extern	extern	debug	fred	0:04	node00	

5. You can see the cpu load of the job by running sstat (which we will cover in later chapters):

`sstat 5` (My JOBID is 5, use your own for this exercise.)

You should see:

```

JobID MaxVMSize MaxVMSizenode MaxVMSizetask AveVMSize      MaxRSS
MaxRSSNode MaxRSSTask     AveRSS MaxPages MaxPagesNode   MaxPagesTask
AvePages      MinCPU MinCPUNode MinCPUTask     AveCPU    NTasks AveCPUFreq
ReqCPUFreqMin ReqCPUFreqMax ReqCPUFreqGov ConsumedEnergy  MaxDiskRead
MaxDiskReadNode MaxDiskReadTask  AveDiskRead MaxDiskWrite MaxDiskWriteNode
MaxDiskWriteTask AveDiskWrite TRESUsageInAve TRESUsageInMax TRESUsageInMaxNode
TRESUsageInMaxTask TRESUsageInMin TRESUsageInMinNode TRESUsageInMinTask
TRESUsageInTot TRESUsageOutAve TRESUsageOutMax TRESUsageOutMaxNode
TRESUsageOutMaxTask TRESUsageOutMin TRESUsageOutMinNode TRESUsageOutMinTask
TRESUsageOutTot
-----
-----
```

JobID	MaxVMSize	MaxVMSizenode	MaxVMSizetask	AveVMSize	MaxRSS
5.batch	295180K	node00	0	295180K	9676K
node00	0	9676K	0	node00	0
00:00.000	node00		00:00.000	1	2.30M
0	0	0	69902	node00	0
69902	29405	node00		0	29405
cpu=00:00:00,+ cpu=00:00:00,+ cpu=node00,energy+ cpu=00:00:00,fs/d+					
cpu=00:00:00,+ cpu=node00,energy+ cpu=00:00:00,fs/d+ cpu=00:00:00,+					
energy=0,fs/di+ energy=0,fs/di+ energy=node00,fs/d+ fs/disk=0					
energy=0,fs/di+ energy=node00,fs/d+ fs/disk=0 energy=0,fs/di+					

- There is a lot to see in that output, and you can narrow down by using just the **header fields** you want:
`sstat 5 --format JobID,MaxVMSize,MaxVMSizenode,TRESUsageOutMaxNode%30`

Should now show:

```

JobID MaxVMSize MaxVMSizenode          TRESUsageOutMaxNode
-----
```

JobID	MaxVMSize	MaxVMSizenode	TRESUsageOutMaxNode
5.0	342840K	node00	energy=node00,fs/disk=node00

This only shows JobID, Memory allocated to the job(s), the Nodes running the job(s), and the Trackable Resources (TRES) including energy and disk i/o.

Get the pids from the running job

Now that there is a job running, you can check the pids of that running job

- Using scontrol listpids:

```
ssh node00 scontrol listpids 5
```

Should show:

PID	JOBID	STEPID	LOCALID	GLOBALID
607	5	2	0	0
603	5	0	0	0
599	5	3	0	0
-1	5	extern	0	0
453	5	extern	-	-
632	5	extern	-	-
763	5	extern	-	-
768	5	extern	-	-
461	5	batch	0	0
462	5	batch	-	-
463	5	batch	-	-
464	5	batch	-	-
465	5	batch	-	-
474	5	batch	-	-
475	5	batch	-	-
478	5	batch	-	-
479	5	batch	-	-
605	5	1	0	0

2. You can see the expected termination time of any of the proc_id's by running:

```
ssh node00 scontrol pidinfo 599 (for step 3 of jobid 5)
```

Should show:

```
Slurm job id 5 ends at Wed Nov 25 23:30:28 2020
slurm_get_rem_time is 31535624
```

Meaning there is about **31535624 seconds (365 days) left** on this PID in this job allocation

3. Cancel all fred's running jobs:

```
scancel -u fred
```

Exercise 6 - Task Allocation, Placement, and Binding

The concept of placing applications on allocated resources takes into consideration the allocation (which is accomplished by `srun`, `sbatch`, or `salloc`), placement (which happens along with the allocation), and the binding of the tasks on the allocated resources. Binding tasks to allocated resources depends completely on how the job was submitted, and the **configuration of the task affinity plugin** within the controller.

1. SETUP: For this exercise to work correctly, the **SelectType plugin** needs to be set in a particular manner. (We'll talk about what this plugin does at a later time in training). For now, **Edit `/etc/slurm/slurm.conf` and confirm the following policies:**

```
SelectType=select/cons_tres
SelectTypeParameters=CR_CORE_MEMORY
```

2. If the plugin needed changing, then restart the scheduler (**as user root**):

`/lab_scripts/restart.sh`

3. **As root**, create a file called `alloc.c` and put the following in it (It's also included in `/lab_scripts`):

```
#define __GNU_SOURCE           /* See feature_test_macros(7) */

#include <sched.h>
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <sys/types.h>
#include <unistd.h>
#include <errno.h>

void print_affinity()
{
    pid_t pid = getpid();
    cpu_set_t mask = {0};
    int rc, i;
    if ((rc = sched_getaffinity(pid, sizeof(cpu_set_t), &mask))) {
        printf("error: %d", rc);
        exit (1);
    }

    printf("%d: ", pid);
    for (i = 0; i < 4; i++) {
        if (CPU_ISSET(i, &mask))
            printf("%d,", i);
    }
    printf("\n");
}

void reset_affinity()
{
    int rc, i;
    pid_t pid = getpid();
    cpu_set_t mask = {0};

    for (i = 0; i < 4; i++) {
        CPU_SET(i, &mask);
    }
    if ((rc = sched_setaffinity(pid, sizeof(cpu_set_t), &mask))) {
        printf("error: %d %d %s", rc, errno, strerror(errno));
        exit (1);
    }
    printf("reset affinity\n");
}

int main()
{
    int i, j;
    for (i = 1; ; i++) {
        print_affinity();
        for (j = 0; j < 999999999; j++);

        if ((i % 15) == 0)
            reset_affinity();
    }
}
```

4. **As root**, compile the program (You can compile it as root in `/lab_scripts` folder):

```
gcc -o alloc alloc.c
```

5. Still as root, copy the file to /usr/local/bin:
`cp alloc /usr/local/bin`
6. Still as root, copy the file to the compute nodes:
`pdcp alloc /usr/local/bin`
7. Open 2 other terminals.
8. As fred in the first window, in the login node container, launch alloc
`alloc`

You should see:

```
17488: 0,1,2,3,  
17488: 0,1,2,3,  
17488: 0,1,2,3,  
17488: 0,1,2,3,  
17488: 0,1,2,3,  
17488: 0,1,2,3,  
...  
(output truncated)
```

NOTE: The first column is the PID. Yours will be different than the one shown here, obviously. It is running on the login node that you logged into when you attached to the cluster. When you actually run alloc through srun, the scheduler will assign it to run on a cluster node, which means you'll see the PID on the assigned compute node. But we'll do that later. The 0,1,2,3 refer to the CPU IDs found on the computer running alloc.

9. In the next window, run top, pointing to the PID you see from the previous step:
`top -p <PID>`
10. Expand the CPU list by pressing the number 1.

You should see something similar to this:

```
top - 23:47:08 up 2 days, 18:33,  3 users,  load average: 1.22, 1.06, 0.88  
Tasks:  1 total,   1 running,   0 sleeping,   0 stopped,   0 zombie  
%Cpu0 :  7.8 us,  2.0 sy,  0.0 ni, 89.1 id,  0.0 wa,  0.0 hi,  1.0 si,  0.0 st  
%Cpu1 :  9.7 us,  2.0 sy,  0.0 ni, 87.9 id,  0.0 wa,  0.0 hi,  0.3 si,  0.0 st  
%Cpu2 :  8.5 us,  1.4 sy,  0.0 ni, 89.8 id,  0.0 wa,  0.0 hi,  0.3 si,  0.0 st  
%Cpu3 : 99.7 us,  0.3 sy,  0.0 ni,  0.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st  
KiB Mem : 16398028 total,    916000 free,   4726416 used, 10755612 buff/cache  
KiB Swap: 2097148 total,   2096112 free,      1036 used.  8943824 avail Mem  
  
          PID USER      PR  NI      VIRT      RES      SHR S %CPU %MEM     TIME+ COMMAND  
17488  fred       20   0     4224      692      620 R 100.0  0.0  3:35.79 alloc
```

NOTE: In this screenshot, the linux scheduler has pinned the load on Cpu3. It will probably rotate around among the Cpus a bit

11. Break the *alloc* program in the first window, and quit *top* in the 2nd window.

12. Now submit a job as **fred** launching alloc through srun:

```
srun -wnode00 -u -N1 -n1 -c2 alloc
```

Note:

- w<NODE> ... Requests the specific node(s) the scheduler to allocate this work to.
- N<1> Means 1 node is requested.
- u Means run this job unbuffered, so you can actually see the output the alloc command generates as it runs.
- n1 Means one core (task) is requested. The default allocation is 1 core=1 task in Slurm.
- c2 Means request 2 CPUs per task. We are only requesting 1 task (-n1), but in this example we want need 2 CPUs.

In step 6, you can see the CPUs allocated using the *alloc* command we compiled earlier.

In this cluster, remember there are 4 CPUs/node.

You should see:

```
<PID>: 0,1,  
...
```

(output truncated)

Meaning the hex values of CPUs 0,1, (CPU0 and CPU1) are turned on. Even with the “reset affinity” flag in the *alloc* program, it still only allows the 2 CPUs access to run the application. And, if you were to use taskset on **node00**’s PID to change the affinity mask, Cgroups will not allow all CPUs to be consumed.

13. In your 2nd terminal window, ssh to node00 as **root**:

```
ssh node00
```

(No password is required.)

14. On **Node00**, run top attaching to the pid that’s running alloc (it’s printing repeatedly in the terminal window:

```
top -p <PID>
```

You should see a similar output from the previous top.

15. Remember to expand the CPUs by pressing “1”.

16. In your **third terminal window**, ssh to node00:

```
ssh node00
```

17. Change the taskset value to 1:

```
taskset -p 0x1 <PID>
```

The affinity will change from <PID>: 0,1, to <PID>: 0, It will stay that way until the alloc program changes the task affinity back to f(all CPUs. In this case CPU0 and CPU1, since we submitted with the srun -c 2 flag.)

18. Do the same `taskset -p` but force a change to 0x2.

That will force the app to load up CPU1 (as seen in the top output)

Exercise 7: Use Bash completion for Slurm to Submit Jobs

Slurm Bash completion (more commonly called "tab completion") has been expanded from the previous release (it now works). In this exercise, you will submit sbatch and srun commands using Slurm Bash tab completion feature.

1. As the **fred** user on the **login** node, request an interactive allocation of two nodes:

```
salloc -N2
```

The `LaunchParameters=use_interactive_step` setting is configured, so you should notice that you are automatically logged in the 1st node of the two-node request. You should see something similar to this:

```
[fred@login ~]$ salloc -N2
salloc: Granted job allocation 2
salloc: Nodes node[00-01] are ready for job
[fred@node00 ~]$
```

2. Verify that tab completion is working. Enter the following command, and **in the following instructions where you see <Tab><Tab> (in red), you should actually tap the Tab key twice:**

```
srun -<Tab><Tab>
```

You will see something like this:

```
-A    -D    -G    -J    -M    -Q    -V    -Z    -d    -i    -m    -p    -s    -v
-B    -E    -H    -K    -N    -S    -W    -b    -e    -k    -n    -q    -t    -w
-C    -F    -I    -L    -O    -T    -X    -c    -h    -l    -o    -r    -u    -x
```

3. Continue typing by adding `-t` to the end, and then tap the **Tab** key a couple of times:

```
srun --t-<Tab><Tab>
```

You should see specific options available to the `srun` command, similar to the following:

```
--task-epilog=      --thread-spec=      --time=
--task-prolog=      --threads-per-core=   --tmp=
--tasks-per-node=   --threads=          --tres-per-task=
--test-only         --time-min=
```

4. Continue typing by adding `i` to the end of what you previously entered, and then tap the **Tab** key a couple of times, again:

```
srun --ti<Tab><Tab>
```

Notice that the option name is auto-populated with the `--time` string, since that is the only option available to the `srun` command that begins with `--ti`.

5. Complete and enter the command:

```
srun --time=1 hostname
```

You should see something similar to this:

```
node01  
node00
```

6. Using Slurm tab completion, facilitate entering the following more complex command, which is all-one-command that wraps around. **DO NOT COPY PASTE THE TEXT FROM THIS DOCUMENT. INSTEAD, USE TAB COMPLETION TO HELP YOU HAND-TYPE IT:**

```
srun --time=10 --nodelist=node0[0-1] --account=bedrock  
--cluster=cluster --comment=bash_completion --mem=1M --ntasks-per-  
core=1 --output=slurm_completion.out hostname
```

7. View the output:

```
cat slurm_completion.out
```

As expected output, you should see the two nodes which are allocated to this job:

```
node00  
node01
```

8. End the current interactive allocation of resource (effectively giving them back to the scheduler to be re-assigned) by exiting from the shell on the 1st node of the 2-node set:

```
exit
```

Notice that, per the prompt, you have been returned to the `login` node.

```
[fred@node00 ~]$ exit  
exit  
salloc: Relinquishing job allocation 1  
[fred@login ~]$
```

NOTE: Slurm tab completion also works for the `sbatch` command. In the following exercise step, there are various components of the `sbatch` command we haven't talked about yet, but tab completion helps you to fill out the parameters, even if you don't know them, or if you do not yet have them memorized.

9. **DO NOT COPY PASTE THE FOLLOWING COMMAND.** Use tab completion to enter the following complex set of parameters as a single `sbatch` command line:

```
sbatch      --(time of 2 minutes)  
           --(account called "bedrock")
```

```
--(an array of 40 elements, hint: --array=1-40)
--(execute this command on the cluster called "cluster")
--(give it 2 megabytes of memory)
--(ask for all 10 nodes of that cluster)
--(put a comment on the job: completion_is_great)
--(run this command in a wrapper: "srun hostname")
```

(Here's that command for reference):

```
sbatch --time=2 --account=bedrock --array=1-40 --cluster=cluster --mem=2 --nodes=10 --
comment=completion_is_great --wrap="srun hostname"
```

NOTE: You may have noticed, as you were typing that lengthy command, that Slurm can auto-fill some of the parameter values, as well as option names themselves. It does this by looking them up with a query back to the Slurm system.

10. Look up a cluster name for your current cluster. Enter the command in pieces, using tab completion:

```
srun --clus<Tab><Tab>
srun --cluster=<Tab><Tab>
```

You should see:

```
srun --cluster=cluster,
```

(You can tap backspace to remove the comma.)

11. Finish the command by completing and entering the following:

```
srun --cluster=cluster hostname
```

12. Look up the parts of the following command, one at a time, but all on the same line:

```
srun --cluster=
--account=
--nodelist=
```

(Tap the `y` key to see the full list, when prompted.)

Notice that you see a huge list like this (full list not shown):

```
...
cloud0143, cloud0316, cloud0489, cloud0662, cloud0835, cloud1008,
cloud0144, cloud0317, cloud0490, cloud0663, cloud0836, cloud1009,
cloud0145, cloud0318, cloud0491, cloud0664, cloud0837, cloud1010,
cloud0146, cloud0319, cloud0492, cloud0665, cloud0838, cloud1011,
cloud0147, cloud0320, cloud0493, cloud0666, cloud0839, cloud1012,
cloud0148, cloud0321, cloud0494, cloud0667, cloud0840, cloud1013,
cloud0149, cloud0322, cloud0495, cloud0668, cloud0841, cloud1014,
cloud0150, cloud0323, cloud0496, cloud0669, cloud0842, cloud1015,
cloud0151, cloud0324, cloud0497, cloud0670, cloud0843, cloud1016,
cloud0152, cloud0325, cloud0498, cloud0671, cloud0844, cloud1017,
cloud0153, cloud0326, cloud0499, cloud0672, cloud0845, cloud1018,
cloud0154, cloud0327, cloud0500, cloud0673, cloud0846, cloud1019,
cloud0155, cloud0328, cloud0501, cloud0674, cloud0847, cloud1020,
cloud0156, cloud0329, cloud0502, cloud0675, cloud0848, cloud1021,
cloud0157, cloud0330, cloud0503, cloud0676, cloud0849, cloud1022,
cloud0158, cloud0331, cloud0504, cloud0677, cloud0850, cloud1023,
cloud0159, cloud0332, cloud0505, cloud0678, cloud0851, cloud1024,
cloud0160, cloud0333, cloud0506, cloud0679, cloud0852, node00,
cloud0161, cloud0334, cloud0507, cloud0680, cloud0853, node01,
cloud0162, cloud0335, cloud0508, cloud0681, cloud0854, node02,
cloud0163, cloud0336, cloud0509, cloud0682, cloud0855, node03,
cloud0164, cloud0337, cloud0510, cloud0683, cloud0856, node04,
cloud0165, cloud0338, cloud0511, cloud0684, cloud0857, node05,
cloud0166, cloud0339, cloud0512, cloud0685, cloud0858, node06,
cloud0167, cloud0340, cloud0513, cloud0686, cloud0859, node07,
cloud0168, cloud0341, cloud0514, cloud0687, cloud0860, node08,
```

You can turn off the value lookup portion of Slurm tab completion by setting the `SLURM_COMP_VALUE` environment variable to '0'.

13. Press **Ctrl-c**, to interrupt the command.
14. Disable the lookup portion for option values (options names will remain available via tab completion):
`export SLURM_COMP_VALUE=0`
15. Now try the lookup via tab completion again, and run the following command:
`srun --nodelist=<Tab><Tab>`
`srun --nodelist=all<Backspace><Backspace><Backspace><Backspace>`
16. Change the command to look like this and enter it.
`srun --nodelist=node0[0-9] hostname`

As the final output, should see something like the following:

```
node04
node03
node00
node02
node06
node05
node09
node07
node01
node08
```

End of exercise.

Cleanup

1. Reset the Docker Slurm cluster environment back to its original, pristine state by **exiting the login Docker container back to the Ubuntu AWS prompt**. And **as ubuntu**:

```
cd ~/docker-scale-out  
make clean && make
```

Job Arrays Discussion

Job arrays offer a mechanism for submitting and managing collections of similar jobs quickly and easily. Job arrays with millions of tasks can be submitted in milliseconds (subject to configured size limits). All jobs must have the same initial options (e.g. size, time limit, etc.), however it is possible to change some of these options after the job has begun execution using the scontrol command specifying the JobID of the array or individual **ArrayJobID**.

```
$ scontrol update job=101 ...
$ scontrol update job=101_1 ...
```

Job arrays are only supported for batch jobs and the array index values are specified using the --array or -a option of the **sbatch** command. The option argument can be specific array index values, a range of index values, and an optional step size as shown in the examples below. Note that the minimum index value is zero and the maximum value is a Slurm configuration parameter (**MaxArraySize** minus one). Jobs which are part of a job array will have the environment variable **SLURM_ARRAY_TASK_ID** set to its array index value.

```
# Submit a job array with index values between 0 and 31
$ sbatch --array=0-31      -N1 tmp

# Submit a job array with index values of 1, 3, 5 and
$ sbatch --array=1,3,5,7 -N1 tmp

# Submit a job array with index values between 1 and 7
# with a step size of 2 (i.e. 1, 3, 5 and 7)
$ sbatch --array=1-7:2    -N1 tmp
```

A maximum number of simultaneously running tasks from the job array may be specified using a "%" separator. For example "**--array=0-15%4**" will limit the number of simultaneously running tasks from this job array to 4.

Job ID and Environment Variables

Job arrays will have two additional environment variable set:

SLURM_ARRAY_JOB_ID will be set to the first job ID of the array.

SLURM_ARRAY_TASK_ID will be set to the job array index value.

SLURM_ARRAY_TASK_COUNT will be set to the number of tasks in the job array.

SLURM_ARRAY_TASK_MAX will be set to the highest job array index value.

SLURM_ARRAY_TASK_MIN will be set to the lowest job array index value.

For example a job submission of this sort

```
sbatch --array=1-3 -N1 tmp
```

will generate a job array containing three jobs. If the sbatch command responds

```
Submitted batch job 36
```

then the environment variables will be set as follows:

```
SLURM_JOB_ID=36
SLURM_ARRAY_JOB_ID=36
SLURM_ARRAY_TASK_ID=1
SLURM_ARRAY_TASK_COUNT=3
SLURM_ARRAY_TASK_MAX=3
SLURM_ARRAY_TASK_MIN=1

SLURM_JOB_ID=37
SLURM_ARRAY_JOB_ID=36
SLURM_ARRAY_TASK_ID=2
SLURM_ARRAY_TASK_COUNT=3
SLURM_ARRAY_TASK_MAX=3
SLURM_ARRAY_TASK_MIN=1

SLURM_JOB_ID=38
SLURM_ARRAY_JOB_ID=36
SLURM_ARRAY_TASK_ID=3
SLURM_ARRAY_TASK_COUNT=3
SLURM_ARRAY_TASK_MAX=3
SLURM_ARRAY_TASK_MIN=1
```

All Slurm commands and APIs recognize the **SLURM_JOB_ID** value. Most commands also recognize the **SLURM_ARRAY_JOB_ID** plus **SLURM_ARRAY_TASK_ID** values separated by an underscore as identifying an element of a job array. Using the example above, "37" or "36_2" would be equivalent ways to identify the second array element of job 36. A set of APIs has been developed to operate on an entire job array or select tasks of a job array in a single function call. The function response consists of an array identifying the various error codes for various tasks of a job ID. For example the `job_resume2()` function might return an array of error codes indicating that tasks 1 and 2 have already completed; tasks 3 through 5 are resumed successfully, and tasks 6 through 99 have not yet started.

File Names

Two additional options are available to specify a job's **stdin**, **stdout**, and **stderr** file names: **%A** will be replaced by the value of **SLURM_ARRAY_JOB_ID** (as defined above) and **%a** will be replaced by the value of **SLURM_ARRAY_TASK_ID** (as defined above). The default output file format for a job array is "**slurm-%A_%a.out**". An example of explicit use of the formatting is:

```
sbatch -o slurm-%A_%a.out --array=1-3 -N1 tmp
```

which would generate output files names of this sort "**slurm-36_1.out**", "**slurm-36_2.out**" and "**slurm-36_3.out**". If these file name options are used without being part of a job array then "%A" will be replaced by the current job ID and "%a" will be replaced by 4,294,967,294 (equivalent to 0xffffffff or NO_VAL).

scancel Command Usage

If the job ID of a job array is specified as input to the scancel command then all elements of that job array will be cancelled. Alternately an array ID, optionally using regular expressions, may be specified for job cancellation.

```
# Cancel array ID 1 to 3 from job array 20
$ scancel 20_[1-3]

# Cancel array ID 4 and 5 from job array 20
$ scancel 20_4 20_5

# Cancel all elements from job array 20
$ scancel 20

# Cancel the current job or job array element (if job array)
if [[-z $SLURM_ARRAY_JOB_ID]]; then
    scancel $SLURM_JOB_ID
else
    scancel ${SLURM_ARRAY_JOB_ID}_${SLURM_ARRAY_TASK_ID}
fi
```

squeue Command Use

When a job array is submitted to Slurm, only one job record is created. Additional job records will only be created when the state of a task in the job array changes, typically when a task is allocated resources or its state is modified using the scontrol command. By default, the squeue command will report all of the tasks associated with a single job record on one line and use a regular expression to indicate the "array_task_id" values as shown below

```
$ squeue
JOBID      PARTITION  NAME  USER  ST   TIME  NODES NODELIST(REASON)
1080_[5-1024]  debug    tmp   mac   PD  0:00      1  (Resources)
1080_1        debug    tmp   mac   R   0:17      1  tux0
1080_2        debug    tmp   mac   R   0:16      1  tux1
1080_3        debug    tmp   mac   R   0:03      1  tux2
1080_4        debug    tmp   mac   R   0:03      1  tux3
```

An option of "--array" or "-r" has also been added to the squeue command to print one job array element per line as shown below. The environment variable "SQUEUE_ARRAY" is equivalent to including the "--array" option on the squeue command line.

```
$ squeue -r
JOBID PARTITION  NAME  USER  ST   TIME  NODES NODELIST(REASON)
1082_3    debug    tmp   mac   PD  0:00      1  (Resources)
1082_4    debug    tmp   mac   PD  0:00      1  (Priority)
1080      debug    tmp   mac   R   0:17      1  tux0
1081      debug    tmp   mac   R   0:16      1  tux1
1082_1    debug    tmp   mac   R   0:03      1  tux2
1082_2    debug    tmp   mac   R   0:03      1  tux3
```

The squeue --step/-s and --job/-j options can accept job or step specifications of the same format.

```
$ squeue -j 1234_2,1234_3
...
$ squeue -s 1234_2.0,1234_3.0
...
echo "This was run on $SLURM_JOB_NODELIST"
```

Two additional job output format field options have been added to squeue:

%F prints the array_job_id value

%K prints the array_task_id value

(all of the obvious letters to use were already assigned to other job fields).

scontrol Command Usage

Use of the **scontrol show job** option shows two new fields related to job array support. The JobID is a unique identifier for the job. The ArrayJobID is the JobID of the first element of the job array. The ArrayTaskID is the array index of this particular entry, either a single number or an expression identifying the entries represented by this job record (e.g. "5-1024"). Neither field is displayed if the job is not part of a job array. The optional job ID specified with the scontrol show job or scontrol show step commands can identify job array elements by specifying ArrayJobId and ArrayTaskId with an underscore between them (eg. <ArrayJobID>_<ArrayTaskId>).

The scontrol command will operate on all elements of a job array if the job ID specified is ArrayJobID. Individual job array tasks can be modified using the ArrayJobID_ArrayTaskID as shown below.

```
$ sbatch --array=1-4 -J array ./sleepme 86400
Submitted batch job 21845

$ squeue
  JOBID   PARTITION      NAME      USER      ST      TIME      NODES      NODELIST
 21845_1    canopo      array     david      R      0:13      1      dario
 21845_2    canopo      array     david      R      0:13      1      dario
 21845_3    canopo      array     david      R      0:13      1      dario
 21845_4    canopo      array     david      R      0:13      1      dario

$ scontrol update JobID=21845_2 name=arturo
$ squeue
  JOBID   PARTITION      NAME      USER      ST      TIME      NODES      NODELIST
 21845_1    canopo      array     david      R      17:03      1      dario
 21845_2    canopo      arturo    david      R      17:03      1      dario
 21845_3    canopo      array     david      R      17:03      1      dario
 21845_4    canopo      array     david      R      17:03      1      dario
```

The **scontrol hold, holdu, release, requeue, requeuehold, suspend and resume** commands can also either operate on all elements of a job array or individual elements as shown below.

```
$ scontrol suspend 21845
$ squeue
  JOBID   PARTITION      NAME      USER      ST      TIME      NODES      NODELIST
 21845_1    canopo      array     david      S 25:12      1      dario
 21845_2    canopo      arturo    david      S 25:12      1      dario
```

```

21845_3    canopo    array    david   S 25:12  1    dario
21845_4    canopo    array    david   S 25:12  1    dario

$ scontrol resume 21845
$ squeue
  JOBID PARTITION      NAME      USER ST TIME  NODES NODELIST
21845_1    canopo    array    david R 25:14  1    dario
21845_2    canopo    arturo   david R 25:14  1    dario
21845_3    canopo    array    david R 25:14  1    dario
21845_4    canopo    array    david R 25:14  1    dario

$ scontrol suspend 21845_3
$ squeue
  JOBID PARTITION      NAME      USER ST TIME  NODES NODELIST
21845_1    canopo    array    david R 25:14  1    dario
21845_2    canopo    arturo   david R 25:14  1    dario
21845_3    canopo    array    david P 25:14  1    dario
21845_4    canopo    array    david R 25:14  1    dario

$ scontrol resume 21845_3
$ squeue
  JOBID PARTITION      NAME      USER ST TIME  NODES NODELIST
21845_1    canopo    array    david R 25:14  1    dario
21845_2    canopo    arturo   david R 25:14  1    dario
21845_3    canopo    array    david R 25:14  1    dario
21845_4    canopo    array    david R 25:14  1    dario

```

Job Dependencies

A job which is to be dependent upon an entire job array should specify itself dependent upon the **ArrayJobID**. Since each array element can have a different exit code, the interpretation of the **afterok** and **afternotok** clauses will be based upon the highest exit code from any task in the job array.

When a job dependency specifies the job ID of a job array:

- The **after** clause is satisfied after all tasks in the job array start.
- The **afterany** clause is satisfied after all tasks in the job array complete.
- The **aftercorr** clause is satisfied after the corresponding task ID in the specified job has completed successfully (ran to completion with an exit code of zero).
- The **afterok** clause is satisfied after all tasks in the job array complete successfully.
- The **afternotok** clause is satisfied after all tasks in the job array complete with at least one tasks not completing successfully.

Examples of use are shown below:

```

$ squeue -j 1234_2,1234_3
...
$ squeue -s 1234_2.0,1234_3.0
...

```

```

# Wait for specific job array elements
  sbatch --depend=after:123_4 my.job
  sbatch --depend=afterok:123_4:123_8 my.job2

# Wait for entire job array to complete
  sbatch --depend=afterany:123 my.job

# Wait for corresponding job array elements
  sbatch --depend=aftercorr:123 my.job

# Wait for entire job array to complete successfully
  sbatch --depend=afterok:123 my.job

# Wait for entire job array to complete and at least one task fails
  sbatch --depend=afternotok:123 my.job

```

Other Command Use

The following Slurm commands do not currently recognize job arrays and their use requires the use of Slurm job IDs, which are unique for each array element: **sacct**, **sbatch**, **smap**, **sreport**, **sshare**, **sstat**, and **trigger**. The **sattach**, **sprio** and **sstat** commands have been modified to permit specification of either job IDs or job array elements. The **sview** command has been modified to permit display of a job's ArrayJobId and ArrayTaskId fields. Both fields are displayed with a value of "N/A" if the job is not part of a job array.

System Administration

A new configuration parameter has been added to control the maximum job array size: **MaxArraySize**. The smallest index that can be specified by a user is zero and the maximum index is MaxArraySize minus one. The default value of MaxArraySize is 1001. The maximum MaxArraySize supported in Slurm is 4000001. Be mindful about the value of MaxArraySize as job arrays offer an easy way for users to submit large numbers of jobs very quickly.

The sched/backfill plugin has been modified to improve performance with job arrays. Once one element of a job array is discovered to not be runnable or impact the scheduling of pending jobs, the remaining elements of that job array will be quickly skipped.

Slurm creates a single job record when a job array is submitted. Additional job records are only created as needed, typically when a task of a job array is started, which provides a very scalable mechanism to manage large job counts. Each task of the job array will have a unique Slurm "job_id", but all will have the same "array_job_id" value.

Job Array Exercises

In this set of labs, you will learn how to submit and monitor job arrays in Slurm.

Exercise 1: Submit a job array of 30 sub-jobs, taking input files as data

Job arrays are a handy tool for submitting many jobs that differ very little (usually only by the input they are receiving through a file). The idea is to create one batch script file to submit dozens to hundreds or thousands of jobs instead of creating an individual file for each job that only differs a bit.

In this example, we want 30 jobs to run in our job array. The job script is called `myslurmarray.sh`. The program we want to run 30 times is call `myprogram` and it requires an input file. In this example each input filename looks like this (`input1.dat`, `input2.dat`, `input3.dat` ... `input30.dat`). Each `inputX.dat` file has the content of it's `inputX` number. For example, `input1.dat` has the number 1 in it. `Input2.dat` has the number 2 in it, etc.

1. As fred, create a file called `myslurmarray.sh` and add the following to it:

```
#!/bin/bash
#SBATCH -J myprogram
#SBATCH -c 1
#SBATCH -N 1
#SBATCH -t 0-2:00
#SBATCH --array=1-30
#SBATCH -o myprogram%A_%a.out
# "%A" is replaced by the job ID and "%a" with the array index
#SBATCH -e myprogram%A_%a.err

./myprogram input${SLURM_ARRAY_TASK_ID}.dat

sleep 10
```

2. Generate the `myprogram` binary. Start by creating `myprogram.c` and add the following to it (you can copy this from the `/lab_scripts` folder to fred's home directory):

```
#include <stdio.h>
#include <unistd.h>

int main( int argc, char *argv[] )  {

    if( argc == 2 ) {
        printf("%s\n", argv[1]);
    }
    else if( argc > 2 ) {
        printf("Too many arguments supplied.\n");
    }
    else {
        printf("One argument expected.\n");
    }
    sleep(1);
}
```

3. Compile it by running:

```
gcc myprogram.c -o myprogram
```

4. Test the program by running it:

```
./myprogram
```

You should see:

```
One argument expected.
```

That is because this program expects at least 1 argument.

5. Try again by adding input to it:

```
./myprogram 100
```

You should now see:

```
100
```

6. Try it with 2 arguments:

```
./myprogram 100 200
```

You should see:

```
Too many arguments supplied.
```

7. Generate the 30 .dat files that will be used by the myprogram binary as part of the array job:

```
for x in {1..30}; do echo $x > input$x.dat ;done
```

8. Submit the array:

```
sbatch myslurmarray.sh
```

9. Watch the progress of the job array:

```
watch squeue -r
```

(see next page)

Should show:

JOBID	PARTITION	NAME	USER	ST	TIME	NODES	NODELIST (REASON)
1_21	debug	myprogra	fred	PD	0:00	1	(Resources)
1_22	debug	myprogra	fred	PD	0:00	1	(Resources)
1_23	debug	myprogra	fred	PD	0:00	1	(Resources)
1_24	debug	myprogra	fred	PD	0:00	1	(Resources)
1_25	debug	myprogra	fred	PD	0:00	1	(Resources)
1_26	debug	myprogra	fred	PD	0:00	1	(Resources)
1_27	debug	myprogra	fred	PD	0:00	1	(Resources)
1_28	debug	myprogra	fred	PD	0:00	1	(Resources)
1_29	debug	myprogra	fred	PD	0:00	1	(Resources)
1_30	debug	myprogra	fred	PD	0:00	1	(Resources)
1_1	debug	myprogra	fred	R	0:07	1	node00
1_2	debug	myprogra	fred	R	0:07	1	node00
1_3	debug	myprogra	fred	R	0:07	1	node00
1_4	debug	myprogra	fred	R	0:07	1	node00
1_5	debug	myprogra	fred	R	0:07	1	node01
1_6	debug	myprogra	fred	R	0:07	1	node01
1_7	debug	myprogra	fred	R	0:07	1	node01
1_8	debug	myprogra	fred	R	0:07	1	node01
1_9	debug	myprogra	fred	R	0:07	1	node02
1_10	debug	myprogra	fred	R	0:07	1	node02
1_11	debug	myprogra	fred	R	0:07	1	node02
1_12	debug	myprogra	fred	R	0:07	1	node02
1_13	debug	myprogra	fred	R	0:07	1	node03
1_14	debug	myprogra	fred	R	0:07	1	node03
1_15	debug	myprogra	fred	R	0:07	1	node03
1_16	debug	myprogra	fred	R	0:07	1	node03
1_17	debug	myprogra	fred	R	0:07	1	node04
1_18	debug	myprogra	fred	R	0:07	1	node04
1_19	debug	myprogra	fred	R	0:07	1	node04
1_20	debug	myprogra	fred	R	0:07	1	node04

In my case, the jobID was 1. Each subjob in the array was between 1-30.

10. **Evaluate the results.**

You will see that each .err and .out filename corresponds to the jobID and array subjobID. If the jobID I received when I submitted the job was 2, for example, then I would have a .out file called myprogram1_1.out, myprogram1_2.out, myprogram1_3.out, and so forth.

11. The contents of the .out files correspond to the inputX number:

cat myprogram1_1.out

Shows:

input1.dat

Cleanup

1. Reset the Docker Slurm cluster environment back to its original, pristine state by exiting the login Docker container back to the Ubuntu AWS prompt. And **as ubuntu**:

```
cd ~/docker-scale-out  
make clean && make
```

Slurm RESTful API Discussion

REST is an acronym for Representational State Transfer. Like any other architectural style, REST also does have its own 6 guiding constraints which must be satisfied if an interface needs to be referred as RESTful. These principles are as follows:

Guiding Principle of REST (from the rest website)

1. **Client-server** – By separating the user interface concerns from the data storage concerns, we improve the portability of the user interface across multiple platforms and improve scalability by simplifying the server components.
2. **Stateless** – Each request from client to server must contain all of the information necessary to understand the request, and cannot take advantage of any stored context on the server. Session state is therefore kept entirely on the client.
3. **Cacheable** – Cache constraints require that the data within a response to a request be implicitly or explicitly labeled as cacheable or non-cacheable. If a response is cacheable, then a client cache is given the right to reuse that response data for later, equivalent requests.
4. **Uniform interface** – By applying the software engineering principle of generality to the component interface, the overall system architecture is simplified, and the visibility of interactions is improved. In order to obtain a uniform interface, multiple architectural constraints are needed to guide the behavior of components. REST is defined by four interface constraints: identification of resources; manipulation of resources through representations; self-descriptive messages; and hypermedia as the engine of application state.
5. **Layered system** – The layered system style allows an architecture to be composed of hierarchical layers by constraining component behavior such that each component cannot “see” beyond the immediate layer with which they are interacting.
6. **Code on demand (optional)** – REST allows client functionality to be extended by downloading and executing code in the form of applets or scripts. This simplifies clients by reducing the number of features required to be pre-implemented.

The mechanism that provides this functionality in Slurm is `slurmrestd`. It is the REST API interface for Slurm.

Resource:

The key abstraction of information in REST is a resource. Any information that can be named can be a resource: a document or image, a temporal service, a collection of other resources, a non-virtual object (e.g. a person), and so on. REST uses a resource identifier to identify the particular resource involved in an interaction between components.

The state of the resource at any particular timestamp is known as resource representation. A representation consists of data, metadata describing the data and hypermedia links which can help the clients in transition to the next desired state.

The data format of a representation is known as a media type. The media type identifies a specification that defines how a representation is to be processed. A truly RESTful API looks like hypertext. Every addressable unit of information carries an address, either explicitly (e.g., link and id attributes) or implicitly (e.g., derived from the media type definition and representation structure).

Resource Methods

Another important thing associated with REST is resource methods to be used to perform the desired transition. A large number of people wrongly relate resource methods to HTTP GET/PUT/POST/DELETE methods.

Roy Fielding has never mentioned any recommendation around which method to be used in which condition. All he emphasizes is that it should be uniform interface. If you decide HTTP POST will be used for updating a resource – rather than most people recommend HTTP PUT – it's alright and application interface will be RESTful.

Ideally, everything that is needed to change the resource state shall be part of API response for that resource – including methods and in what state they will leave the representation

What is OpenAPI (aka Swagger)

The OpenAPI Specification allows you to describe your entire REST API, including:

- Available endpoints (/users) and operations on each endpoint (GET /users, POST /users)
- Operation parameters Input and output for each operation
- Authentication methods
- Contact information, license, terms of use and other information.
- API specifications can be written in YAML or JSON. The format is easy to learn and readable to both humans and machines

See <https://swagger.io/docs/specification/about/>

OpenAPI is a simple specification for what your program will receive and return. It is MOSTLY free (some less free) tools that will work with the specification. The Pretty documentation can be automatically generated.

Slurm API Methods

The following are the Slurm Methods:

- DELETE /slurm/v0.0.39/job/{job_id}
- GET /slurm/v0.0.39/diag
- GET /slurm/v0.0.39/job/{job_id}
- GET /slurm/v0.0.39/jobs
- GET /slurm/v0.0.39/node/{node_name}
- GET /slurm/v0.0.39/nodes
- GET /slurm/v0.0.39/partition/{partition_name}
- GET /slurm/v0.0.39/partitions
- GET /slurm/v0.0.39/reservation/{reservation_name}
- GET /slurm/v0.0.39/reservations
- GET /slurm/v0.0.39/ping
- GET /slurm/v0.0.39/licenses
- POST /slurm/v0.0.39/job/submit
- POST /slurm/v0.0.39/job/{job_id}
- POST /slurmdbd/v0.0.39/clusters
- POST /slurmdbd/v0.0.39/wckeys
- DELETE /slurmdbd/v0.0.39/account/{account_name}

- DELETE /slurmdb/v0.0.39/association
- DELETE /slurmdb/v0.0.39/cluster/{cluster_name}
- DELETE /slurmdb/v0.0.39/qos/{qos_name}
- DELETE /slurmdb/v0.0.39/user/{user_name}
- DELETE /slurmdb/v0.0.39/wckey/{wckey}
- GET /slurmdb/v0.0.39/diag
- GET /slurmdb/v0.0.39/account/{account_name}
- GET /slurmdb/v0.0.39/accounts
- GET /slurmdb/v0.0.39/association
- GET /slurmdb/v0.0.39/associations
- GET /slurmdb/v0.0.39/cluster/{cluster_name}
- GET /slurmdb/v0.0.39/clusters
- GET /slurmdb/v0.0.39/config
- GET /slurmdb/v0.0.39/job/{job_id}
- GET /slurmdb/v0.0.39/jobs
- GET /slurmdb/v0.0.39/qos
- GET /slurmdb/v0.0.39/qos/{qos_name}
- GET /slurmdb/v0.0.39/tres
- GET /slurmdb/v0.0.39/user/{user_name}
- GET /slurmdb/v0.0.39/users
- GET /slurmdb/v0.0.39/wckey/{wckey}
- GET /slurmdb/v0.0.39/wckeys
- POST /slurmdb/v0.0.39/config
- POST /slurmdb/v0.0.39/accounts
- POST /slurmdb/v0.0.39/tres
- POST /slurmdb/v0.0.39/users

They are documented here:

https://slurm.schedmd.com/rest_api.html#Slurm

The Slurm API Models

The following are the Slurm API Models:

1. dbv0.0.39_account -
2. dbv0.0.39_account_info -
3. dbv0.0.39_account_response -
4. dbv0.0.39_association -
5. dbv0.0.39_association_short_info -
6. dbv0.0.39_associations_info -
7. dbv0.0.39_cluster_info -
8. dbv0.0.39_config_info -
9. dbv0.0.39_config_response -
10. dbv0.0.39_coordinator_info -
11. dbv0.0.39_diag -
12. dbv0.0.39_error -

13. dbv0.0.39_job -
14. dbv0.0.39_job_exit_code -
15. dbv0.0.39_job_info -
16. dbv0.0.39_job_step -
17. dbv0.0.39_qos -
18. dbv0.0.39_qos_info -
19. dbv0.0.39_response_account_delete -
20. dbv0.0.39_response_association_delete -
21. dbv0.0.39_response_cluster_add -
22. dbv0.0.39_response_cluster_delete -
23. dbv0.0.39_response_qos_delete -
24. dbv0.0.39_response_tres -
25. dbv0.0.39_response_user_delete -
26. dbv0.0.39_response_user_update -
27. dbv0.0.39_response_wckey_add -
28. dbv0.0.39_response_wckey_delete -
29. dbv0.0.39_tres_info -
30. dbv0.0.39_user -
31. dbv0.0.39_user_info -
32. dbv0.0.39_wckey -
33. dbv0.0.39_wckey_info -
34. dbv0_0_38_association_default -
35. dbv0_0_38_association_max -
36. dbv0_0_38_association_max_jobs -
37. dbv0_0_38_association_max_jobs_per -
38. dbv0_0_38_association_max_per -
39. dbv0_0_38_association_max_per_account -
40. dbv0_0_38_association_max_tres -
41. dbv0_0_38_association_max_tres_minutes -
42. dbv0_0_38_association_max_tres_minutes_per -
43. dbv0_0_38_association_max_tres_per -
44. dbv0_0_38_association_min -
45. dbv0_0_38_association_usage -
46. dbv0_0_38_cluster_info_associations -
47. dbv0_0_38_cluster_info_controller -
48. dbv0_0_38_diag_RPCs -
49. dbv0_0_38_diag_rollups -
50. dbv0_0_38_diag_time -
51. dbv0_0_38_diag_time_1 -
52. dbv0_0_38_diag_users -
53. dbv0_0_38_job_array -
54. dbv0_0_38_job_array_limits -
55. dbv0_0_38_job_array_limits_max -
56. dbv0_0_38_job_array_limits_max_running -
57. dbv0_0_38_job_comment -
58. dbv0_0_38_job_exit_code_signal -

59. dbv0_0_38_job_het -
60. dbv0_0_38_job_mcs -
61. dbv0_0_38_job_required -
62. dbv0_0_38_job_reservation -
63. dbv0_0_38_job_state -
64. dbv0_0_38_job_step_CPU -
65. dbv0_0_38_job_step_CPU_requested_frequency -
66. dbv0_0_38_job_step_nodes -
67. dbv0_0_38_job_step_statistics -
68. dbv0_0_38_job_step_statistics_CPU -
69. dbv0_0_38_job_step_statistics_energy -
70. dbv0_0_38_job_step_step -
71. dbv0_0_38_job_step_step_het -
72. dbv0_0_38_job_step_step_task -
73. dbv0_0_38_job_step_step_tres -
74. dbv0_0_38_job_step_step_tres_requested -
75. dbv0_0_38_job_step_tasks -
76. dbv0_0_38_job_step_time -
77. dbv0_0_38_job_time -
78. dbv0_0_38_job_time_system -
79. dbv0_0_38_job_time_total -
80. dbv0_0_38_job_time_user -
81. dbv0_0_38_job_tres -
82. dbv0_0_38_job_wckey -
83. dbv0_0_38_qos_limits -
84. dbv0_0_38_qos_limits_max -
85. dbv0_0_38_qos_limits_max_accruing -
86. dbv0_0_38_qos_limits_max_accruing_per -
87. dbv0_0_38_qos_limits_max_jobs -
88. dbv0_0_38_qos_limits_max_jobs_per -
89. dbv0_0_38_qos_limits_max_tres -
90. dbv0_0_38_qos_limits_max_tres_minutes -
91. dbv0_0_38_qos_limits_max_tres_minutes_per -
92. dbv0_0_38_qos_limits_max_tres_per -
93. dbv0_0_38_qos_limits_max_wall_clock -
94. dbv0_0_38_qos_limits_max_wall_clock_per -
95. dbv0_0_38_qos_limits_min -
96. dbv0_0_38_qos_limits_min_tres -
97. dbv0_0_38_qos_limits_min_tres_per -
98. dbv0_0_38_qos_preempt -
99. dbv0_0_38_user_associations -
100. dbv0_0_38_user_default -
101. v0.0.39_diag -
102. v0.0.39_error -
103. v0.0.39_job_properties -
104. v0.0.39_job_resources -

```
105.v0.0.39_job_response_properties -  
106.v0.0.39_job_submission -  
107.v0.0.39_job_submission_response -  
108.v0.0.39_jobs_response -  
109.v0.0.39_node -  
110.v0.0.39_node_allocation -  
111.v0.0.39_nodes_response -  
112.v0.0.39_partition -  
113.v0.0.39_partitions_response -  
114.v0.0.39_ping -  
115.v0.0.39_pings -  
116.v0.0.39_reservation -  
117.v0.0.39_reservations_response -  
118.v0.0.39_signal -  
119.v0_0_38_diag_statistics -  
120.v0_0_38_reservation_purge_completed -
```

They are documented here:

https://slurm.schedmd.com/rest_api.html#_Models

slurmrestd

slurmrestd - Interface to Slurm via REST API. It can be used in two modes:

- Inetd (Daemon) Mode: slurmrestd will read and write to **STDIN** and **STDOUT**. It can detect if it is connected to a socket or a locally TTY (interactive mode).
- Listen Mode: slurmrestd will open a listening socket on each requested host:port pair or UNIX socket.

slurmrestd can be launched with the following options:

[host:port] Hostname and port to listen against. Host may be (IPv4/IPv6) IP or a resolvable hostname.

Hostnames are only looked up at startup and do not change for the life of the process.

unix:/path/to/socket ... Listen on local UNIX socket. Must have permission to create socket in filesystem.

-f <file> Read Slurm configuration from the specified file. See NOTES below.

-g <group id> ... Change group id (and drop supplemental groups) before processing client request. This should be a unique group with no write access or special permissions. Do not set this user to SlurmUser or root.

-h Help - print a brief summary of command options.

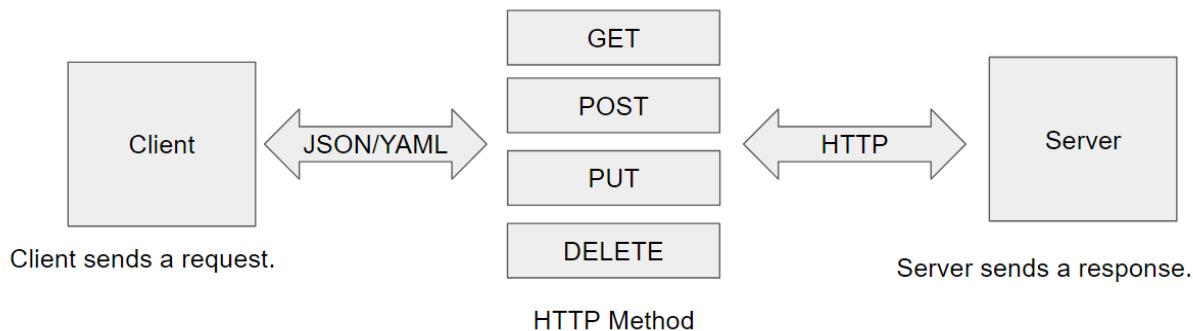
(table of options continues on next page)

- t <THREAD COUNT>** Specify number of threads to use to process client connections. Ignored in Inetd mode.
Default: 20
- u <user id>** Change user id before processing client request. This should be a unique group with no write access or special permissions. Do not set this user to SlurmUser or root.
- v** Verbose operation. Multiple -v's increase verbosity. Higher verbosity levels will have significant performance impact.
- V** Print version information and exit.

The following environment variables can be used to override settings compiled into slurmctld.

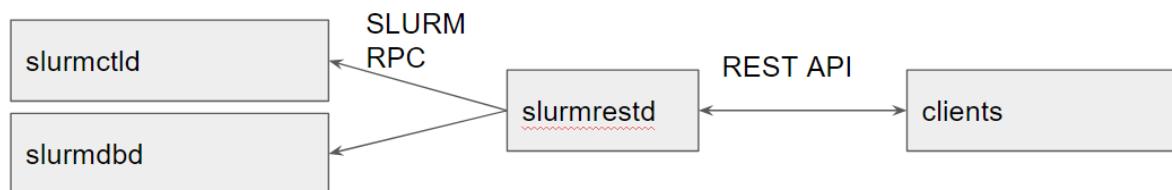
- **SLURM_CONF** - The location of the Slurm configuration file.
- **SLURMRESTD_DEBUG** - Set debug level explicitly. Valid values are 1-10. See -v
- **SLURMRESTD_LISTEN** - Comma delimited list of host:port pairs or unix sockets to listen on.

The following diagram displays the flow of communication:

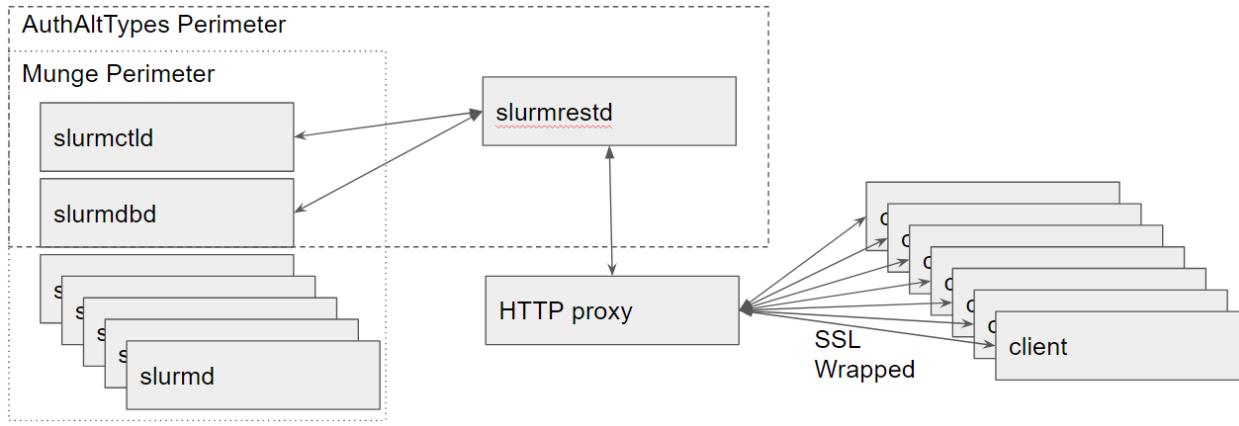


slurmrestd is the tool that runs inside of the Slurm perimeter that will translate JSON/YAML requests into Slurm RPC requests.

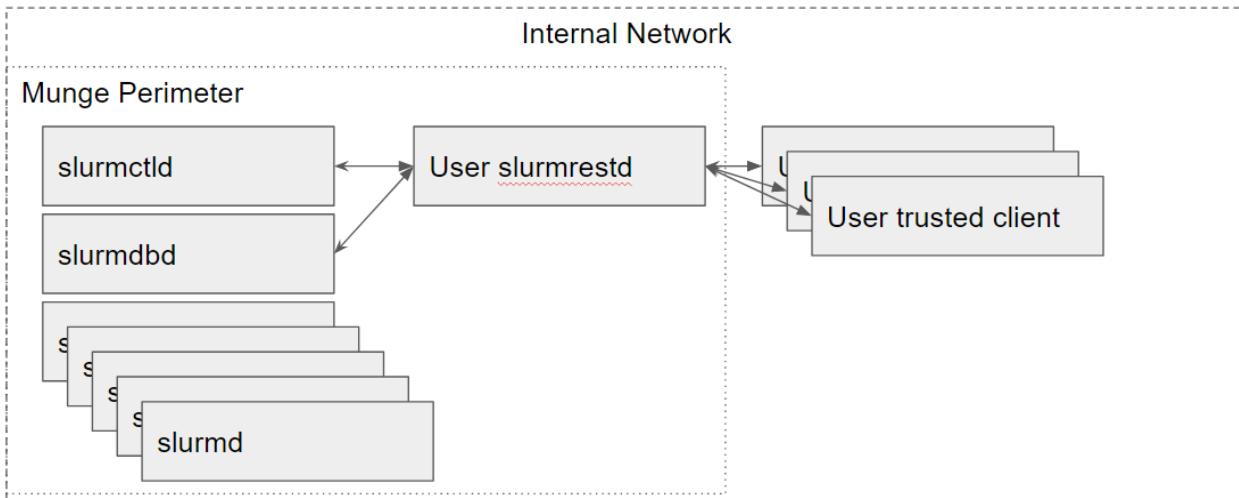
The way the Slurm REST API works is this:



The external REST method is as follows:



The `slurmrestd` API architecture in User Listen mode looks like this:



Slurmrestd Design

1. `slurmrestd` acts as a framework for requests
 - a. Modular addition of new REST commands via registering URL paths
2. Built-in data type to translate JSON and YAML
3. Built-in support for generating OpenAPI specification
4. Conformant HTTP server to handle interfacing with clients and proxies

Examples

Inet Mode Example (unprivileged user):

```
> echo -e "GET /slurm/v1/diag HTTP/1.1\r\nAccept: */*\r\n" | slurmrestd
slurmrestd: operations_router: /slurm/v1/diag for pipe:[722494]
HTTP/1.1 200 OK
Content-Length: 973
{
  "parts_packedg": 1,
  "req_timeg": 1567712456,
... JSON continues ...
```

Listen Mode Example (listen on 3 ports):

```
> slurmrestd -k test -vvv localhost:9997 [::1]:7272 10.10.0.1:38484
slurmrestd: debug:  Reading slurm.conf file: /etc/slurm.conf
slurmrestd: debug:  Interactive mode activated (TTY detected on STDIN)
slurmrestd: debug:  listen: localhost:9997 fd: 3
slurmrestd: debug:  _socket_thread_listen: thread for fd: 3
slurmrestd: debug:  listen: ip6-localhost:7272 fd: 4
slurmrestd: debug:  _socket_thread_listen: thread for fd: 4
slurmrestd: debug:  listen: spheron:38484 fd: 5
slurmrestd: debug:  server listen mode activated
slurmrestd: debug:  _socket_thread_listen: thread for fd: 5
```

Inet Mode Example (unprivileged user with AltAuth):.

```
> echo -e "GET http://localhost/slurm/v1/diag HTTP/1.1\r\nAccept: */*\r\n" | slurmrestd -f
/etc/slurm.token.conf
slurmrestd: operations_router: /slurm/v1/diag for pipe:[1052487]
HTTP/1.1 200 OK
Content-Length: 973
{
    "parts_packedg": 1,
    "req_timeg": 1568051342,
    "req_time_startg": 1568050812,
    "server_thread_count": 3,
... JSON continues ...
```

YAML Mode Example:

```
> echo -e "GET http://localhost/slurm/v1/diag HTTP/1.1\r\nAccept: text/yaml\r\n" |
slurmrestd
slurmrestd: error: _on_url: URL Schema currently not supported for pipe:[782247]
slurmrestd: error: _on_url: URL host currently not supported for pipe:[782247]
slurmrestd: operations_router: /slurm/v1/diag for pipe:[782247]
HTTP/1.1 200 OK
Content-Length: 1210
%YAML 1.1
---
!!str parts_packedg: !!int 1
... YAML Continues ....
```

Security

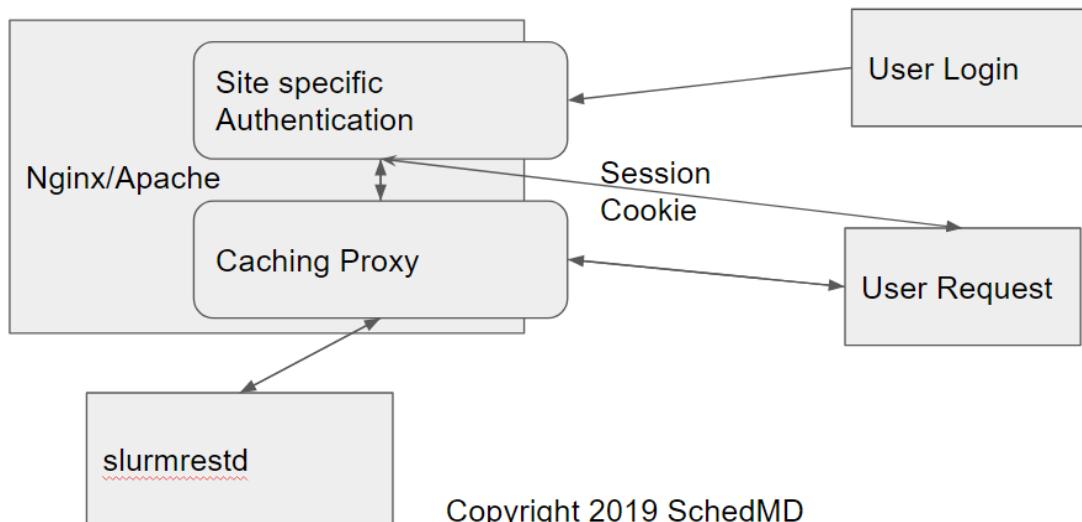
- Authentication - All remote clients must be authenticated via HTTP headers:
 - X-SLURM-API-KEY or X-SLURM-USER-TOKEN
 - X-SLURM-USER-NAME
- Authorization / Mutation
 - Can be offloaded to authenticating proxy.
 - Requests are parsed by cli_filter.
- Denial
 - JSON/YAML requests will not even be parsed without authentication and clients will be rejected with 403 errors.

- User per POSIX thread
 - Utilizes new AuthAltTypes framework.
 - Allows `slurmrestd` to exist outside of Munge.
- Authentication Types
 - Local
 - Only allowed for piping directly into `slurmrestd`
 - Authenticating Proxy
 - Proxy is given a key that will allow proxy to authenticate as any user
 - User Token
 - Each user will have a token to authenticate.

Authentication with REST API

1. Authenticating proxy to verify users and user requests prior to being sent to REST API.
2. Authentication can be:
 - a. Site specific
 - i. Too many authentication schemes to support them all.
 - ii. Filter/modify user requests
 - b. Scalable
 - i. External/Slow authentication systems can be:
 1. Cached or Offloaded
 2. Parallelized

Authenticating HTTP Proxy Example:



Simple Authenticating HTTP Proxy Demo

```

> cd src/contribs/auth_proxy_example/simple
> make #starts docker auth-proxy:latest
> slurmrestd localhost:9999 & #start background daemon
> curl 'http://localhost:8080/slurm/v1/ping'

```

```

<html><head><title>401 Authorization Required</title></head>
<body><center><h1>401 Authorization Required</h1></center>
<hr><center>nginx</center></body></html>

> curl --user test:test -c /tmp/cookiejar 'http://localhost:8080/auth/'
'http://localhost:8080/slurm/v1/ping'
<p>Hello test.</p><p>You entered test as your password.</p>
{ "errors": [], "ping": { "spheron": { }, "ping": "UP", "status": 0, "mode": "primary" }
}

```

IPv4 and IPv6 Support

slurmrestd supports IPv6 and IPv4 clients (or proxies).

slurmrestd can act as REST gateway into IPv4 cluster to external IPv6 clients.

slurmrestd must be able to talk to slurmcld/slurmdbd over IPv4.

IPv6 Mode Example:

```

> slurmrestd -vv [::1]:9999 & slurmrestd: slurmrestd: debug: listen: ip6-localhost:9999
fd: 3

> curl -s 'http://[::1]:9999/slurm/v1/ping'

slurmrestd: debug: parse_http: Accepted HTTP connection from ::%2698417920:49016
slurmrestd: debug: connection ::%2698417920:49016 url path: /slurm/v1/ping query: (null)
slurmrestd: operations_router: /slurm/v1/ping for ::%2698417920:49016
{   "errors": [   ],   "ping": {       "spheron": {       },       "ping": "UP",       "status": 0,
"mode": "primary"   }   }
slurmrestd: debug: parse_http: Closed HTTP connection from ::%2698417920:49016

```

Example Queries

Job submission request example (sbatch):

```

$ cat demo
POST /slurm/v1/job/submit HTTP/1.1
Accept: text/yaml
Content-Type: application/json
Content-Length: 348

{
  "job": {
    "account": "test",
    "ntasks": 20,
    "name": "test18.1",
    "nodes": [2, 4],
    "current_working_directory": "/tmp/",
    "user_id": 1000,
    "group_id": 1000,
    "environment": {
      "PATH": "/bin:/usr/bin:/usr/local/bin/",
      "LD_LIBRARY_PATH": "/lib:/lib64:/usr/local/lib"
    },
    "script": "#!/bin/bash\necho it works"
  }
}

```

Job submission example (sbatch):

```
$ slurmrestd 2>/dev/null < demo
HTTP/1.1 200 OK
Content-Length: 131
%YAML 1.1
---
!!str errors: !!seq []
!!str job_id: !!int 115
!!str step_id: !!str BATCH
!!str job_submit_user_msg: !!null null
...
$ squeue -j 115
      JOBID   PARTITION     USER   ST      TIME  NODES NODELIST(REASON)
          115       debug     nate   PD      0:00      4  (Priority))
```

Job submission request example (sbatch HetJob):

```
> cat demo
POST /slurm/v1/job/submit HTTP/1.1
Accept: text/yaml
Content-Type: application/json
Content-Length: 654
{"job": [{"account": "test", "ntasks": 20, "name": "test18.1", "nodes": [2, 4], "current_working_directory": "/tmp/", "user_id": 1000, "group_id": 1000, "environment": {"PATH": "/bin:/usr/bin:/usr/local/bin/", "LD_LIBRARY_PATH": "/lib:/lib64:/usr/local/lib" }}, {"account": "test", "ntasks": 2, "name": "test18.3", "nodes": [2, 4], "current_working_directory": "/tmp/", "user_id": 1000, "group_id": 1000, "environment": {"PATH": "/bin:/usr/bin:/usr/local/bin/", "LD_LIBRARY_PATH": "/lib:/lib64:/usr/local/lib" }}, "script": "#!/bin/bash\nsrun echo it works" }]
```

Job submission example (sbatch HetJob):

```
$ slurmrestd < demo
HTTP/1.1 200 OK
Content-Length: 131

%YAML 1.1
---
!!str errors: !!seq []
!!str job_id: !!int 118
!!str step_id: !!str BATCH
!!str job_submit_user_msg: !!null null
...
```

Job submission example (sbatch HetJob):

```
$ squeue -j 118
      JOBID   PARTITION     USER   ST      TIME  NODES NODELIST(REASON)
        118+0       debug     nate   PD      0:00      4  (None)
        118+1       debug     nate   PD      0:00      4  (None) ...
```

Job submission example (sbatch array):

```
$ cat demo
POST /slurm/v1/job/submit HTTP/1.1
Accept: text/yaml
Content-Type: application/json
Content-Length: 374
{"job": {"account": "test", "array": "1-1000", "ntasks": 20, "name": "test18.1", "nodes": [2, 4], "current_working_directory": "/tmp/", "user_id": 1000, "group_id": 1000, "environment": {"PATH": "/bin:/usr/bin/:/usr/local/bin/", "LD_LIBRARY_PATH": "/lib:/lib64:/usr/local/lib"}, "script": "#!/bin/bash\nnsrun echo it works"}...}
```

Job submission example (sbatch array)

```
$ slurmrestd < demo
HTTP/1.1 200 OK
Content-Length: 132

%YAML 1.1
---
!!str errors: !!seq []
!!str job_id: !!int 1202
!!str step_id: !!str BATCH
!!str job_submit_user_msg: !!null null
...
```

Job submission example (sbatch array)

	JOBID	PARTITION	USER	ST	TIME	NODES	NODELIST (REASON)
	1202_[528-1000]	debug	nate	PD	0:00	4	(Resources)
	1202_527	debug	nate	R	0:00	4	host[5-8]
	1202_526	debug	nate	R	0:00	4	host[1-4]

Cancelling job example (sbatch array):

```
> cat demo
DELETE /slurm/v1/job/2202 HTTP/1.1
Accept: text/yaml

$ slurmrestd < demo
HTTP/1.1 200 OK
Content-Length: 41
%YAML 1.1
---
!!str errors: !!seq []
...

$ scontrol show job 2202 | grep JobState
JobState=CANCELLED Reason=None Dependency=(null)
```

View job example (sbatch array):

```
> echo -e 'GET /slurm/v1/job/2203 HTTP/1.1\r\nACCEPT: text/json\r\n\r\n' | slurmrestd
{"account": "test", "accrue_time": 1568158776, "admin_comment": "", "array_job_id": 0, "array_task_
```

```

id": {}, "array_max_tasks": 0, "array_task_str": "", "assoc_id": 4, "batch_features": "", "batch_flag": true, "batch_host": "host1", "bitflags": ["JOB_WAS_RUNNING"], "boards_per_node": 0, "burst_buffer": "", "burst_buffer_state": "", "cluster": "linux", "cluster_features": "", "command": "", "comment": "", "contiguous": false, "core_spec": {}, "thread_spec": {}, "cores_per_socket": {}, "billable_tres": 12, "cpus_per_task": {}, "cpu_freq_min": {}, "cpu_freq_max": {}, "cpu_freq_gov": {}, "cpus_per_tries": "", "deadline": 0, "delay_boot": 0, "dependency": "", "derived_ec": 0, "eligible_time": 1568158776, "end_time": 1599694776, "exc_nodes": "", "execution_node_by_index": [], "exit_code": 0, "features": "", "fed_origin_str": "", "fed_siblings_active": 0, "fed_siblings_active_str": "", "fed_siblings_viable": 0, "fed_siblings_viable_str": "", "gres_detail": [], "group_id": 1000, "job_id": 2203, "job_resources": {}, "job_state": "RUNNING", "last_sched_eval": 1568158776, "licenses": "", "max_cpus": 0, "max_nodes": 0, "mcs_label": "", "mem_per_tres": "", "name": "wrap", "network": "", "nodes": "host1", "nice": {}, "node_index": [0, 0], "ntasks_per_core": {}, "ntasks_per_node": 0, "ntasks_per_socket": {}, "ntasks_per_board": 0, "num_cpus": 12, "num_nodes": 1, "num_tasks": 1, "pack_job_id": 0, "pack_job_id_set": "", "pack_job_offset": 0, "partition": "debug", "pn_min_memory": {}, "mem_per_cpu": {}, "pn_min_cpus": 1, "pn_min_tmp_disk": 0, "power_flags": [], "preempt_time": 0, "pre_sus_time": 0, "priority": 4294901636, "profile": {}, "qos": "normal", "reboot": false, "req_nodes": "", "requested_node_by_index": [], "requeue": true, "resize_time": 0, "restart_cnt": 0, "resv_name": "", "sched_nodes": "", "select_jobinfo": "", "shared": "none", "show_flags": ["SHOW_ALL", "SHOW_LOCAL"], "sockets_per_board": 0, "sockets_per_node": {}, "start_time": 1568158776, "start_protocol_ver": 8960, "state_desc": "", "state_reason": "None", "std_err": "", "std_in": "/dev/null", "std_out": "", "submit_time": 1568158776, "suspend_time": 0, "system_comment": "", "time_limit": 525600, "time_min": 0, "threads_per_core": {}, "tres_bind": "", "tres_freq": "", "tres_per_job": "", "tres_per_node": "", "tres_per_socket": "", "tres_per_task": "", "tres_req_str": "cpu=1,node=1,billing=1", "tres_alloc_str": "cpu=12,node=1,billing=12", "user_id": 1000, "user_name": "", "wait4switch": 0, "wckey": "", "work_dir": "/home/nate/slurm/restapi/demo"}

```

Diagnostics query example (sbatch array):

```

> echo -e 'GET /slurm/v1/diag HTTP/1.1\r\nACCEPT: text/json\r\n\r\n' | slurmrestd
{
  "parts_packedg": 1,
  "req_timeeg": 1568158683,
  "req_time_startg": 1568153903,
  "server_thread_count": 3,
  "agent_queue_size": 0,
  "agent_count": 0,
  "dbd_agent_queue_size": 0,
  "gettimeofday_latency": 18,
  "schedule_cycle_max": 12041,
  "schedule_cycle_last": 18,
  "schedule_cycle_sum": 1008716,
  "schedule_cycle_counter": 1138,
  "schedule_cycle_depth": 5109,
  "schedule_queue_len": 0,
  "jobs_submitted": 2130,
  "jobs_started": 2083,
  "jobs_completed": 2082,
  "jobs_canceled": 54,
  "jobs_failed": 0,
  "jobs_pending": 0,
  "jobs_running": 0,
  "job_states_ts": 1568158674,
  "bf_backfilled_jobs": 1,
  "bf_last_backfilled_jobs": 1,
  "bf_backfilled_pack_jobs": 0,
  "bf_cycle_counter": 130,
  "bf_cycle_sum": 37282,
  "bf_cycle_last": 65,
  "bf_cycle_max": 690,
  "bf_last_depth": 1,
  "bf_last_depth_try": 1,
  "bf_depth_sum": 3356,
  "bf_depth_try_sum": 260,
  "bf_queue_len": 0,
  "bf_queue_len_sum": 0,
  "bf_when_last_cycle": 1568158222,
  "bf_active": 0
}

```

Additional Resources and Links

<https://slurm.schedmd.com/download.html>

https://slurm.schedmd.com/rest_api.html

<https://slurm.schedmd.com/rest.html>

<https://slurm.schedmd.com/jwt.html>

slurmrestd Lab Exercises

In this training environment, there is a docker container named "rest." In it, there is a user called "slurmrestd" that runs the /usr/local/sbin/slurmrestd, that listens on port 80.

Exercise 1: OpenAPI Generation

Using curl, you will download the openapi specification (ignoring the fact that there are 3 API specs, we combine them all in our tool), which will download a json file using the openapi generator.

1. For the purposes of this exercise, extend the life of the JWT, by exporting the token with a lifespan longer than the default 3 seconds. **As fred:**

```
export $(scontrol token lifespan=99999)
```

2. Make sure the JWT environment is present:

```
echo $SLURM_JWT
```

Should show (something like):

```
SLURM_JWT=eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJleHAiOiE2NTQyODY2ODcsImh  
dCI6MTY1NDE4NjY4OCwic3VuIjoiZnJlZCJ9.Ai6WvcSfKES8cwjG5z1-
```

3. Run the following 2 commands to download and combine the json:

```
echo -e "GET http://localhost/openapi/v3 \  
HTTP/1.1\r\nAccept:application/json\r\n" | slurmrestd -a \  
rest_auth/local -s v0.0.39,dv0.0.39 | sed '0,/^\r$/d' > openapi.json  
  
openapi-generator-cli generate -i openapi.json -g python --strict-\  
spec=false -o py_api_client --skip-validate-spec
```

4. These command generated the openapi.json file which contains all of the methods for the slurmrestd. The Openapigenerator creates a directory in which the python wrapper package for our Rest API are stored. It is called py_api_json. View it:

```
ls -l
```

```
total 176  
-rw-r--r-- 1 fred users 175384 Nov 20 16:20 openapi.json
```

NOTE: In the py_api_client/docs directory, you will see an SlurmApi.md file, which was generated and contains all the documentation for the python library

5. Next, install the newly-generated python code:

```
pushd py_api_client; python3 setup.py install --user; popd;
```

Now that you have the python3 openapi client for slurmrestd, you can use it to make slurmrest calls.

6. It's possible to use this method to generate the openapi Slurm documentation as well. **Again as fred,** enter the following (**THIS COMMAND IS ON AN ENTIRE LINE BY ITSELF-IT WRAPS IN THIS DOCUMENT**):

```
openapi-generator-cli generate -i openapi.json -g html \
--strict-spec=true -o html --skip-validate-spec
```

7. Change directory into the html folder:

```
cd html
```

8. Use lynx (non-gui browser, which is installed by default in this cluster) to see the generated documentation page:

```
lynx index.html
```

You should see:

The screenshot shows a terminal window displaying the Slurm Rest API documentation. The title bar says "Slurm Rest API (p1 of 423)". The content includes API details, access information, and methods. It ends with a message to press space for the next page.

```
Slurm Rest API (p1 of 423)

API to access and control Slurm.

More information: https://www.schedmd.com/

Contact Info: sales@schedmd.com

Version: 0.0.39

BasePath:

Apache 2.0

https://www.apache.org/licenses/LICENSE-2.0.html

Access

1. HTTP Basic Authentication
2. APIKey KeyParamName:X-SLURM-USER-TOKEN KeyInQuery:false KeyInHeader:true
3. APIKey KeyParamName:X-SLURM-USER-NAME KeyInQuery:false KeyInHeader:true

Methods

-- press space for next page --
Arrow keys: Up and Down to move. Right to follow a link; Left to go back.
H)elp O)ptions P)rint G)o M)ain screen Q)uit /=search [delete]=history list
```

Feel free to link around the docs

You can generate the docs in other formats as well: [html2](#), [asciidoc](#), [cwiki](#), [openampi-yaml](#) (See <https://openapi-generator.tech/docs/generators/>)

We also provide the generated OpenAPI documentation online. It can be found here:

https://slurm.schedmd.com/rest_api.html

Exercise 2: Direct Slurm REST query using JWT:

Using this method, it would be the same as `sdiag` from the slurm command set.

1. For the purposes of this exercise, extend the life of the JWT, by exporting the token with a lifespan longer than the default 3 seconds. **As fred**:

```
export $(scontrol token lifespan=99999)
```
2. Run the following command **as fred** to get Slurm configuration data in JSON format (**THIS COMMAND IS ON AN ENTIRE LINE BY ITSELF-IT WRAPS IN THIS DOCUMENT**):

```
curl -s -H X-SLURM-USER-NAME:$ (whoami) -H \
X-SLURM-USER-TOKEN:$SLURM_JWT 'http://rest/slurm/v0.0.39/diag' | jq
```

You should see the following:

```
{
  "meta": {
    "plugin": {
      "type": "openapi/v0.0.39",
      "name": "Slurm OpenAPI v0.0.39"
    },
    "Slurm": {
      "version": {
        "major": 22,
        "micro": 0,
        "minor": 5
      },
      "release": "22.05.0"
    }
  },
  "errors": [],
  "statistics": {
    "rpcs_by_message_type": [
      {
        "message_type": "MESSAGE_NODE_REGISTRATION_STATUS",
        "type_id": 1002,
        "count": 30,
        "average_time": 1115,
        "total_time": 33474
      },
      :
      (output truncated)
      :
      "bf_backfilled_jobs": 0,
      "bf_last_backfilled_jobs": 0,
      "bf_backfilled_het_jobs": 0,
      "bf_cycle_counter": 0,
      "bf_cycle_mean": 0,
      "bf_depth_mean": 0,
      "bf_depth_mean_try": 0,
      "bf_cycle_last": 0,
      "bf_cycle_max": 0,
      "bf_queue_len": 0,
      "bf_queue_len_mean": 0,
      "bf_table_size": 0,
      "bf_table_size_mean": 0,
      "bf_when_last_cycle": 0,
      "bf_active": false
    }
  }
}
```

3. To verify the status of the headnode, you can run the following equivalent of “scontrol ping” (**THIS COMMAND IS ON AN ENTIRE LINE BY ITSELF-IT WRAPS IN THIS DOCUMENT**):

```
curl -s -H X-SLURM-USER-NAME:$ (whoami) -H \
X-SLURM-USER-TOKEN:$SLURM_JWT 'http://rest/slurm/v0.0.39/ping' | jq
```

You should see:

```
{
  "meta": {
    "plugin": {
      "type": "openapi/v0.0.39",
      "name": "Slurm OpenAPI v0.0.39"
    },
    "Slurm": {
      "version": {
        "major": 22,
        "micro": 0,
        "minor": 5
      },
      "release": "22.05.0"
    }
  },
  "errors": [],
  "pings": [
    {
      "hostname": "mgmtnode",
      "ping": "UP",
      "status": 0,
      "mode": "primary"
    },
    {
      "hostname": "mgmtnode2",
      "ping": "UP",
      "status": 0,
      "mode": "backup"
    }
  ]
}
```

4. To get the diag information in YAML format (**THIS COMMAND IS ON AN ENTIRE LINE BY ITSELF-IT WRAPS IN THIS DOCUMENT**):

```
curl -s -H X-SLURM-USER-NAME:$ (whoami) -H \
X-SLURM-USER-TOKEN:$SLURM_JWT -H \
"Accept: application/x-yaml" 'http://rest/slurm/v0.0.39/diag' ;echo
```

(output on next page)

You should see:

```
%YAML 1.1
--- !!map
!!str meta: !!map
  !!str plugin: !!map
    !!str type: !!str openapi/v0.0.39
    !!str name: !!str Slurm OpenAPI v0.0.39
  !!str Slurm: !!map
    !!str version: !!map
      !!str major: !!int 22
      !!str micro: !!int 0
      !!str minor: !!int 5
    !!str release: !!str 22.05.0
  !!str errors: !!seq []
  !!str statistics: !!map
    !!str rpcs_by_message_type: !!seq
      - !!map
        !!str message_type: !!str MESSAGE_NODE_REGISTRATION_STATUS
        !!str type_id: !!int 1002
        !!str count: !!int 40
        !!str average_time: !!int 989
        !!str total_time: !!int 39567
      - !!map
        !!str message_type: !!str REQUEST_CONTROL_STATUS
        !!str type_id: !!int 2053
        !!str total_time: !!int 1415
      :
  (output truncated)
  :
  !!str jobs_running: !!int 0
  !!str job_states_ts: !!int 1654197750
  !!str bf_backfilled_jobs: !!int 0
  !!str bf_last_backfilled_jobs: !!int 0
  !!str bf_backfilled_het_jobs: !!int 0
  !!str bf_cycle_counter: !!int 0
  !!str bf_cycle_mean: !!int 0
  !!str bf_depth_mean: !!int 0
  !!str bf_depth_mean_try: !!int 0
  !!str bf_cycle_last: !!int 0
  !!str bf_cycle_max: !!int 0
  !!str bf_queue_len: !!int 0
  !!str bf_queue_len_mean: !!int 0
  !!str bf_table_size: !!int 0
  !!str bf_table_size_mean: !!int 0
  !!str bf_when_last_cycle: !!int 0
  !!str bf_active: !!bool false
  ...
```

5. To get the ping information in YAML format (**THIS COMMAND IS ON AN ENTIRE LINE BY ITSELF-IT WRAPS IN THIS DOCUMENT**):

```
curl -s -H X-SLURM-USER-NAME:$ (whoami) -H \
X-SLURM-USER-TOKEN:$SLURM_JWT -H "Accept: text/yaml" \
'http://rest/slurm/v0.0.39/ping';echo
```

You should see:

```
%YAML 1.1
--- !!map
!!str meta: !!map
  !!str plugin: !!map
    !!str type: !!str openapi/v0.0.39
    !!str name: !!str Slurm OpenAPI v0.0.39
  !!str Slurm: !!map
    !!str version: !!map
      !!str major: !!int 22
      !!str micro: !!int 0
      !!str minor: !!int 5
    !!str release: !!str 22.05.0
  !!str errors: !!seq []
  !!str pings: !!seq
  - !!map
    !!str hostname: !!str mgmtnode
    !!str ping: !!str UP
    !!str status: !!int 0
    !!str mode: !!str primary
  - !!map
    !!str hostname: !!str mgmtnode2
    !!str ping: !!str UP
    !!str status: !!int 0
    !!str mode: !!str backup
  ...
```

Exercise 3: Direct Slurm REST job submission

in this exercise you will use a direct Slurm REST call to submit and confirm job submission

1. As **fred**, update the JWT token:

```
export $(scontrol token lifespan=99999)
```

2. As fred, create job.json (found in the /lab_scripts/ folder):

```
{  
    "job": {  
        "tasks": 8,  
        "name": "test",  
        "nodes": 2,  
        "current_working_directory": "/tmp/",  
        "environment": {  
            "PATH": "/bin:/usr/bin/:/usr/local/bin/",  
            "LD_LIBRARY_PATH": "/lib/:/lib64/:/usr/local/lib"  
        }  
    },  
    "script": "#!/bin/bash\nsleep 100"  
}
```

3. Submit the job (THIS COMMAND IS ON AN ENTIRE LINE BY ITSELF-IT WRAPS IN THIS DOCUMENT):

```
curl -s -H X-SLURM-USER-NAME:$(whoami) -H \  
X-SLURM-USER-TOKEN:$SLURM_JWT -X POST \  
'http://rest/slurm/v0.0.39/job/submit' -H \  
"Content-Type:application/json" -d @job.json | jq
```

You should see:

```
{  
    "meta": {  
        "plugin": {  
            "type": "openapi/v0.0.39",  
            "name": "Slurm OpenAPI v0.0.39",  
            "data_parser": "v0.0.39"  
        },  
        "client": {  
            "source": "[login]:55696"  
        },  
        "Slurm": {  
            "version": {  
                "major": 23,  
                "micro": 0,  
                "minor": 2  
            },  
            "release": "23.02.0"  
        }  
    },  
    "errors": [],  
    "warnings": [],  
    "result": {  
        "job_id": 2,  
        "step_id": "batch",  
        "error_code": 0,  
        "error": "No error",  
        "job_submit_user_msg": ""  
    },  
    "job_id": 2,  
    "step_id": "batch",  
    "job_submit_user_msg": ""  
}
```

The job should be running. In my case, job 2.

4. To see the running job, issue the following (**THIS COMMAND IS ON AN ENTIRE LINE BY ITSELF-IT WRAPS IN THIS DOCUMENT**):

```
curl -s -H X-SLURM-USER-NAME:$ (whoami) -H \
X-SLURM-USER-TOKEN:$SLURM_JWT 'http://rest/slurm/v0.0.39/jobs' | jq
```

You should see:

```
{
  "meta": {
    "plugin": {
      "type": "openapi/v0.0.39",
      "name": "Slurm OpenAPI v0.0.39"
    },
    "Slurm": {
      "version": {
        "major": 22,
        "micro": 0,
        "minor": 5
      },
      "release": "22.05.0"
    }
  },
  "errors": [],
  "jobs": [
    {
      "account": "bedrock",
      "accrue_time": 1654197857,
      "admin_comment": "",
      "array_job_id": 0,
      "array_task_id": null,
      "array_max_tasks": 0,
      "array_task_string": "",
      "association_id": 14,
      "batch_features": "",
      "cores_per_socket": null,
      "billable_tres": 8,
      "cpus_per_task": null,
      "cpu_frequency_maximum": null,
      "cpu_frequency_minimum": null,
      "delay_boot": 0,
      "shared": null,
      "show_flags": [
        "SHOW_ALL",
        "SHOW_DETAIL",
        "SHOW_LOCAL"
      ],
      "sockets_per_board": 0,
      "sockets_per_node": null,
      "start_time": 1654197858,
      "state_description": "",
      "state_reason": "None",
      "standard_error": "",
      "standard_input": "",
      "standard_output": "",
      "submit_time": 1654197857,
      "suspend_time": 0,
      "system_comment": "",
      "time_limit": null,
      "time_minimum": 0,
      "threads_per_core": null,
      "tres_bind": "",
      "tres_freq": "",
      "tres_per_job": "",
      "tres_per_node": "",
      "tres_per_socket": "",
      "tres_per_task": "",
      "tres_req_str": "cpu=8,mem=31958M,node=2,billing=8",
      "tres_alloc_str": "cpu=8,mem=31958M,node=2,billing=8",
      "user_id": 1010,
      "user_name": "fred",
      "wckey": "",
      "current_working_directory": "/tmp/"
    }
  ]
}
```

5. It is also possible to query a single job (**THIS COMMAND IS ON AN ENTIRE LINE BY ITSELF-IT WRAPS IN THIS DOCUMENT**):

```
curl -s -H X-SLURM-USER-NAME:$ (whoami) -H \
X-SLURM-USER-TOKEN:$SLURM_JWT 'http://rest/slurm/v0.0.39/job/1' | jq
```

6. To get information about the configured nodes and their state (**THIS COMMAND IS ON AN ENTIRE LINE BY ITSELF-IT WRAPS IN THIS DOCUMENT**):

```
curl -s -H X-SLURM-USER-NAME:$ (whoami) -H \
X-SLURM-USER-TOKEN:$SLURM_JWT 'http://rest/slurm/v0.0.39/nodes' | jq
```

NOTE: In Slurm v23.02, the reservations on a node have been added to this output

7. Or, to get a specific node (**THIS COMMAND IS ON AN ENTIRE LINE BY ITSELF-IT WRAPS IN THIS DOCUMENT**):

```
curl -s -H X-SLURM-USER-NAME:$ (whoami) -H \
X-SLURM-USER-TOKEN:$SLURM_JWT \
'http://rest/slurm/v0.0.39/node/node00' | jq
```

NOTE: To find the rest of the JSON calls and submit policies, you can go to the rest docker node, and check out the /root/slurm/src/slurmrestd/plugins/openapi/v0.0.39/openapi.json file. Or, since we generated the file locally in a previous step, just look at the openapi.json file in fred's home folder.

Exercise 4: Submitting a Proxy-authenticated job through slurmrestd

In this exercise you will submit a job using curl and having the REST API (slurmrestd) process the request.

1. Authenticate as fred:

```
curl -c ~/.cookiejar 'http://proxy:8080/auth/?user=fred'
```

2. Refer to, and use, the job.json from the last exercise (it's the same):

```
{
  "job": {
    "tasks": 8,
    "name": "test",
    "nodes": 2,
    "current_working_directory": "/tmp/",
    "environment": {
      "PATH":"/bin:/usr/bin/:/usr/local/bin/",
      "LD_LIBRARY_PATH": "/lib/:/lib64/:/usr/local/lib"
    }
  },
  "script": "#!/bin/bash\nsrun sleep 100"
}
```

3. Submit the job (**THIS COMMAND IS ON AN ENTIRE LINE BY ITSELF-IT WRAPS IN THIS DOCUMENT**):

```
curl -b ~/.cookiejar -s -X POST \
'http://proxy:8080/slurm/v0.0.39/job/submit' -H \
"Content-Type: application/json" --data-binary @/lab_scripts/job.json
```

Exercise 5: Cancel Job using curl through slurmrestd and using auth

No sane user should be doing this, but it's for the API programmers.

1. First, authenticate:

```
curl -c ~/.cookiejar 'http://proxy:8080/auth/?user=fred'
```

2. Then submit a job:

```
sbatch -N1 -t100 --wrap="sleep 100"
```

3. Record the JOBID: _____

4. Cancel the job through curl (**THIS COMMAND IS ON AN ENTIRE LINE BY ITSELF-IT WRAPS IN THIS DOCUMENT**):

```
curl -b ~/.cookiejar -X DELETE \
"http://proxy:8080/slurm/v0.0.39/job/<THE JOB ID>"
```

You should see:

```
{
  "meta": {
    "plugin": {
      "type": "openapi\v0.0.39",
      "name": "Slurm OpenAPI v0.0.39"
    },
    "Slurm": {
      "version": {
        "major": 22,
        "micro": 0,
        "minor": 5
      },
      "release": "22.05.0"
    }
  },
  "errors": [
  ]
}
```

5. Using squeue, verify the job has been cancelled.

Cleanup

1. Reset the Docker Slurm cluster environment back to its original, pristine state by exiting the login Docker container back to the Ubuntu AWS prompt. And **as ubuntu**:

```
cd ~/docker-scale-out  
make clean && make
```

Slurm Job Management

There are many tools to manage the workload on the cluster, either running or pending. They include:

- `scancel`
- `scontrol update`
- `scontrol show (Jobs, nodes, partitions, reservations)`
- `sinfo`
- `sview`

`scancel`

Description

`scancel` is used to signal or cancel jobs, job arrays or job steps. An arbitrary number of jobs or job steps may be signaled using job specification filters or a space separated list of specific job and/or job step IDs. If the job ID of a job array is specified with an array ID value then only that job array element will be cancelled. If the job ID of a job array is specified without an array ID value then all job array elements will be cancelled. While a heterogeneous job is in pending state, only the entire job can be cancelled rather than its individual components.

A request to cancel an individual component of a heterogeneous job not in pending state will return an error.

After the job has begun execution, the individual component can be cancelled. A job or job step can only be signaled by the owner of that job or user root. If an attempt is made by an unauthorized user to signal a job or job step, an error message will be printed and the job will not be signaled.

`scancel` Usage

`-A, --account=account`

Restrict the `scancel` operation to jobs under this charge account.

`-b, --batch`

Signal only the batch step (the shell script), but not any other steps nor any children of the shell script.

This is useful when the shell script has to trap the signal and take some application defined action. This is not applicable if `step_id` is specified. NOTE: The shell itself may exit upon receipt of many signals. You may avoid this by explicitly trap signals within the shell script (e.g. "trap <arg> <signals>"). See the shell documentation for details. Also see the `-f, --full` option.

`--ctld`

Send the job signal request to the `slurmctld` daemon rather than directly to the `slurmd` daemons. This increases overhead, but offers better fault tolerance. This is the default behavior on architectures using front end nodes (e.g. Cray ALPS computers) or when the `--clusters` option is used.

`-f, --full`

Signal all steps associated with the job including any batch step (the shell script plus all of its child processes). By default, signals other than SIGKILL are not sent to the batch step. Also see the `-b, --batch` option.

`--help`

Print a help message describing all `scancel` options.

-H, --hurry

Do not stage out any burst buffer data.

-i, --interactive

Interactive mode. Confirm each job_id.step_id before performing the cancel operation.

-M, --clusters=<string>

Clusters to issue commands to. Note that the SlurmDBD must be up for this option to work properly.

-n, --jobname=job_name, --name=job_name

Restrict the `scancel` operation to jobs with this job name.

-p, --partition=partition_name

Restrict the `scancel` operation to jobs in this partition.

-q, --qos=qos

Restrict the `scancel` operation to jobs with this quality of service.

-Q, --quiet

Do not report an error if the specified job is already completed. This option is incompatible with the `--verbose` option.

-R, --reservation=reservation_name

Restrict the `scancel` operation to jobs with this reservation name.

--sibling=cluster_name

Remove an active sibling job from a federated job.

-s, --signal=signal_name

The name or number of the signal to send. If this option is not used the specified job or step will be terminated. Note: If this option is used the signal is sent directly to the slurmd where the job is running bypassing the slurmctld thus the job state will not change even if the signal is delivered to it. Use the `scontrol` command if you want the job state change be known to slurmctld.

-t, --state=job_state_name

Restrict the `scancel` operation to jobs in this state. `job_state_name` may have a value of either "PENDING", "RUNNING" or "SUSPENDED".

-u, --user=user_name

Restrict the `scancel` operation to jobs owned by this user.

--usage

Print a brief help message listing the `scancel` options.

-v, --verbose

Print additional logging. Multiple v's increase logging detail. This option is incompatible with the –quiet option.

-V, --version

Print the version number of the `scancel` command.

-w, --nodelist=host1,host2,...

Cancel any jobs using any of the given hosts. The list may be specified as a comma-separated list of hosts, a range of hosts (host[1-5,7,...] for example), or a filename. The host list will be assumed to be a filename only if it contains a "/" character.

--wckey=wckey

Restrict the `scancel` operation to jobs using this workload characterization key.

ENVIRONMENT VARIABLES

Some `scancel` options may be set via environment variables. These environment variables, along with their corresponding options, are listed below. (Note: the commandline options will always override the variables)

Environment Variable	Related Option
SCANCEL_ACCOUNT	-A, --account=account
SCANCEL_BATCH	-b, --batch
SCANCEL_CTLID	--ctlid
SCANCEL_FULL	-f, --full
SCANCEL_HURRY	-H, --hurry
SCANCEL_INTERACTIVE	-i, --interactive
SCANCEL_NAME	-n, --name=job_name
SCANCEL_PARTITION	-p, --partition=partition_name
SCANCEL_QOS	-q, --qos=qos
SCANCEL_STATE	-t, --state=job_state_name
SCANCEL_USER	-u, --user=user_name
SCANCEL_VERBOSE	-v, --verbose
SCANCEL_WCKEY	--wckey=wckey
SLURM_CONF	(The location of the Slurm configuration file.)

EXAMPLES

Send SIGTERM to steps 1 and 3 of job 1234:

```
scancel --signal=TERM 1234.1 1234.3
```

Cancel job 1234 along with all of its steps:

```
scancel 1234
```

Send SIGKILL to all steps of job 1235, but do not cancel the job itself:

```
scancel --signal=KILL 1235
```

Send SIGUSR1 to the batch shell processes of job 1236:

```
scancel --signal=USR1 --batch 1236
```

Cancel job all pending jobs belonging to user "bob" in partition "debug":

```
scancel --state=PENDING --user=bob --partition=debug
```

Cancel only array ID 4 of job array 1237

```
scancel 1237_4
```

scontrol update

`scontrol` is used to view or modify Slurm configuration including: job, job step, node, partition, reservation, and overall system configuration. Most of the commands can only be executed by user root or an Administrator. If an attempt to view or modify configuration information is made by an unauthorized user, an error message will be printed and the requested action will not occur. If no command is entered on the execute line, `scontrol` will operate in an interactive mode and prompt for input. It will continue prompting for input and executing commands until explicitly terminated. If a command is entered on the execute line, `scontrol` will execute that command and terminate. All commands and options are case-insensitive, although node names, partition names, and reservation names are case-sensitive (node names "LX" and "lX" are distinct). All commands and options can be abbreviated to the extent that the specification is unique. A modified Slurm configuration can be written to a file using the `scontrol write config` command. The resulting file will be named using the convention "slurm.conf.<datetime>" and located in the same directory as the original "slurm.conf" file. The directory containing the original slurm.conf must be writable for this to occur.

scontrol update SPECIFICATION

Update job, step, node, partition, powercapping or reservation configuration per the supplied specification. `SPECIFICATION` is in the same format as the Slurm configuration file and the output of the `show` command described above. It may be desirable to execute the `show` command (described above) on the specific entity you want to update, then use cut-and-paste tools to enter updated configuration values to the update. Note that while most configuration values can be changed using this command, not all can be changed using this mechanism. In particular, the hardware configuration of a node or the physical addition or removal of nodes from the cluster may only be accomplished through editing the Slurm configuration file and executing the `reconfigure` command (described above).

SPECIFICATIONS FOR UPDATE COMMAND, JOBS

Note that update requests done by either root, SlurmUser or Administrators are not subject to certain restrictions. For instance, if an Administrator changes the QOS on a pending job, certain limits such as the `TimeLimit` will not be changed automatically as changes made by the Administrators are allowed to violate these restrictions.

`Account=<account>`

Account name to be changed for this job's resource use. Value may be cleared with blank data value, "Account="".

`AdminComment=<spec>`

Arbitrary descriptive string. Can only be set by a Slurm administrator.

`ArrayTaskThrottle=<count>`

Specify the maximum number of tasks in a job array that can execute at the same time. Set the count to zero in order to eliminate any limit. The task throttle count for a job array is reported as part of its `ArrayTaskId` field, preceded with a percent sign. For example "ArrayTaskId=1-10%2" indicates the

maximum number of running tasks is limited to 2.

BurstBuffer=<spec>

Burst buffer specification to be changed for this job's resource use. Value may be cleared with blank data value, "BurstBuffer=". Format is burst buffer plugin specific.

Clusters=<spec>

Specifies the clusters that the federated job can run on.

ClusterFeatures=<spec>

Specifies features that a federated cluster must have to have a sibling job submitted to it. Slurm will attempt to submit a sibling job to a cluster if it has at least one of the specified features.

Comment=<spec>

Arbitrary descriptive string.

Contiguous=<yes|no>

Set the job's requirement for contiguous (consecutive) nodes to be allocated. Possible values are "YES" and "NO". Only the Slurm administrator or root can change this parameter.

CoreSpec=<count>

Number of cores to reserve per node for system use. The job will be charged for these cores, but be unable to use them. Will be reported as "*" if not constrained.

CPUsPerTask=<count>

Change the CPUsPerTask job's value.

Deadline=<time_spec>

It accepts times of the form HH:MM:SS to specify a deadline to a job at a specific time of day (seconds are optional). You may also specify midnight, noon, fika (3 PM) or teatime (4 PM) and you can have a time-of-day suffixed with AM or PM for a deadline in the morning or the evening. You can specify a deadline for the job with a date of the form MMDDYY or MM/DD/YY or MM.DD.YY, or a date and time as YYYY-MM-DD[THH:MM[:SS]]. You can also give times like now + count time-units, where the time-units can be minutes, hours, days, or weeks and you can tell Slurm to put a deadline for tomorrow with the keyword tomorrow. The specified deadline must be later than the current time. Only pending jobs can have the deadline updated. Only the Slurm administrator or root can change this parameter.

DelayBoot=<time_spec>

Change the time to decide whether to reboot nodes in order to satisfy job's feature specification if the job has been eligible to run for less than this time period. See salloc/sbatch man pages option --delay-boot.

Dependency=<dependency_list>

Defer job's initiation until specified job dependency specification is satisfied. Cancel dependency with an empty dependency_list (e.g. "Dependency=").

<dependency_list> is of the form:

<type:job_id[:job_id][,type:job_id[:job_id]]>.

Many jobs can share the same dependency and these jobs may even belong to different users.

`after:job_id[:jobid...]`

This job can begin execution after the specified jobs have begun execution.

`afterany:job_id[:jobid...]`

This job can begin execution after the specified jobs have terminated.

`afternotok:job_id[:jobid...]`

This job can begin execution after the specified jobs have terminated in some failed state (non-zero exit code, node failure, timed out, etc).

`afterok:job_id[:jobid...]`

This job can begin execution after the specified jobs have successfully executed (ran to completion with an exit code of zero).

`singleton`

This job can begin execution after any previously launched jobs sharing the same job name and user have terminated. In other words, only one job by that name and owned by that user can be running or suspended at any point in time.

`EligibleTime=<time_spec>`

See `StartTime`.

`EndTime`

The time the job is expected to terminate based on the job's time limit. When the job ends sooner, this field will be updated with the actual end time.

`ExcNodeList=<nodes>`

Set the job's list of excluded node. Multiple node names may be specified using simple node range expressions (e.g. "lx[10-20]"). Value may be cleared with blank data value, "ExcNodeList="".

`Extra=<spec>`

An arbitrary string enclosed in double quotes if using spaces or some special characters.

`Features=<features>`

Set the job's required node features. The list of features may include multiple feature names separated by ampersand (AND) and/or vertical bar (OR) operators. For example: `Features="opteron&video"` or `Features="fast|faster"`. In the first example, only nodes having both the feature "opteron" AND the feature "video" will be used. There is no mechanism to specify that you want one node with feature "opteron" and another node with feature "video" in case no node has both features. If only one of a set of possible options should be used for all allocated nodes, then use the OR operator and enclose the options within square brackets. For example: `"Features=[rack1|rack2|rack3|rack4]"` might be used to specify that all nodes must be allocated on a single rack of the cluster, but any of those four racks can be used. A request can also specify the number of nodes needed with some feature by appending an asterisk and count after the feature name. For example `"Features=graphics*4"` indicates that at least four allocated nodes must have the feature "graphics." Parenthesis are also supported for features to be ANDed together. For example `"Features=[(knl&a2a&flat)*4&haswell*2]"` indicates the resource allocation should include 4 nodes with ALL of the features "knl", "a2a", and "flat" plus 2 nodes with the feature "haswell". Constraints with node counts may only be combined with AND operators. Value may be cleared with blank data value, for example `"Features=""`.

Gres=<list>

Specifies a comma delimited list of generic consumable resources. The format of each entry on the list is "name[:count[*cpu]]". The name is that of the consumable resource. The count is the number of those resources with a default value of 1. The specified resources will be allocated to the job on each node allocated unless "*cpu" is appended, in which case the resources will be allocated on a per cpu basis. The available generic consumable resources is configurable by the system administrator. A list of available generic consumable resources will be printed and the command will exit if the option argument is "help". Examples of use include "Gres=gpus:2*cpu,disk=40G" and "Gres=help".

JobId=<job_list>

Identify the job(s) to be updated. The job_list may be a comma separated list of job IDs. Either JobId or JobName is required.

Licenses=<name>

Specification of licenses (or other resources available on all nodes of the cluster) as described in salloc/sbatch/srun man pages.

MinCPUsNode=<count>

Set the job's minimum number of CPUs per node to the specified value.

MinMemoryCPU=<megabytes>

Set the job's minimum real memory required per allocated CPU to the specified value. Either MinMemoryCPU or MinMemoryNode may be set, but not both.

MinMemoryNode=<megabytes>

Set the job's minimum real memory required per node to the specified value. Either MinMemoryCPU or MinMemoryNode may be set, but not both.

MinTmpDiskNode=<megabytes>

Set the job's minimum temporary disk space required per node to the specified value. Only the Slurm administrator or root can change this parameter.

TimeMin=<timespec>

Change TimeMin value which specifies the minimum time limit minutes of the job.

JobName=<name>

Identify the name of jobs to be modified or set the job's name to the specified value. When used to identify jobs to be modified, all jobs belonging to all users are modified unless the UserID option is used to identify a specific user. Either JobId or JobName is required.

Name[=<name>]

See JobName.

Nice[=<adjustment>]

Update the job with an adjusted scheduling priority within Slurm. With no adjustment value the

scheduling priority is decreased by 100. A negative nice value increases the priority, otherwise decreases it. The adjustment range is +/- 2147483645. Only privileged users can specify a negative adjustment.

NodeList=<nodes>

Change the nodes allocated to a running job to shrink its size. The specified list of nodes must be a subset of the nodes currently allocated to the job. Multiple node names may be specified using simple node range expressions (e.g. "lx[10-20]"). After a job's allocation is reduced, subsequent srun commands must explicitly specify node and task counts which are valid for the new allocation.

NumCPUs=<min_count>[-<max_count>]

Set the job's minimum and optionally maximum count of CPUs to be allocated.

NumNodes=<min_count>[-<max_count>]

Set the job's minimum and optionally maximum count of nodes to be allocated. If the job is already running, use this to specify a node count less than currently allocated and resources previously allocated to the job will be relinquished. After a job's allocation is reduced, subsequent srun commands must explicitly specify node and task counts which are valid for the new allocation. Also see the NodeList parameter above. This is the same than ReqNodes.

NumTasks=<count>

Set the job's count of required tasks to the specified value. This is the same than ReqProcs.

OverSubscribe=<yes|no>

Set the job's ability to share compute resources (i.e. individual CPUs) with other jobs. Possible values are "YES" and "NO". This option can only be changed for pending jobs.

Partition=<name>

Set the job's partition to the specified value.

Priority=<number>

Set the job's priority to the specified value. Note that a job priority of zero prevents the job from ever being scheduled. By setting a job's priority to zero it is held. Set the priority to a non-zero value to permit it to run. Explicitly setting a job's priority clears any previously set nice value and removes the priority/multifactor plugin's ability to manage a job's priority. In order to restore the priority/multifactor plugin's ability to manage a job's priority, hold and then release the job. Only the Slurm administrator or root can increase job's priority.

QOS=<name>

Set the job's QOS (Quality Of Service) to the specified value. Value may be cleared with blank data value, "QOS="".

Reboot=<yes|no>

Set the job's flag that specifies whether to force the allocated nodes to reboot before starting the job. This is only supported with some system configurations and therefore it could be silently ignored.

ReqCores=<count>

Change the job's requested Cores count.

ReqNodeList=<nodes>

Set the job's list of required node. Multiple node names may be specified using simple node range expressions (e.g. "lx[10-20]"). Value may be cleared with blank data value, "ReqNodeList="".

ReqNodes=<min_count>[-<max_count>]

See NumNodes.

ReqProcs=<count>

See NumTasks.

ReqSockets=<count>

Change the job's requested socket count.

ReqThreads=<count>

Change the job's requested threads count.

Requeue=<0|1>

Stipulates whether a job should be requeued after a node failure: 0 for no, 1 for yes.

ReservationName=<name>

Set the job's reservation to the specified value. Value may be cleared with blank data value, "ReservationName="".

ResetAccrueTime

Reset the job's accrue time value to 0 meaning it will loose any time previously accrued for priority.

Helpful if you have a large queue of jobs already in the queue and want to start limiting how many jobs can accrue time without waiting for the queue to flush out.

StdOut=<filepath>

Set the batch job's stdout file path.

Shared=<yes|no>

See OverSubscribe option above.

StartTime=<time_spec>

Set the job's earliest initiation time. It accepts times of the form HH:MM:SS to run a job at a specific time of day (seconds are optional). (If that time is already past, the next day is assumed.) You may also specify midnight, noon, fika (3 PM) or teatime (4 PM) and you can have a time-of-day suffixed with AM or PM for running in the morning or the evening. You can also say what day the job will be run, by specifying a date of the form MMDDYY or MM/DD/YY or MM.DD.YY, or a date and time as YYYY-MM-DD[THH:MM[:SS]]. You can also give times like now + count time-units, where the time-units can be minutes, hours, days, or weeks and you can tell Slurm to run the job today with the keyword today and to run the job tomorrow with the keyword tomorrow.

Notes on date/time specifications:

- Although the 'seconds' field of the HH:MM:SS time specification is allowed by the code, note that the poll time of the Slurm scheduler is not precise enough to guarantee dispatch of the job on the exact second. The job will be eligible to start on the next poll following the specified time. The exact poll interval depends on the Slurm scheduler (e.g., 60 seconds with the default sched/builtin).
- If no time (HH:MM:SS) is specified, the default is (00:00:00).
- If a date is specified without a year (e.g., MM/DD) then the current year is assumed, unless the combination of MM/DD and HH:MM:SS has already passed for that year, in which case the next year is used.

Switches=<count>[@<max-time-to-wait>]

When a tree topology is used, this defines the maximum count of switches desired for the job allocation. If Slurm finds an allocation containing more switches than the count specified, the job remain pending until it either finds an allocation with desired switch count or the time limit expires. By default there is no switch count limit and no time limit delay. Set the count to zero in order to clean any previously set count (disabling the limit). The job's maximum time delay may be limited by the system administrator using the SchedulerParameters configuration parameter with the max_switch_wait parameter option. Also see wait-for-switch.

wait-for-switch=<seconds>

Change max time to wait for a switch <seconds> secs.

TasksPerNode=<count>

Change the job's requested TasksPerNode.

ThreadSpec=<count>

Number of threads to reserve per node for system use. The job will be charged for these threads, but be unable to use them. Will be reported as "*" if not constrained.

TimeLimit=<time>

The job's time limit. Output format is [days-]hours:minutes:seconds or "UNLIMITED". Input format (for update command) set is minutes, minutes:seconds, hours:minutes:seconds, days-hours, days hours:minutes or days-hours:minutes:seconds. Time resolution is one minute and second values are rounded up to the next minute. If changing the time limit of a job, either specify a new time limit value or precede the time and equal sign with a "+" or "-" to increment or decrement the current time limit (e.g. "TimeLimit+=30"). In order to increment or decrement the current time limit, the JobId specification must precede the TimeLimit specification. Only the Slurm administrator or root can increase job's TimeLimit.

UserID=<UID or name>

Used with the JobName option to identify jobs to be modified. Either a user name or numeric ID (UID), may be specified.

WCKey=<key>

Set the job's workload characterization key to the specified value.

SPECIFICATIONS FOR UPDATE COMMAND, STEPS

StepId=<job_id>[.<step_id>]

Identify the step to be updated. If the job_id is given, but no step_id is specified then all steps of the identified job will be modified. This specification is required.

CompFile=<completion file>

Update a step with information about a steps completion. Can be useful if step statistics aren't directly available through a jobacct_gather plugin. The file is a space-delimited file with format for Version 1 is as follows :

1	34461	0	2	0	3	1361906011	1361906015	1	1	3368	13357	/bin/sleep
A	B	C	D	E	F	G	H	I	J	K	L	M

Field Descriptions:

- A - file version
- B - ALPS apid
- C - Inblocks
- D - Outblocks
- E - Exit status
- F - Number of allocated CPUs
- G - Start time
- H - End time
- I - Utime
- J - Stime
- K - Maxrss
- L - UID
- M - Command name

TimeLimit=<time>

The job's time limit. Output format is [days-]hours:minutes:seconds or "UNLIMITED". Input format (for update command) set is minutes, minutes:seconds, hours:minutes:seconds, days-hours, days hours:minutes or days-hours:minutes:seconds. Time resolution is one minute and second values are rounded up to the next minute. If changing the time limit of a step, either specify a new time limit value or precede the time with a "+" or "-" to increment or decrement the current time limit (e.g. "TimeLimit=+30"). In order to increment or decrement the current time limit, the StepId specification must precede the TimeLimit specification.

SPECIFICATIONS FOR UPDATE COMMAND, NODES

NodeName=<name>

Identify the node(s) to be updated. Multiple node names may be specified using simple node range expressions (e.g. "lx[10-20]"). This specification is required.

ActiveFeatures=<features>

Identify the feature(s) currently active on the specified node. Any previously active feature specification will be overwritten with the new value. Also see AvailableFeatures. Typically ActiveFeatures will be identical to AvailableFeatures; however ActiveFeatures may be configured as a subset of the AvailableFeatures. For example, a node may be booted in multiple configurations. In that case, all possible

configurations may be identified as AvailableFeatures, while ActiveFeatures would identify the current node configuration.

AvailableFeatures=<features>

Identify the feature(s) available on the specified node. Any previously defined available feature specification will be overwritten with the new value. AvailableFeatures assigned via `scontrol` will only persist across the restart of the slurmctld daemon with the -R option and state files preserved or slurmctld's receipt of a SIGHUP. Update slurm.conf with any changes meant to be persistent across normal restarts of slurmctld or the execution of `scontrol reconfig`. Also see ActiveFeatures.

CpuBind=<node>

Specify the task binding mode to be used by default for this node. Supported options include: "none", "board", "socket", "ldom" (NUMA), "core", "thread" and "off" (remove previous binding mode).

Gres=<gres>

Identify generic resources to be associated with the specified node. Any previously defined generic resources will be overwritten with the new value. Specifications for multiple generic resources should be comma separated. Each resource specification consists of a name followed by an optional colon with a numeric value (default value is one) (e.g. "Gres=bandwidth:10000,gpus"). Generic resources assigned via `scontrol` will only persist across the restart of the slurmctld daemon with the -R option and state files preserved or slurmctld's receipt of a SIGHUP. Update slurm.conf with any changes meant to be persistent across normal restarts of slurmctld or the execution of `scontrol reconfig`.

Reason=<reason>

Identify the reason the node is in a "DOWN", "DRAINED", "DRAINING", "FAILING" or "FAIL" state. Use quotes to enclose a reason having more than one word.

State=<state>

Identify the state to be assigned to the node. Possible node states are "NoResp", "ALLOC", "ALLOCATED", "COMPLETING", "DOWN", "DRAIN", "FAIL", "FAILING", "FUTURE", "IDLE", "MAINT", "MIXED", "PERFCTRS/NPC", "RESERVED", "POWER_DOWN", "POWER_UP", "RESUME" or "UNDRAIN".

Not all of those states can be set using the `scontrol` command only the following can:

"CANCEL_REBOOT",

"DOWN", "DRAIN", "FAIL", "FUTURE", "RESUME", "NoResp", "POWER_DOWN", "POWER_UP" and "UNDRAIN". If a node is in a "MIXED" state it usually means the node is in multiple states. For instance if only part of the node is "ALLOCATED" and the rest of the node is "IDLE" the state will be "MIXED". If you want to remove a node from service, you typically want to set its state to "DRAIN". "CANCEL_REBOOT" cancels a pending reboot on the node (same as `scontrol cancel_reboot <node>`). "FAILING" is similar to

"DRAIN" except that some applications will seek to relinquish those nodes before the job completes. "PERFCTRS/NPC" indicates that Network Performance Counters associated with this node are in use, rendering this node as not usable for any other jobs. "RESERVED" indicates the node is in an advanced reservation and not generally available. "RESUME" is not an actual node state, but will change a node state from "DRAINED", "DRAINING", "DOWN" or "REBOOT" to either "IDLE" or "ALLOCATED" state as appropriate. "UNDRAIN" clears the node from being drained (like "RESUME"), but will not change the

node's base state (e.g. "DOWN"). Setting a node "DOWN" will cause all running and suspended jobs on that node to be terminated. "POWER_DOWN" and "POWER_UP" will use the configured SuspendProg and ResumeProg programs to explicitly place a node in or out of a power saving mode. If a node is already in the process of being powered up or down, the command will only change the state of the node but won't have any effect until the configured ResumeTimeout or SuspendTimeout is reached. Use of this command can be useful in situations where a ResumeProg like capmc in Cray machines is stalled and one wants to restore the node to "IDLE" manually, in this case rebooting the node and setting the state to "POWER_DOWN" will cancel the previous "POWER_UP" state and the node will become "IDLE". The "NoResp" state will only set the "NoResp" flag for a node without changing its underlying state. While all of the above states are valid, some of them are not valid new node states given their prior state. If the node state code printed is followed by "~", this indicates the node is presently in a power saving mode (typically running at reduced frequency). If the node state code is followed by "#", this indicates the node is presently being powered up or configured. If the node state code is followed by "\$", this indicates the node is currently in a reservation with a flag value of "maintenance". If the node state code is followed by "@", this indicates the node is currently scheduled to be rebooted. Generally only "DRAIN", "FAIL" and "RESUME" should be used. NOTE: The `scontrol` command should not be used to change node state on Cray systems. Use Cray tools such as `xtprocadmin` instead.

`Weight=<weight>`

Identify weight to be associated with specified nodes. This allows dynamic changes to weight associated with nodes, which will be used for the subsequent node allocation decisions. Weight assigned via `scontrol` will only persist across the restart of the `slurmctld` daemon with the `-R` option and state files preserved or `slurmctld`'s receipt of a `SIGHUP`. Update `slurm.conf` with any changes meant to be persistent across normal restarts of `slurmctld` or the execution of `scontrol reconfig`.

`scontrol show`

The `scontrol` command can not only be used to modify job parameters, but can also be a useful tool in identifying characteristics of existing jobs.

When using the `show` parameter, there are many areas to focus on. For example, you can use the `show ENTITY=ID`:

Display the state of the specified entity with the specified identification. `ENTITY` may be:

- aliases
- assoc_mgr
- bbstat
- burstbuffer
- config
- daemons
- dwstat
- federation
- frontend
- job
- node
- partition
- powercap

- reservation
- slurmd
- step
- topology
- hostlist
- hostlistsorted
- hostnames ID

In Slurm V23.02, scontrol show job has been updated - Requested TRES and allocated TRES will now always be printed when showing jobs, instead of one TRES output that was either the requested or allocated

sinfo

sinfo is the command used to view information about slurm nodes and partitions. The options are:

-a, --all

Display information about all partitions. This causes information to be displayed about partitions that are configured as hidden and partitions that are unavailable to user's group.

-d, --dead

If set only report state information for non-responding (dead) nodes.

-e, --exact

If set, do not group node information on multiple nodes unless their configurations to be reported are identical. Otherwise cpu count, memory size, and disk space for nodes will be listed with the minimum value followed by a "+" for nodes with the same partition and state (e.g., "250+").

--federation

Show all partitions from the federation if a member of one.

--future

Report nodes in FUTURE state.

-h, --noheader

Do not print a header on the output.

--help

Print a message describing all **sinfo** options.

--hide

Do not display information about hidden partitions. By default, partitions that are configured as hidden or are not available to the user's group will not be displayed (i.e. this is the default behavior).

-i <seconds>, --iterate=<seconds>

Print the state on a periodic basis. Sleep for the indicated number of seconds between reports. By default, prints a time stamp with the header.

--local

Show only jobs local to this cluster. Ignore other clusters in this federation. Overrides --federation.

-l, --long

Print more detailed information. This is ignored if the --format option is specified.

-M, --clusters=<string>

Clusters to issue commands to. Multiple cluster names may be comma separated. A value of 'all' will query to run on all clusters. Note that the SlurmDBD must be up for this option to work properly. This option implicitly sets the --local option.

-n <nodes>, --nodes=<nodes>

Print information only about the specified node(s). Multiple nodes may be comma separated or expressed using a node range expression. For example "linux[00-07]" would indicate eight nodes, "linux00" through "linux07." Performance of the command can be measurably improved for systems with large numbers of nodes when a single node name is specified.

--noconvert

Don't convert units from their original type (e.g. 2048M won't be converted to 2G).

-N, --Node

Print information in a node-oriented format with one line per node and partition. That is, if a node belongs to more than one partition, then one line for each node-partition pair will be shown. If --partition is also specified, then only one line per node in this partition is shown. The default is to print information in a partition-oriented format. This is ignored if the --format option is specified.

-o <output_format>, --format=<output_format>

Specify the information to be displayed using an `sinfo` format string. Format strings transparently used by `sinfo` when running with various options are:

default:

"%#P %.5a %.10l %.6D %.6t %N"

--summarize

"%#P %.5a %.10l %.16F %N"

--long

"%#P %.5a %.10l %.10s %.4r %.8h %.10g %.6D %.11T %N"

--Node

"%#N %.6D %#P %6t"

--long --Node

"%#N %.6D %#P %.11T %.4c %.8z %.6m %.8d %.6w %.8f %20E"

--list-reasons

"%20E %9u %19H %N"

--long --list-reasons

"%20E %12U %19H %6t %N"

In the above format strings, the use of "#" represents the maximum length of any partition name or node list to be printed. A pass is made over the records to be printed to establish the size in order to align the `sinfo` output,

then a second pass is made over the records to print them. Note that the literal character "#" itself is not a valid field length specification, but is only used to document this behaviour.

The field specifications available include:

%all	Print all fields available for this data type with a vertical bar separating each field.
%a	State/availability of a partition
%A	Number of nodes by state in the format "allocated/idle". Do not use this with a node state option ("%" or "%T") or the different node states will be placed on separate lines.
%b	Features currently active on the nodes, also see %f
%B	The max number of CPUs per node available to jobs in the partition.
%c	Number of CPUs per node
%C	Number of CPUs by state in the format "allocated/idle/other/total". Do not use this with a node state option ("%" or "%T") or the different node states will be placed on separate lines.
%d	Size of temporary disk space per node in megabytes
%D	Number of nodes
%e	Free memory of a node
%E	The reason a node is unavailable (down, drained, or draining states).
%f	Features available the nodes, also see %b
%F	Number of nodes by state in the format "allocated/idle/other/total". Note the use of this format option with a node state format option ("%" or "%T") will result in the different node states being reported on separate lines.
%g	Groups which may use the nodes
%G	Generic resources (gres) associated with the nodes
%h	Jobs may oversubscribe compute resources (i.e. CPUs), "yes", "no", "exclusive" or "force"
%H	Print the timestamp of the reason a node is unavailable.
%I	Partition job priority weighting factor.
%I	Maximum time for any job in the format "days-hours:minutes:seconds"
%L	Default time for any job in the format "days-hours:minutes:seconds"
%m	Size of memory per node in megabytes
%M	PreemptionMode
%n	List of node hostnames
%N	List of node names
%o	List of node communication addresses
%O	CPU load of a node
%p	Partition scheduling tier priority.
%P	Partition name followed by "*" for the default partition, also see %R
%r	Only user root may initiate jobs, "yes" or "no"
%R	Partition name, also see %P
%s	Maximum job size in nodes
%S	Allowed allocating nodes
%t	State of nodes, compact form
%T	State of nodes, extended form
%u	Print the user name of who set the reason a node is unavailable.
%U	Print the user name and uid of who set the reason a node is unavailable.
%v	Print the version of the running slurmd daemon.
%V	Print the cluster name if running in a federation
%w	Scheduling weight of the nodes

%X Number of sockets per node
 %Y Number of cores per socket
 %Z Number of threads per core
 %z Extended processor information: number of sockets, cores, threads (S:C:T) per node
 %.<*> right justification of the field
 %<Number><*> size of field

-O <output_format>, --Format=<output_format>

Specify the information to be displayed. Also see the -o <output_format>, --format=<output_format> option described below (which supports greater flexibility in formatting, but does not support access to all fields because we ran out of letters). Requests a comma separated list of job information to be displayed. The format of each field is "type[:[.]size]"

size is the minimum field size. If no size is specified, 20 characters will be allocated to print the information.

A dot (.) indicates the output should be right justified and size must be specified. By default, output is left justified. Valid type specifications include:

<i>all</i>	Print all fields available in the -o format for this data type with a vertical bar separating
<i>each</i>	field.
<i>allocmem</i>	Prints the amount of allocated memory on a node.
<i>allocnodes</i>	Allowed allocating nodes.
<i>available</i>	State/availability of a partition.
<i>cluster</i>	Print the cluster name if running in a federation
<i>cpus</i>	Number of CPUs per node.
<i>cpusload</i>	CPU load of a node.
<i>freemem</i>	Free memory of a node.
<i>cpusstate</i>	Number of CPUs by state in the format "allocated/idle/other/total". Do not use this with a node state option ("%t" or "%T") or the different node states will be placed on separate lines.
<i>cores</i>	Number of cores per socket.
<i>defaulttime</i>	Default time for any job in the format "days-hours:minutes:seconds".
<i>disk</i>	Size of temporary disk space per node in megabytes.
<i>features</i>	Features available on the nodes. Also see <i>features_act</i> .
<i>features_act</i>	Features currently active on the nodes. Also see <i>features</i> .
<i>groups</i>	Groups which may use the nodes.
<i>gres</i>	Generic resources (gres) associated with the nodes.
<i>maxcpuspernode</i>	The max number of CPUs per node available to jobs in the partition.
<i>memory</i>	Size of memory per node in megabytes.
<i>nodes</i>	Number of nodes.
<i>nodeaddr</i>	List of node communication addresses.
<i>nodeai</i>	Number of nodes by state in the format "allocated/idle". Do not use this with a node state option ("%t" or "%T") or the different node states will be placed on separate lines.
<i>nodeaiot</i>	Number of nodes by state in the format "allocated/idle/other/total". Do not use this

	with a node state option ("%t" or "%T") or the different node states will be placed on separate lines.
<i>nodehost</i>	List of node hostnames.
<i>nodelist</i>	List of node names.
<i>oversubscribe</i>	Jobs may oversubscribe compute resources (i.e. CPUs), "yes", "no", "exclusive" or "force".
<i>partition</i>	Partition name followed by "*" for the default partition, also see %R.
<i>partitionname</i>	Partition name, also see %P.
<i>port</i>	Node TCP port.
<i>preemptmode</i>	PreemptionMode.
<i>priorityjobfactor</i>	Partition factor used by priority/multifactor plugin in calculating job priority.
<i>prioritytier</i> or <i>priority</i>	Partition scheduling tier priority.
<i>reason</i>	The reason a node is unavailable (down, drained, or draining states).
<i>root</i>	Only user root may initiate jobs, "yes" or "no".
<i>size</i>	Maximum job size in nodes.
<i>statecompact</i>	State of nodes, compact form.
<i>statelong</i>	State of nodes, extended form.
<i>sockets</i>	Number of sockets per node.
<i>socketcorethread</i>	Extended processor information: number of sockets, cores, threads (S:C:T) per node.
<i>time</i>	Maximum time for any job in the format "days-hours:minutes:seconds".
<i>timestamp</i>	Print the timestamp of the reason a node is unavailable.
<i>threads</i>	Number of threads per core.
<i>user</i>	Print the user name of who set the reason a node is unavailable.
<i>userlong</i>	Print the user name and uid of who set the reason a node is unavailable.
<i>version</i>	Print the version of the running slurmd daemon.
<i>weight</i>	Scheduling weight of the nodes.

-p <partition>, --partition=<partition>

Print information only about the specified partition(s). Multiple partitions are separated by commas.

-r, --responding

If set only report state information for responding nodes.

-R, --list-reasons

List reasons nodes are in the down, drained, fail or failing state. When nodes are in these states Slurm supports optional inclusion of a "reason" string by an administrator. This option will display the first 20 characters of the reason field and list of nodes with that reason for all nodes that are, by default, down, drained, draining or failing. This option may be used with other node filtering options (e.g. -r, -d, -t, -n), however, combinations of these options that result in a list of nodes that are not down or drained or failing will not produce any output. When used with -l the output additionally includes the current node state.

-s, --summarize

List only a partition state summary with no node state details. This is ignored if the --format option is specified.

-S <sort_list>, --sort=<sort_list>

Specification of the order in which records should be reported. This uses the same field specification as the <output_format>. Multiple sorts may be performed by listing multiple sort fields separated by commas. The field specifications may be preceded by "+" or "-" for ascending (default) and descending order respectively. The partition field specification, "P", may be preceded by a "#" to report partitions in the same order that they appear in Slurm's configuration file, slurm.conf. For example, a sort value of "+P,-m" requests that records be printed in order of increasing partition name and within a partition by decreasing memory size. The default value of sort is "#P,-t" (partitions ordered as configured then decreasing node state). If the --Node option is selected, the default sort value is "N" (increasing node name).

-t <states> , --states=<states>

List nodes only having the given state(s). Multiple states may be comma separated and the comparison is case insensitive. Possible values include (case insensitive): ALLOC, ALLOCATED, COMP, COMPLETING, DOWN, DRAIN (for node in DRAINING or DRAINED states), DRAINED, DRAINING, FAIL, FUTURE, FUTR, IDLE, MAINT, MIX, MIXED, NO_RESPOND, NPC, PERFCTRS, POWER_DOWN, POWER_UP, RESV, RESERVED, UNK, and UNKNOWN. By default nodes in the specified state are reported whether they are responding or not. The --dead and --responding options may be used to filtering nodes by the responding flag.

-T, --reservation

Only display information about Slurm reservations.

--usage

Print a brief message listing the `sinfo` options.

-v, --verbose

Provide detailed event logging through program execution.

-V, --version

Print version information and exit.

svIEW

`svIEW` can be used to view Slurm configuration, job, step, node and partitions state information. Authorized users can also modify select information.

The primary display modes are Jobs and Partitions, each with a selection tab. There is also an optional map of the nodes on the left side of the window which will show the nodes associated with each job or partition. Left-click on the tab of the display you would like to see. Right-click on the tab in order to control which fields will be displayed. Within the display window, left-click on the header to control the sort order of entries (e.g. increasing or decreasing) in the display.

You can also left-click and drag the headers to move them right or left in the display. If a JobID has an arrow next to it, click on that arrow to display or hide information about that job's steps. Right-click on a line of the display to get more information about the record.

There is an Admin Mode option which permits the user root to modify many of the fields displayed, such as node state or job time limit. In the mode, a Slurm Reconfigure Action is also available. It is recommended that Admin Mode be used only while modifications are actively being made. Disable Admin Mode immediately after the changes to avoid possibly making unintended changes.

API Changes for Slurm V23.02

Partial support for '--json' and '--yaml' formated outputs have been implemented for sacctmgr, sdiag, sinfo, squeue, and scontrol. The resultant data ouput will be filtered by normal command arguments. Formatting arguments will continue to be ignored.

Managing Jobs Exercises

In this set of labs, you will learn how to manage jobs using job modication and viewing utilities

Exercise 1: Modify a job using scontrol

In this lab you will modify the nodes of a running job so that a pending job will run, all using the scontrol command

1. As fred, submit a job that will take all 10 nodes (specifically node00-node09) for 2 hours:

```
sbatch -wnode[00-09] -n40 --mem=1000 -t120 --wrap="sleep 7200"
```

2. Now submit a 2-node job, which will go pending because all the nodes are in use currently:

```
sbatch -wnode[08-09] -n8 --mem=1000 -t120 --wrap="sleep 7200"
```

3. Run squeue to see the queue:

```
squeue -l
```

Should show:

Wed Nov 27 00:05:22 2019								
JOBid	PARTITION	NAME	USER	STATE	TIME	TIME_LIMI	NODES	NODELIST (REASON)
2	debug	wrap	fred	PENDING	0:00	2:00:00	2	(Resources)
1	debug	wrap	fred	RUNNING	0:51	2:00:00	10	node[00-09]

4. Modify the first job so that we can free up 2 nodes for the 2nd job to run:

```
scontrol update jobid=1 NumNodes=8
```

Note: Both jobs should now be running.

Also note: As you remove nodes from running jobs, your job may or may not like it.

Depending on the type of workload, this may not even be an option for you.

5. Cancel the jobs:

```
scancel -u fred
```

Exercise 2: Retrieving job information using scontrol

In this lab you will use scontrol to gather detailed job information

1. As fred, submit a job that will take all 10 nodes (specifically node00-node09) for 2 hours:

```
sbatch -wnode[00-09] -n40 --mem=1000 -t120 --wrap="sleep 7200"
```

2. Make a note of the job id number.

NOTE: You will need to replace “**JOBID**” with that value in the next step.

3. Now, using scontrol, show the details about the job:

```
scontrol -d show job JOBID
```

NOTE: I like the `-d` switch with `scontrol` on a running job, because it shows which CPU ID's are allocated to the job

It should show something similar to the following:

```
JobId=3 JobName=wrap
  UserId=fred(1010) GroupId=users(100) MCS_label=N/A
  Priority=4294901757 Nice=0 Account=bedrock QOS=normal
  JobState=RUNNING Reason=None Dependency=(null)
  Requeue=1 Restarts=0 BatchFlag=1 Reboot=0 ExitCode=0:0
  DerivedExitCode=0:0
  RunTime=00:00:14 TimeLimit=02:00:00 TimeMin=N/A
  SubmitTime=2022-06-02T12:41:01 EligibleTime=2022-06-02T12:41:01
  AccrueTime=2022-06-02T12:41:01
  StartTime=2022-06-02T12:41:02 EndTime=2022-06-02T14:41:02 Deadline=N/A
  SuspendTime=None SecsPreSuspend=0 LastSchedEval=2022-06-02T12:41:02 Scheduler=Main
  Partition=debug AllocNode:Sid=login:158
  ReqNodeList=node[00-09] ExcNodeList=(null)
  NodeList=node[00-09]
  BatchHost=node00
  NumNodes=10 NumCPUs=40 NumTasks=40 CPUs/Task=1 ReqB:S:C:T=0:0:0:*
  ReqTRES=cpu=40,mem=10000M,node=10,billing=40
  AllocTRES=cpu=40,mem=10000M,node=10,billing=40
  Socks/Node=* NtasksPerN:B:S:C=0:0:0:0 CoreSpec=*
  JOB_GRES=(null)
    Nodes=node[00-09] CPU_IDS=0-3 Mem=1000 GRES=
    MinCPUsNode=1 MinMemoryNode=1000M MinTmpDiskNode=0
    Features=(null) DelayBoot=00:00:00
    OverSubscribe=OK Contiguous=0 Licenses=(null) Network=(null)
    Command=(null)
    WorkDir=/home/fred
    StdErr=/home/fred/slurm-3.out
    StdIn=/dev/null
    StdOut=/home/fred/slurm-3.out
    Power=
```

This information can be very useful in determining if a job has failed for some reason. The most relevant is the “Reason” value and “JobState” fields.

4. For example, issue the following invalid job submission:

```
sbatch -wnode[00-09] -n41 --mem=1000 -t120 --wrap="sleep 7200" (NOTICE
-n41, which is invalid.)
```

5. Make a note of the batch job number (for **JOBID**, below):

NOTE: This job will be accepted to the queue, but will go to the state "Pending" for reason of PartitionConfig, meaning there is no partition configured to accept that allocation request)

6. Check it out with `scontrol show job` (replace **JOBID** with the job id number from above):

```
scontrol show job JOBID | grep -C 100 --color=auto PartitionConfig
```

Should show something similar to the following:

```
JobId=4 JobName=wrap
UserId=fred(1010) GroupId=users(100) MCS_label=N/A
Priority=4294901756 Nice=0 Account=bedrock QOS=normal
JobState=PENDING Reason=PartitionConfig Dependency=(null)
Requeue=1 Restarts=0 BatchFlag=1 Reboot=0 ExitCode=0:0
RunTime=00:00:00 TimeLimit=02:00:00 TimeMin=N/A
SubmitTime=2022-06-02T12:41:58 EligibleTime=2022-06-02T12:41:58
AccrueTime=2022-06-02T12:41:58
StartTime=Unknown EndTime=Unknown Deadline=N/A
SuspendTime=None SecsPreSuspend=0 LastSchedEval=2022-06-02T12:41:58
Scheduler=Main
Partition=debug AllocNode:Sid=login:158
ReqNodeList=node[00-09] ExcNodeList=(null)
 NodeList=(null)
NumNodes=11 NumCPUs=41 NumTasks=41 CPUs/Task=1 ReqB:S:C:T=0:0:0:*
TRES=cpu=41,mem=10000M,node=10,billing=41
Socks/Node=* NtasksPerN:B:S:C=0:0:0: CoreSpec=*
MinCPUSNode=1 MinMemoryNode=1000M MinTmpDiskNode=0
Features=(null) DelayBoot=00:00:00
OverSubscribe=OK Contiguous=0 Licenses=(null) Network=(null)
Command=(null)
WorkDir=/home/fred
StdErr=/home/fred/slurm-4.out
StdIn=/dev/null
StdOut=/home/fred/slurm-4.out
Power=
```

NOTE: The JobState and Reason are key in troubleshooting. First, the JobState is PENDING because the controller cannot allocate 41 tasks. There are only 40 cores in this system. The Reason states “PartitionConfig” because the partition, which defines the resources on the system, isn’t configured for 41 cores.

Cleanup

1. Reset the Docker Slurm cluster environment back to its original, pristine state by exiting the login Docker container back to the Ubuntu AWS prompt. And **as the ubuntu user**:

```
cd ~/docker-scale-out
make clean && make
```

Slurm Accounting and Reporting Discussion

Slurm has a few tools for displaying utilization, as collected in the Slurm database. How often the data gets collected is generally on the hour. Once the usage information has been “Rolled up” into the database (for sreport), it is a simple matter of framing a query around a time frame.

The main reporting tools are:

- **sreport** Generate reports from the slurm accounting data.
- **sstat** Display various status information of a running job/step.
- **sacct** Displays accounting data for all jobs and job steps in the Slurm job accounting log or Slurm database

sreport

This is the most useful general tool for observing usage across the cluster. Using a variety of switches, you can format the output in any number of ways.

Here is an example of listing the number of CPU minutes used by account (UserUtilizationByAccount) between 6am this morning (start=2019-06-04T05:59:59) and 1pm this afternoon (end=2019-06-04T12:59:59):

```
# sreport cluster UserUtilizationByAccount start=2019-06-04T05:59:59 end=2019-06-04T12:59:59
-----
Cluster/User/Account Utilization 2019-06-04T06:00:00 - 2019-06-04T12:59:59 (25200 secs)
Usage reported in CPU Minutes
-----
Cluster      Login      Proper Name      Account      Used      Energy
-----
cluster      fred        bedrock        30          0
cluster      bambam     bedrock        5           0
cluster      pebbles    bedrock        2           0
cluster      wilma      bedrock        0           0
```

It shows that during that period of time, 37 CPU minutes were consumed on the cluster: 30 by fred (the heavy user during this time period), 5 by bambam, and 2 by pebbles.

A simple twist on this report would be to show the Utilization in an account listed by user (AccountUtilizationByUser) over the same time period:

```
# sreport cluster AccountUtilizationByUser start=2019-06-04T05:59:59 end=2019-06-04T12:59:59
-----
Cluster/Account/User Utilization 2019-06-04T06:00:00 - 2019-06-04T12:59:59 (25200 secs)
Usage reported in CPU Minutes
-----
Cluster      Account      Login      Proper Name      Used      Energy
-----
cluster      root        38          0
cluster      bedrock     38          0
cluster      bedrock     bambam     5           0
cluster      bedrock     fred        30         0
cluster      bedrock     pebbles    2           0
cluster      bedrock     wilma      0           0
```

The reason the Energy label is printed is because since 2 revisions of Slurm ago, it was added and is part of the default output. You can, of course, format your output with what any fields you want (show only Cluster, Login and Used columns):

```
# sreport cluster AccountUtilizationByUser start=2019-06-04T05:59:59 end=2019-06-04T12:59:59
format=Cluster,Login,Used
-----
Cluster/Account/User Utilization 2019-06-04T06:00:00 - 2019-06-04T12:59:59 (25200 secs)
Usage reported in CPU Minutes
-----
Cluster      Login      Used
-----
cluster            38
cluster            38
cluster      bambam      5
cluster      fred        30
cluster    pebbles       2
cluster      wilma        0
```

Another way to report is by percent utilization.

```
# sreport -t hourper cluster Utilization start=2019-06-04T05:59:59 End=2019-06-04T12:59:59
-----
Cluster Utilization 2019-06-04T06:00:00 - 2019-06-04T12:59:59
Usage reported in CPU Hours/Percentage of Total
-----
Cluster          Allocated          Down          PLND Down          Idle
Reserved          Reported
-----
cluster           1 (0.45%)         0 (0.00%)        0 (0.00%)      134 (95.69%)
5 (3.87%)      140 (100.00%)
```

This is a little hard to read because of the wrapping text. But you can see that during our time period, there were 134 total CPU hours in which the cluster was idle (about 95% of the time reported) out of a total of 140 CPU hours possible (6am-1pm=7 hours x 5 nodes x 4 CPUs per node= $7*5*4=140$)

The number of jobs can also be calculated by running a job report, and adding SizesByAccount and PrintJobCount flags:

```
# sreport job SizesByAccount PrintJobCount start=2019-06-04T05:59:59 End=2019-06-04T12:59:59
-----
Job Sizes 2019-06-04T06:00:00 - 2019-06-04T12:59:59 (25200 secs)
Units are in number of jobs ran
-----
Cluster      Account      0-49 CPUs      50-249 CPUs      250-499 CPUs      500-999 CPUs      >= 1000 CPUs % of cluster
-----
cluster      bedrock        2215             0                 0                 0                 0
```

So a total of 2215 jobs were submitted during this time period. All of those jobs requested 0-49 Cpus

In Slurm v23.02, the report command has been changed to modify the count planned (FKA reserved) time for jobs running in IGNORE_JOBS reservations. Previously, this was lumped into IDLE time.

sstat

sstat is used to gather information on running jobs

There is a wealth of accounting information presented in the output:

```
# sstat 17701
      JobID  MaxVMSize  MaxVMSizeNode  MaxVMSizeTask  AveVMSize      MaxRSS  MaxRSSNode
MaxRSSTask      AveRSS  MaxPages  MaxPagesNode  MaxPagesTask  AvePages      MinCPU  MinCPUNode
MinCPUTask      AveCPU   NTasks  AveCPUFreq  ReqCPUFreqMin  ReqCPUFreqMax  ReqCPUFreqGov
ConsumedEnergy  MaxDiskRead  MaxDiskReadNode  MaxDiskReadTask  AveDiskRead  MaxDiskWrite
MaxDiskWriteNode  MaxDiskWriteTask  AveDiskWrite  TRESUsageInAve  TRESUsageInMax
TRESUsageInMaxNode  TRESUsageInMaxTask  TRESUsageInMin  TRESUsageInMinNode  TRESUsageInMinTask
TRESUsageInTot  TRESUsageOutAve  TRESUsageOutMax  TRESUsageOutMaxNode  TRESUsageOutMaxTask
TRESUsageOutMin  TRESUsageOutMinNode  TRESUsageOutMinTask  TRESUsageOutTot
-----
-----
-----
-----
-----
17701.extern
213503982+          0           0           0           0           0           0
```

As in the previous tool (**sreport**), you can manipulate the output using the format flag

sacct (end-user) and sacctmgr (administrator)

This command interfaces directly (though a slurmdbd call) to the database. It is used to report on any value in the database or table in the database. For example, to see the users defined in the database, you can use:

```
# sacctmgr show user
  User  Def Acct     Admin
-----
arnold    bedrock    None
bambam    bedrock    None
barney    bedrock    None
betty     bedrock    None
chip      bedrock    None
dino      bedrock    None
edna      bedrock    None
fred      bedrock    None
gazoo     bedrock    None
pebbles   bedrock    None
root      root Administ+
wilma    bedrock    None
```

The power also comes in showing the associations between the objects in the database:

```
# sacctmgr show assoc tree | less -FSR
  Cluster    Account      User Partition     Share GrpJobs      GrpTRES GrpSubmit
  GrpWall   GrpTRESMins MaxJobs       MaxTRES MaxTRESPerNode MaxSubmit      MaxWall
MaxTRESMins                               QOS   Def QOS GrpTRESRunMin
----- ----- ----- ----- ----- ----- ----- ----- ----- ----- ----- ----- -----
```

Cluster	Account	User	Partition	Share	GrpJobs	GrpTRES	GrpSubmit
normal	cluster	root				1	
normal	cluster	root	root			1	
normal	cluster	bedrock				1	
normal	cluster	bedrock	arnold			1	
normal	cluster	bedrock	bambam			1	
normal	cluster	bedrock	barney			1	
normal	cluster	bedrock	betty			1	
normal	cluster	bedrock	chip			1	
normal	cluster	bedrock	dino			1	
normal	cluster	bedrock	edna			1	
normal	cluster	bedrock	fred			1	
normal	cluster	bedrock	gazoo			1	
normal	cluster	bedrock	pebbles			1	
normal	cluster	bedrock	wilma			1	

In Slurm V23.02, sacct renamed 'Reserved' field to 'Planned' to match sreport and the nomenclature of the 'Planned' node

Accounting and Reporting Exercises

In this set of labs, you will learn how to query the Slurm database for job accounting records.

Exercise 1: Use the `sstat` command to view job information stored in the database:

Use `sstat` to view the status information for running jobs invoked with Slurm.

1. Open a couple of terminals.

2. In one terminal window, as fred, submit an interactive job as follows:

```
srun -n4 -N1 --mem=1000 --acctg-freq=filesystem=1 --pty bash
```

3. Use `squeue` to get the running jobID

NOTE: It's probably 1, since you just cleaned the environment after the last lab).

4. In another terminal window, issue the following sstat command (**the following is all one command, wrapped around in this document**):

```
sstat -o \
jobid,maxvmsize,ntasks%7,avecpufreq,maxdiskread,\
maxdiskwrite,tresusageouttot%25 -j 1
```

It should show something like this:

JobID	MaxVMSize	NTasks	AveCPUFreq	MaxDiskRead	MaxDiskWrite	TRESUsageOutTot
1.0	1. 980K	1	2.59M	133495	861.	energy=0, fs/disk=861

5. Run it again through watch (**the following is all one command, wrapped around in this document**):

```
watch -n .3 "sstat -o\
jobid,maxvmsize,ntasks%7,avecpufreq,maxdiskread,\
maxdiskwrite,tresusageouttot%25 -j 1"
```

6. Now let's put some load on the job. In the first terminal window, enter the following to generate some file system traffic (**the following is all one command, wrapped around in this document**):

```
for x in {1..50} ; do dd if=/dev/zero of=bigfile bs=1k \
count=1k ; sleep .5 ; done
```

7. Observe the Disk Write i/o statistics of the job in the sstat command.

8. Exit the allocation.

Exercise 2: Use the `sacct` command to view accounting database information:

Use `sacct` to view the accounting information with Slurm.

1. Open a couple of terminals (use the ones from Exercise 1).

- In one terminal window, **as fred**, submit another interactive job like in the previous exercise as follows:
`srun --mem=2000 -n4 -N1 --acctg-freq=filesystem=1 --pty bash`

- In the second window, use `squeue` to get the running jobID.

- In the second window, run sacct through watch (**the following is all one command, wrapped around in this document**):

```
watch sacct -o \
JobID%10,JobName,MaxVMSize, \
AveCPU,Ntasks,AllocCPUS,State,ReqMem,MaxDiskWrite
```

- Switch back to the first window and submit another interactive job (i.e. perform and action in the interactive shell that is already open) that adds load to the node, as follows (**the following is all one command, wrapped around in this document**):

```
for x in {1..30} ; do dd if=/dev/zero of=bigfile bs=1k count=1k \
; sleep .5 ; done
```

- Exit out of the allocation, and observe the i/o statistics of the job in the sacct command.

NOTE: Accounting information relating to jobs is bundled up after the job completes. So you won't see much in this watch output until the job completes

- Exit the allocation, and now look at the sacct output.

It should look something like this:

Every 2.0s: sacct -o JobID%10,JobName,MaxVMSize,AveCPU,Ntasks,AllocCPUS,State,ReqMem,MaxDiskWrite								
JobID	JobName	MaxVMSize	AveCPU	NTasks	AllocCPUS	State	ReqMem	MaxDiskWrite
1	bash				4	COMPLETED	2000M	
1.extern	extern	23048K	00:00:00	1	4	COMPLETED		0.00M
1.0	bash	710316K	00:00:01	4	4	COMPLETED		976.56M
2	bash				4	COMPLETED	2000M	
2.extern	extern	23048K	00:00:00	1	4	COMPLETED		0.00M
2.0	bash	710316K	00:00:00	4	4	COMPLETED		976.56M

Cleanup

- Reset the Docker Slurm cluster environment back to its original, pristine state by exiting the login Docker container back to the Ubuntu AWS prompt. And **as ubuntu**:

```
cd ~/docker-scale-out
make clean && make
```

Topology and Application Layout Discussion

Slurm can be configured to support topology-aware resource allocation to optimize job performance. Slurm supports several modes of operation, one to optimize performance on systems with a three-dimensional torus interconnect and another for a hierarchical interconnect. The hierarchical mode of operation supports both [fat-tree](#) or [dragonfly](#) networks, using slightly different algorithms.

Slurm's native mode of resource selection is to consider the nodes as a one-dimensional array. Jobs are allocated resources on a best-fit basis. For larger jobs, this minimizes the number of sets of consecutive nodes allocated to the job.

Three-dimension Topology

Some larger computers rely upon a three-dimensional [torus](#) interconnect. The Cray XT and XE systems also have three-dimensional torus interconnects, but do not require that jobs execute in adjacent nodes. On those systems, Slurm only needs to allocate resources to a job which are nearby on the network. Slurm accomplishes this using a Hilbert curve to map the nodes from a three-dimensional space into a one-dimensional space. Slurm's native best-fit algorithm is thus able to achieve a high degree of locality for jobs.

Hierarchical Networks

Slurm can also be configured to allocate resources to jobs on a hierarchical network to minimize network contention. The basic algorithm is to identify the lowest level switch in the hierarchy that can satisfy a job's request and then allocate resources on its underlying leaf switches using a best-fit algorithm. Use of this logic requires a configuration setting of `TopologyPlugin=topology/tree`.

Note that slurm uses a best-fit algorithm on the currently available resources. This may result in an allocation with more than the optimum number of switches. The user can request a maximum number of switches for the job as well as a maximum time willing to wait for that number using the `--switches` option with the `salloc`, `sbatch` and `srun` commands. The parameters can also be changed for pending jobs using the `scontrol` and `squeue` commands.

At some point in the future Slurm code may be provided to gather network topology information directly. Now the network topology information must be included in a `topology.conf` configuration file as shown in the examples below. The first example describes a three level switch in which each switch has two children. Note that the `SwitchName` values are arbitrary and only used for bookkeeping purposes, but a name must be specified on each line. The leaf switch descriptions contain a `SwitchName` field plus a `Nodes` field to identify the nodes connected to the switch. Higher-level switch descriptions contain a `SwitchName` field plus a `Switches` field to identify the child switches. Slurm's hostlist expression parser is used, so the node and switch names need not be consecutive (e.g. "Nodes=tux[0-3,12,18-20]" and "Switches=s[0-2,4-8,12]" will parse fine).

An optional `LinkSpeed` option can be used to indicate the relative performance of the link. The units used are arbitrary and this information is currently not used. It may be used in the future to optimize resource allocations.

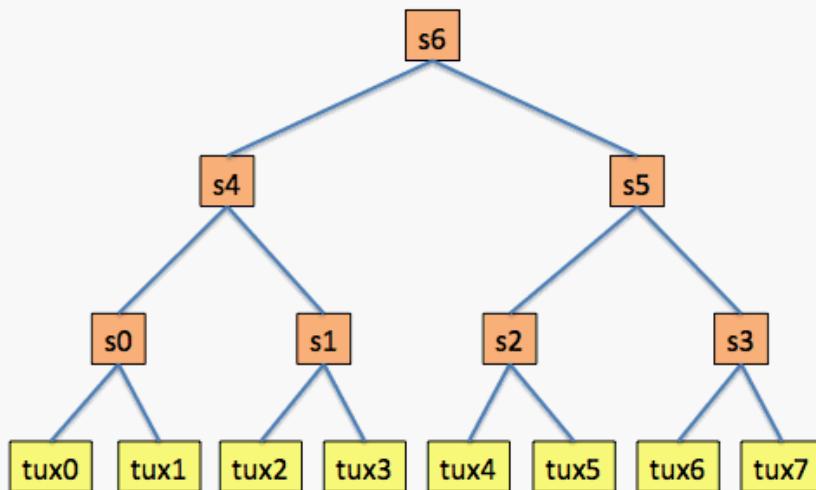
The first example shows what a topology would look like for an eight node cluster in which all switches have only two children as shown in the diagram (not a very realistic configuration, but useful for an example).

```
# topology.conf
# Switch Configuration
SwitchName=s0 Nodes=tux[0-1]
SwitchName=s1 Nodes=tux[2-3]
```

```

SwitchName=s2 Nodes=tux[4-5]
SwitchName=s3 Nodes=tux[6-7]
SwitchName=s4 Switches=s[0-1]
SwitchName=s5 Switches=s[2-3]
SwitchName=s6 Switches=s[4-5]

```

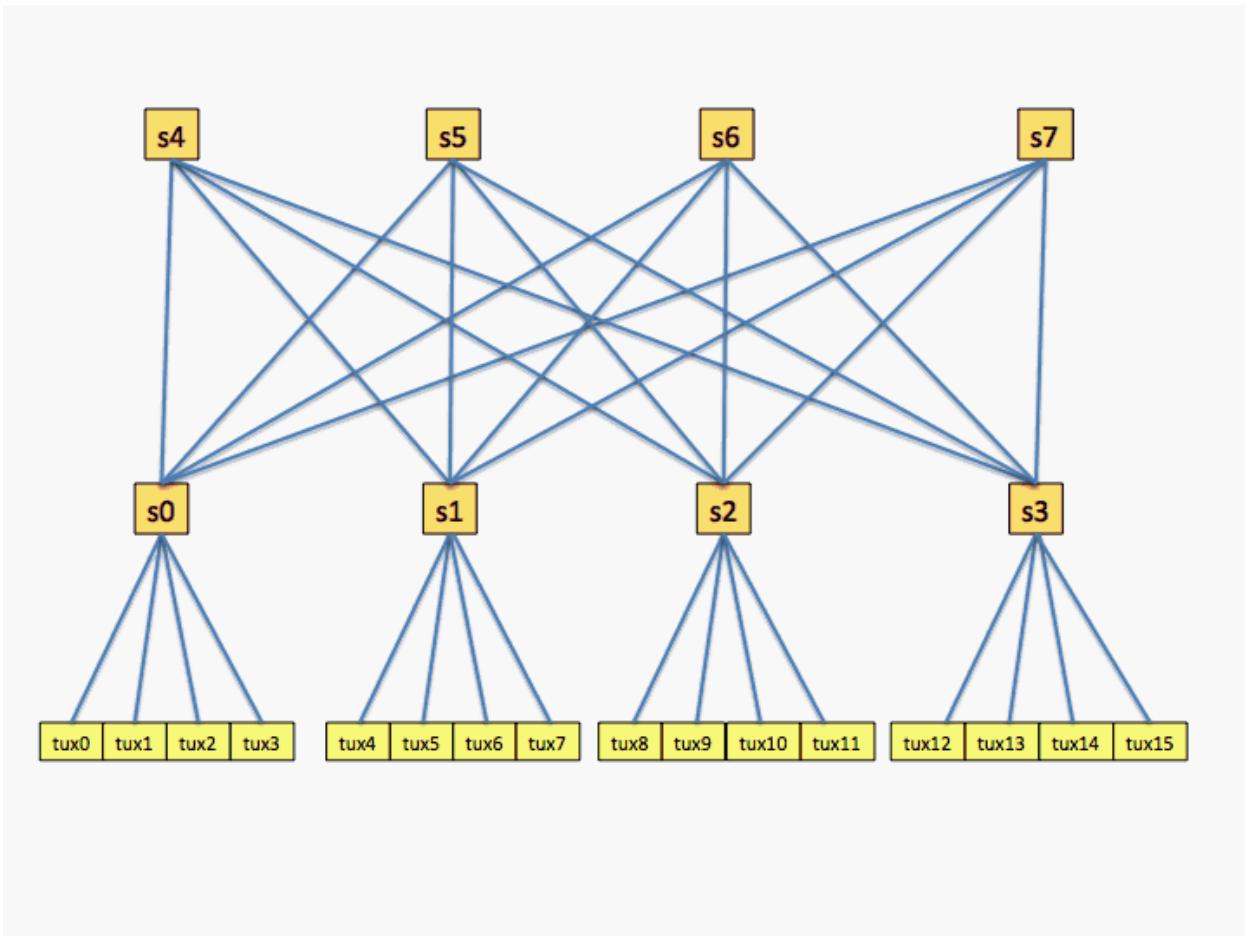


The next example is for a network with two levels and each switch has four connections.

```

# topology.conf
# Switch Configuration
SwitchName=s0 Nodes=tux[0-3] LinkSpeed=900
SwitchName=s1 Nodes=tux[4-7] LinkSpeed=900
SwitchName=s2 Nodes=tux[8-11] LinkSpeed=900
SwitchName=s3 Nodes=tux[12-15] LinkSpeed=1800
SwitchName=s4 Switches=s[0-3] LinkSpeed=1800
SwitchName=s5 Switches=s[0-3] LinkSpeed=1800
SwitchName=s6 Switches=s[0-3] LinkSpeed=1800
SwitchName=s7 Switches=s[0-3] LinkSpeed=1800

```



As a practical matter, listing every switch connection definitely results in a slower scheduling algorithm for Slurm to optimize job placement. The application performance may achieve little benefit from such (attempted) optimization. Listing the leaf switches with their nodes plus one top level switch should result in good performance for both applications and Slurm. The previous example might be configured as follows:

```

# topology.conf
# Switch Configuration
SwitchName=s0 Nodes=tux[0-3]
SwitchName=s1 Nodes=tux[4-7]
SwitchName=s2 Nodes=tux[8-11]
SwitchName=s3 Nodes=tux[12-15]
SwitchName=s4 Switches=s[0-3]

```

NOTE: Compute nodes on switches that lack a common parent switch can be used, but no job will span leaf switches without a common parent (unless the TopologyParam=TopoOptional option is used). For example, it is legal to remove the line "SwitchName=s4 Switches=s[0-3]" from the above topology.conf file. In that case, no job will span more than four compute nodes on any single leaf switch. This configuration can be useful if one wants to schedule multiple physical clusters as a single logical cluster under the control of a single slurmcld daemon.

For systems with a dragonfly network, configure Slurm with TopologyPlugin=topology/tree plus TopologyParam=dragonfly. If a single job can not be entirely placed within a single network leaf switch, the job will be spread across as many leaf switches as possible in order to optimize the job's network bandwidth.

NOTE: Slurm first identifies the network switches which provide the best fit for pending jobs and then selects the nodes with the lowest "weight" within those switches. If optimizing resource selection by node weight is more important than optimizing network topology then do NOT use the topology/tree plugin.

NOTE: The topology.conf file for an Infiniband switch can be automatically generated using the slurmibtopology tool found here:

<https://ftp.fysik.dtu.dk/Slurm/slurmibtopology.sh>

NOTE: The topology.conf file for an Omni-Path (OPA) switch can be automatically generated using the opa2slurm tool found here:

<https://gitlab.com/jtfrey/opa2slurm.>

User Options

For use with the topology/tree plugin, users can also specify the maximum number of leaf switches to be used for their job with the maximum time the job should wait for this optimized configuration. The syntax for this option is "--switches=count[@time]". The system administrator can limit the maximum time that any job can wait for this optimized configuration using the SchedulerParameters configuration parameter with the max_switch_wait option.

Environment Variables

if the topology/tree plugin is used, two environment variables will be set to describe that jobs network topology. Note that these environment variables will contain different data for the tasks launched on each node. Use of these environment variables is at the discretion of the user:

SLURM_TOPOLOGY_ADDR: The value will be set to the names network switches which may be involved in the job's communications from the system's top level switch down to the leaf switch and ending with node name.
A period is used to separate each hardware component name.

SLURM_TOPOLOGY_ADDR_PATTERN: This is set only if the system has the topology/tree plugin configured. The value will be set component types listed in SLURM_TOPOLOGY_ADDR. Each component will be identified as either "switch" or "node".
A period is used to separate each hardware component type

Topology and Application Layout Exercises

In this set of labs, you will learn how to manage network topology and application layout in Slurm

Exercise 1: Create a `topology.conf` file

1. As root, create the file `/etc/slurm/topology.conf` and put the following in it:

```
SwitchName=s0 Nodes=node[00-03]
SwitchName=s1 Nodes=node[04-07]
SwitchName=s2 Nodes=node[08-09]
SwitchName=s3 Switches=s[0-2]
```

2. Enable the Topology/Tree plugin by editing the `/etc/slurm/slurm.conf` file and adding the following:

```
TopologyPlugin=topology/tree
```

3. Restart the scheduler and slurmd's as root:

```
/lab_scripts/restart.sh
```

4. Verify the plugin has been read into the configuration by entering:
`scontrol show config | grep -i topologyplugin`

You should see:

```
TopologyPlugin = topology/tree
```

Exercise 2: Submit a job to a node on a switch

In this exercise, you will submit a job and have the job land on one of the switches, thereby making sure the job doesn't have to span switches which might degrade performance.

1. As the fred user, submit a job as follows:

```
sbatch --switches=1 --mem=1000 -N4 -n16 --wrap "sleep 300"
```

2. View on which nodes and switch the job landed:

```
scontrol show job <JOBID> | grep -i nodelist
```

Note: you should get the following output:

```
NodeList=node[00-03]
```

or

```
NodeList=node[04-07]
```

3. Perform another batch submission and see where it lands:

```
sbatch --switches=1 --mem=1000 -N4 -n16 --wrap "sleep 300"
```

4. View on which nodes and switch the job landed:

```
scontrol show job <JOBID> | grep -i nodelist
```

Note: you should get the following output:

```
NodeList=node[00-03]
```

or

```
NodeList=node[04-07]
```

5. Cancel all of **fred's** jobs:

```
scancel -u fred
```

Exercise 3: Using cpu-bind to Place Tasks on Allocated Resources

One of the methods for placing tasks on allocated resources provides for a patterned or custom distribution of those tasks. One can use the **-m** switch to determine task placement

1. Configure salloc to automatically launch in interactive mode with a prompt on the allocated node:

As root, edit /etc/slurm/slurm.conf and add the following:

```
LaunchParameters=enable_nss_slurm,use_interactive_step
```

And confirm the following is already configured:

```
SelectType=select/cons_tres
SelectTypeParameters=CR_Core_Memory
```

2. Restart the scheduler and slurmd's as root:

```
/lab_scripts/restart.sh
```

NOTE: if the nodes go down because of this, then execute the following:

```
scontrol update nodename=node[00-09] state=resume 2>/dev/null
```

3. As **fred**, generate an 8-core allocation request using the following syntax:

```
salloc -n8 -m block
```

You should end up in an allocation with a prompt on a compute node like this:

```
salloc: Granted job allocation 4
salloc: Waiting for resource configuration
salloc: Nodes node[00-01] are ready for job
```

4. Launch srun so that each task lands on a distinct core, and the distribution of tasks loads up the first node, then moves to the second node, etc:

```
srun -l -n8 --mem=1000 -mblock --cpu-bind=v uptime 1>/dev/null
```

You should see:

```

0: cpu-bind=MASK - node00, task 0 0 [126]: mask 0x1 set
0: cpu-bind=MASK - node00, task 1 1 [127]: mask 0x2 set
0: cpu-bind=MASK - node00, task 2 2 [128]: mask 0x4 set
0: cpu-bind=MASK - node00, task 3 3 [129]: mask 0x8 set
4: cpu-bind=MASK - node01, task 4 0 [127]: mask 0x1 set
4: cpu-bind=MASK - node01, task 5 1 [128]: mask 0x2 set
4: cpu-bind=MASK - node01, task 6 2 [129]: mask 0x4 set
4: cpu-bind=MASK - node01, task 7 3 [130]: mask 0x8 set

```

NOTE: The 1st node fills up first, then the 2nd node with tasks. (Your output may be sorted differently than mine, however.)

- Now submit a similar srun command while still in the allocation, but this time change the distribution types to cyclic to see the different task allocation distribution:

```
srun -l -n8 -m cyclic --cpu-bind=verbose uptime 1>/dev/null
```

You should now see:

```

0: cpu-bind=MASK - node00, task 0 0 [86]: mask 0x1 set
1: cpu-bind=MASK - node01, task 1 0 [87]: mask 0x1 set
0: cpu-bind=MASK - node00, task 2 1 [87]: mask 0x2 set
1: cpu-bind=MASK - node01, task 3 1 [88]: mask 0x2 set
0: cpu-bind=MASK - node00, task 4 2 [88]: mask 0x4 set
1: cpu-bind=MASK - node01, task 5 2 [89]: mask 0x4 set
0: cpu-bind=MASK - node00, task 6 3 [89]: mask 0x8 set
1: cpu-bind=MASK - node01, task 7 3 [90]: mask 0x8 set

```

NOTE: The back and forth between nodes allocation occurs because of cyclic the distribution method.

- Type **exit** to exit the allocation

Exercise 4: Using cpu-bind to Place Tasks on a specific Resource

--cpu-bind allows slurm, within the allocation, to launch the step on a specific resource, such as a CPU or Socket.

- Submit a 10-minute allocation request for 1 node and 4 tasks on node00:

```
salloc -N1 -n4 -t10 -wnode00
```

Note: This sets up an interactive session on node00.

- Submit an srun command, allocating the entire node, and launch a step that will launch on cpu 4 (Hex 0x8) in the allocation that will run stress-ng in the background:

```
srun -N1 --mem=1000 --exclusive --cpu-bind=verbose,mask_cpu:0x4 stress-ng --matrix 1 -t 1m&
```

You should now see:

```
cpu-bind=MASK - node00, task 1 1 [1044]: mask 0x4 set
cpu-bind=MASK - node00, task 2 2 [1045]: mask 0x4 set
cpu-bind=MASK - node00, task 3 3 [1046]: mask 0x4 set
stress-ng: info: [1046] dispatching hogs: 1 matrix
stress-ng: info: [1045] dispatching hogs: 1 matrix
stress-ng: info: [1044] dispatching hogs: 1 matrix
stress-ng: info: [1043] dispatching hogs: 1 matrix
```

NOTE: The hex id 0x8 is the CPU number 3 (in series 0,1,2,3) in this layout. Or, the 4th Core on the CPU in that socket.

3. Press **Enter** to get your prompt back
4. Launch top and expand the CPUs by pressing the number 1

It should look like this:

```
top - 09:31:00 up 56 min, 0 users, load average: 2.19, 1.71, 1.36
Tasks: 23 total, 5 running, 18 sleeping, 0 stopped, 0 zombie
%Cpu0 : 6.2 us, 0.0 sy, 0.0 ni, 93.8 id, 0.0 wa, 0.0 hi, 0.0 si, 0.0 st
%Cpu1 : 0.0 us, 6.2 sy, 0.0 ni, 93.8 id, 0.0 wa, 0.0 hi, 0.0 si, 0.0 st
%Cpu2 : 0.0 us, 6.2 sy, 0.0 ni, 93.8 id, 0.0 wa, 0.0 hi, 0.0 si, 0.0 st
%Cpu3 : 100.0 us, 0.0 sy, 0.0 ni, 0.0 id, 0.0 wa, 0.0 hi, 0.0 si, 0.0 st
MiB Mem : 15967.8 total, 8339.1 free, 5164.4 used, 2464.3 buff/cache
MiB Swap: 2048.0 total, 2048.0 free, 0.0 used. 10300.8 avail Mem
...
```

Note: The stress-ng matrix stressor is running on Cpu 3, or 0x8 hex, the 4th cpu of 4.

Cleanup

1. Reset the Docker Slurm cluster environment back to its original, pristine state by exiting the login Docker container back to the Ubuntu AWS prompt. And **as ubuntu**:
cd ~/docker-scale-out
make clean && make

Cgroups Discussion

For a comprehensive description of Linux Control Groups (cgroups) see the [cgroups documentation at kernel.org](#). Detailed knowledge of cgroups is not required to use cgroups in Slurm, but a basic understanding of the following features of cgroups is helpful:

Cgroup - A container for a set of processes subject to common controls or monitoring, implemented as a directory and a set of files (state objects) in the cgroup virtual filesystem.

Subsystem - A module, typically a resource controller, that applies a set of parameters to the cgroups in a hierarchy.

Hierarchy - A set of cgroups organized in a tree structure, with one or more associated subsystems.

State Objects - Seudofiles that represent the state of a cgroup or apply controls to a cgroup:

- tasks - Identifies the processes (PIDs) in the cgroup.
- Additional state objects specific to each subsystem.

Use of Cgroups in Slurm

Slurm provides cgroup versions of a number of plugins.

- **proctrack** (process tracking)
- **task** (task management)
- **jobacct_gather** (job accounting statistics)

The cgroup plugins can provide a number of benefits over the other more standard plugins, as described below.

Slurm also uses cgroups for resource specialization.

Slurm Cgroups Configuration Overview

There are several sets of configuration options for Slurm cgroups:

- `slurm.conf` provides options to enable the cgroup plugins. Each plugin may be enabled or disabled independently of the others.
- `cgroup.conf` provides general options that are common to all cgroup plugins, plus additional options that apply only to specific plugins.
- System-level resource specialization is enabled using node configuration parameters.

Currently Available Cgroup Plugins

PROCTRACK/CGROUP PLUGIN

The proctrack/cgroup plugin is an alternative to other proctrack plugins such as proctrack/linux for process tracking and suspend and resume capability. proctrack/cgroup uses the cgroups “freezer” subsystem which is more reliable for tracking and control than proctrack/linux.

To enable this plugin, configure the following option in `slurm.conf`:

```
ProctrackType=proctrack/cgroup
```

There are no specific options for this plugin in `cgroup.conf`, but the general options apply. See the `cgroup.conf` man page for details.

TASK/CGROUP PLUGIN

The task/cgroup plugin is an alternative to other task plugins such as the task/affinity plugin for task management. task/cgroup provides the following features:

- The ability to confine jobs and steps to their allocated cpuset.
- The ability to bind tasks to sockets, cores and threads within their step's allocated cpuset on a node.
- Supports block and cyclic distribution of allocated cpus to tasks for binding.
- The ability to confine jobs and steps to specific memory resources.
- The ability to confine jobs to their allocated set of generic resources (gres devices).

The task/cgroup plugin uses the cpuset, memory and devices subsystems.

To enable this plugin, configure the following option in `slurm.conf`:

```
TaskPlugin=task/cgroup
```

There are many specific options for this plugin in `cgroup.conf`. The general options also apply. See the `cgroup.conf` man page for details.

JOBACCT_GATHER/CGROUP PLUGIN

The jobacct_gather/cgroup plugin is an alternative to the jobacct_gather/linux plugin for the collection of accounting statistics for jobs, steps and tasks. jobacct_gather/cgroup uses the cpuacct, memory and blkio subsystems.

NOTE: The cpu and memory statistics collected by this plugin do not represent the same resources as the cpu and memory statistics collected by the jobacct_gather/linux plugin (sourced from /proc/stat). While originally thought to be faster, in practice it has been proven to be slower than the jobacct_gather/linux plugin.

To enable this plugin, configure the following option in `slurm.conf`:

```
JobacctGatherType=jobacct_gather/cgroup
```

There are no specific options for this plugin in `cgroup.conf`, but the general options apply. See the `cgroup.conf` man page for details.

Use of Cgroups for Resource Specialization

Resource Specialization may be used to reserve a subset of cores on each compute node for exclusive use. Slurm compute node daemons (`slurmd`, `slurmstepd`). It may also be used to apply a real memory limit to the

daemons. The daemons are confined to the reserved cores using a special system cgroup in the cpuset hierarchy. The memory limit is enforced using a system cgroup in the memory hierarchy. System-level resource specialization is enabled with special node configuration parameters in `slurm.conf` and core specialization in `core_spec.html`.

NOTE: In Slurm V23.02, slurmstepd cgroups are constrained by total configured memory from `slurm.conf` (`NodeName=> RealMemory=#`) instead of total physical memory.

Organization of Slurm Cgroups

Slurm cgroups are organized as follows. A base directory (mount point) is created at `/sys/fs/cgroup`, or as configured by the `CgroupMountpoint` option in `cgroup.conf`. All cgroup hierarchies are created below this base directory. A separate hierarchy is created for each cgroup subsystem in use. The name of the root cgroup in each hierarchy is the subsystem name. A cgroup named `slurm` is created below the root cgroup in each hierarchy. Below each `slurm` cgroup, cgroups for Slurm users, jobs, steps and tasks are created dynamically as needed. The names of these cgroups consist of a prefix identifying the Slurm entity (user, job, step or task), followed by the relevant numeric id. The following example shows the path of the task cgroup in the cpuset hierarchy for taskid#2 of stepid#0 of jobid#123 for userid#100, using the default base directory (`/sys/fs/cgroup`):

```
/cgroup/cpuset/slurm/uid_100/job_123/step_0/task_2
```

If resource specialization is configured, a special system cgroup is created below the `slurm` cgroup in the cpuset and memory hierarchies:

```
/sys/fs/cgroup/cpuset/slurm/system  
/sys/fs/cgroup/memory/slurm/system
```

NOTE: All these structures apply to a specific compute node. Jobs that use more than one node will have a cgroup structure on each node.

In Slurm V23.02, all cgroup directories created by Slurm are now owned by root. This was the behavior in `cgroup/v2` but not in `cgroup/v1` where by default the step directories ownership were set to the user and group of the job.

CGROUP V2 Plugin

Slurm v22.05 provides support for systems with Control Group v2. The cgroup/v2 plugin is an internal Slurm API used by other plugins, like proctrack/cgroup, task/cgroup and jobacctgather/cgroup. This document gives an overview of how it is designed, with the aim of getting a better idea of what is happening on the system when Slurm constrains resources with this plugin.

Following cgroup v2 rules

Kernel's Control Group v2 has two particularities that affect how Slurm needs to structure its internal cgroup tree.

TOP-DOWN CONSTRAINT

Resources are distributed top-down to the tree, so a controller is only available on a cgroup directory if the parent node has it listed in its `cgroup.controllers` file and added to its `cgroup.subtree_control`. Also, a controller activated in the subtree cannot be disabled if one or more children has them enabled. For Slurm this implies we need to do this kind of management over our hierarchy modifying this file and enabling the required controllers for the child.

NO INTERNAL PROCESS CONSTRAINT

Except for the root cgroup, parent cgroups (really called domain cgroups) can only enable controllers for their children if they do not have any process at their own level. This means we can create a subtree inside a cgroup directory, but before writing to `cgroup.subtree_control`, all the pids listed in the parent's `cgroup.procs` must be migrated to the child. This requires that all processes must live on the leaves of the tree and so it will not be possible to have pids in non-leaf directories.

Following systemd rules

Systemd is currently the most widely used init mechanism. For this reason Slurm needs to find a way to coexist with the rules of systemd. The designers of systemd have conceived a new rule called the "single-writer" rule, which implies that every cgroup has one single owner and nobody else should write to it. Read more about this in [systemd.io Cgroup Delegation Documentation](#). In practice this means that the systemd daemon, started when the kernel boots and which takes pid 1, will consider itself the absolute owner and single writer of the entire cgroup tree. This means that systemd expects that no other process should be modifying any cgroup directly, nor should another process be creating directories or moving pids around, without systemd being aware of it.

There's one method that allows Slurm to work without issues, which is to start Slurm daemons in a systemd Unit with the special systemd option `Delegate=yes`. Starting slurmd within a systemd Unit, will give Slurm a "delegated" cgroup subtree in the filesystem where it will be able to create directories, move pids, and manage its own hierarchy. In practice, what happens is that systemd registers a new Unit in its internal database and relates the cgroup directory to it. Then for any future "intrusive" actions of the cgroup tree, systemd will effectively ignore the "delegated" directories.

This is similar to what happened in cgroup v1, since this is not a kernel rule, but a systemd rule. But this fact combined with the new cgroup v2 rules, forces Slurm to choose a design which coexists with both.

THE REAL PROBLEM: SYSTEMD + RESTARTING SLURMD

When designing the cgroup/v2 plugin for Slurm, the initial idea was to let slurmd setup the required hierarchy in its own cgroup directory. There it would place jobs and steps and move newer forked slurmstepds into the corresponding directories.

This worked fine, until we needed to restart slurmd. Since the hierarchy was already created, the slurmd restart just terminated the slurmd process and then started a new one, but then it would try to put the new process directly in the root of the specific group tree. Since this directory was now a domain controller and not a leaf anymore, systemd would fail to start the daemon.

Lacking any mechanism in systemd to tackle this situation, this left us with no other choice but to separate slurmd and forked slurmstepds into separate subtree directories. Because of the design rule of systemd about being the single-writer on the tree, it was not possible to just do a "mkdir" from slurmd or the slurmstepd itself and then move the stepd process into a new and separate directory, that would mean this directory was not controlled by systemd and would cause problems.

The only way that a "mkdir" could work was if it was done inside a "delegated" cgroup subtree, so we needed to find a way to find a Unit with "Delegate=yes", different from the slurmd one, which would guarantee our independence. So, we really needed to start a new unit for user jobs.

Actually, in systemd there are two types of Units that can get the "Delegate=yes" parameter and that are directly related to a cgroup directory, one is a "Service" and the other is a "Scope". We are interested the "scope":

A Systemd Scope: systemd takes a pid as an argument, creates a cgroup directory and then adds the provided pid to the directory. The scope will remain until this pid is gone.

Because we wanted to keep the systemd scope for any pid, we needed to call a specific function named "abandonScope" in systemd's dbus interface. Abandoning a scope makes it so that the scope will continue to be alive while there's a living pid in its cgroup tree, not just the initial pid.

It is worth noting that a discussion with main systemd developers raised the RemainAfterExit systemd parameter. This parameter is intended to keep the unit alive even if all the processes on it are gone. This option is only valid for "Services" and not for "Scopes". This would be a very interesting option to have if it was included also for Scopes. They stated that its functionality could be extended to not only keep the unit, but to also keep the cgroup directories until the unit was manually terminated. Currently, the unit remains alive but the cgroup is cleaned anyway.

With all this background, we're ready to show which solution was used to make Slurm get away from the problem of the slurmd restart.

- Create a new Scope on slurmd startup for hosting new slurmstepd processes. It does one single call at the first slurmd startup. Slurmd prepares a scope for future slurmstepd pids, and the stepd itself moves itself there when starting. This comes without any performance issue, and conceptually is just like a slower "mkdir" + informing systemd from slurmd only at the first startup. Moving processes from one delegated unit to another delegated unit was approved by systemd developers. The only downside is that the scope needs processes inside or it will terminate and cleanup the cgroup, so slurmd needed to create a "sleep" infinity process, which we encoded into the "slurmstepd infinity" process, which will live forever in the scope. In the future, if the RemainAfterExit parameter is extended to scopes and allows the cgroup tree to not be destroyed, the need for this infinity process would be eliminated.

Finally we ended up with separating *slurmd* from *slurmstepds*, using a scope with "Delegate=yes" option.

CONSEQUENCES OF NOT FOLLOWING SYSTEMD RULES

There is a known issue where `systemd` can decide to cleanup the cgroup hierarchy with the intention of making it match with its internal database. For example, if there are no units in the system with "`Delegate=yes`", it will go through the tree and possibly deactivate all the controllers which it thinks are not in use. In our testing we stopped all our units with "`Delegate=yes`", issued a "`systemctl reload`" or a "`systemctl reset-failed`" and witnessed how the cpuset controller disappeared from our "manually" created directories deep in the cgroup tree. There are other situations, and the fact that `systemd` developers and documentation claim that they are the unique single-writer to the tree, made SchedMD decide to be on the safe side and have Slurm coexist with `systemd`.

It is worth noting that we added `IgnoreSystemd` and `IgnoreSystemdOnFailure` as `cgroup.conf` parameters which will avoid any contact with `systemd`, and will just use a regular "`mkdir`" to create the same directory structures. These parameters are for development and testing purposes only.

WHAT HAPPENS WITH LINUX DISTROS WITHOUT SYSTEMD?

Slurm does not support them, but they can still work. The only requirements are to have `libdbus`, `ebpf` and `systemd` packages installed in the system to compile slurm. Then you can set the `IgnoreSystemd` parameter in `cgroup.conf` to manually create the `/sys/fs/cgroup/system.slice/` directory. With these requirements met, Slurm should work normally.

cgroup/v2 overview

We will explain briefly this plugin's workflow.

SLURMD STARTUP

Fresh system: `slurmd` is started. Some plugins (`proctrack`, `jobacctgather` or `task`) which use cgroup, call `init()` function of cgroup/v2 plugin. What happens immediately is that `slurmd` does a call to `dbus` using `libdbus`, and creates a new `systemd` "Scope". The scope name is predefined and set depending on an internal constant `SYSTEM_CGSCOPE` under `SYSTEM_CGSlice`. It basically ends up with the name "`slurmstepd.scope`" or "`nodename_slurmstepd.scope`" depending on whether Slurm is compiled with `--enable-multiple-slurmd` (prefixes node name) or not. The cgroup directory associated with this scope will be fixed as:

`/sys/fs/cgroup/system.slice/slurmstepd.scope` or
`/sys/fs/cgroup/system.slice/nodename_slurmstepd.scope`.

The scope is also "abandoned" calling the `dbus` method of "`abandonScope`" with the purpose explained previously on this page.

Since the call to `dbus` "`startTransientUnit`" requires a pid as a parameter, `slurmd` needs to fork a "`slurmstepd infinity`" and use this parameter as the argument.

The call to `dbus` is asynchronous, so `slurmd` delivers the message to the `Dbus` bus and then starts an active wait, waiting for the scope directory to show up. If the directory doesn't show up within a hardcoded timeout, it fails. Otherwise it continues and `slurmd` then creates a directory for new `slurmstepds` and for the infinity pid in the recently created scope directory, called "system". It moves the infinity process into there and then enables all the required controllers in the new cgroup directories.

As this is a regular `systemd` Unit, the scope will show up in "`systemctl list-unit-files`" and other `systemd` commands, for example:

```

$ systemctl cat gamba1_slurmstepd.scope
# /run/systemd/transient/gamba1_slurmstepd.scope
# This is a transient unit file, created programmatically via the systemd API. Do not edit.

[Scope]
Delegate=yes
TasksMax=infinity

$ systemctl list-unit-files gamba1_slurmstepd.scope
UNIT FILE STATE VENDOR PRESET
gamba1_slurmstepd.scope transient - 

1 unit files listed.

$ systemctl status gamba1_slurmstepd.scope
● gamba1_slurmstepd.scope
    Loaded: loaded (/run/systemd/transient/gamba1_slurmstepd.scope; transient)
              ...
    Transient: yes
    Active: active (abandoned) since Wed 2022-04-06 14:17:46 CEST; 2h 47min ago
      Tasks: 1
     Memory: 1.6M
        CPU: 258ms
      CGROUP: /system.slice/gamba1_slurmstepd.scop
                  └─system
                      └─113094 /home/lipi/slurm/master/inst/sbin/slurmstepd infinity

april 06 14:17:46 llit systemd[1]: Started gamba1_slurmstepd.scope.

```

Another action of slurmd init will be to detect which controllers are available in the system (in `/sys/fs/cgroup`), and recursively enable the needed ones until reaching its level. It will enable them for the recently created slurmstepd scope.

```

$ cat /sys/fs/cgroup/system.slice/gamba1_slurmstepd.scope/cgroup.controllers
cpuset cpu io memory pids

$ cat /sys/fs/cgroup/system.slice/gamba1_slurmstepd.scope/cgroup.subtree_control
cpuset cpu memory

```

If resource specialization is enabled, slurmd will set its memory and/or cpu constrains at its own level too.

SLURMD RESTART

Slurmd restarts as usual. When restarted, it will detect if the "scope" directory already exists, and will do nothing if it does. Otherwise it will try to setup the scope again.

SLURMSTEPD START

When a new step needs to be created, whether part of a new job or as part of an existing job, slurmd will fork the slurmstepd process in its own cgroup directory. Instantly slurmstepd will start initializing and (if cgroup plugins are enabled) it will infer the scope directory and will move itself into the "waiting" area, which is the `/sys/fs/cgroup/system.slice/slurmstepd_nodename.scope/system` directory. Immediately it

will initialize the job and step cgroup directories and will move itself into them, setting the subtree_controllers as required.

TERMINATION AND CLEANUP

When a job ends, slurmstepd will take care of removing all the created directories. The slurmstepd.scope directory will never be removed or stopped by Slurm, and the "slurmstepd infinity" process will never be killed by Slurm.

When slurmd ends (since on supported systems it has been started by systemd) its cgroup will just be cleaned up by systemd.

HIERARCHY OVERVIEW

Hierarchy will take this form:

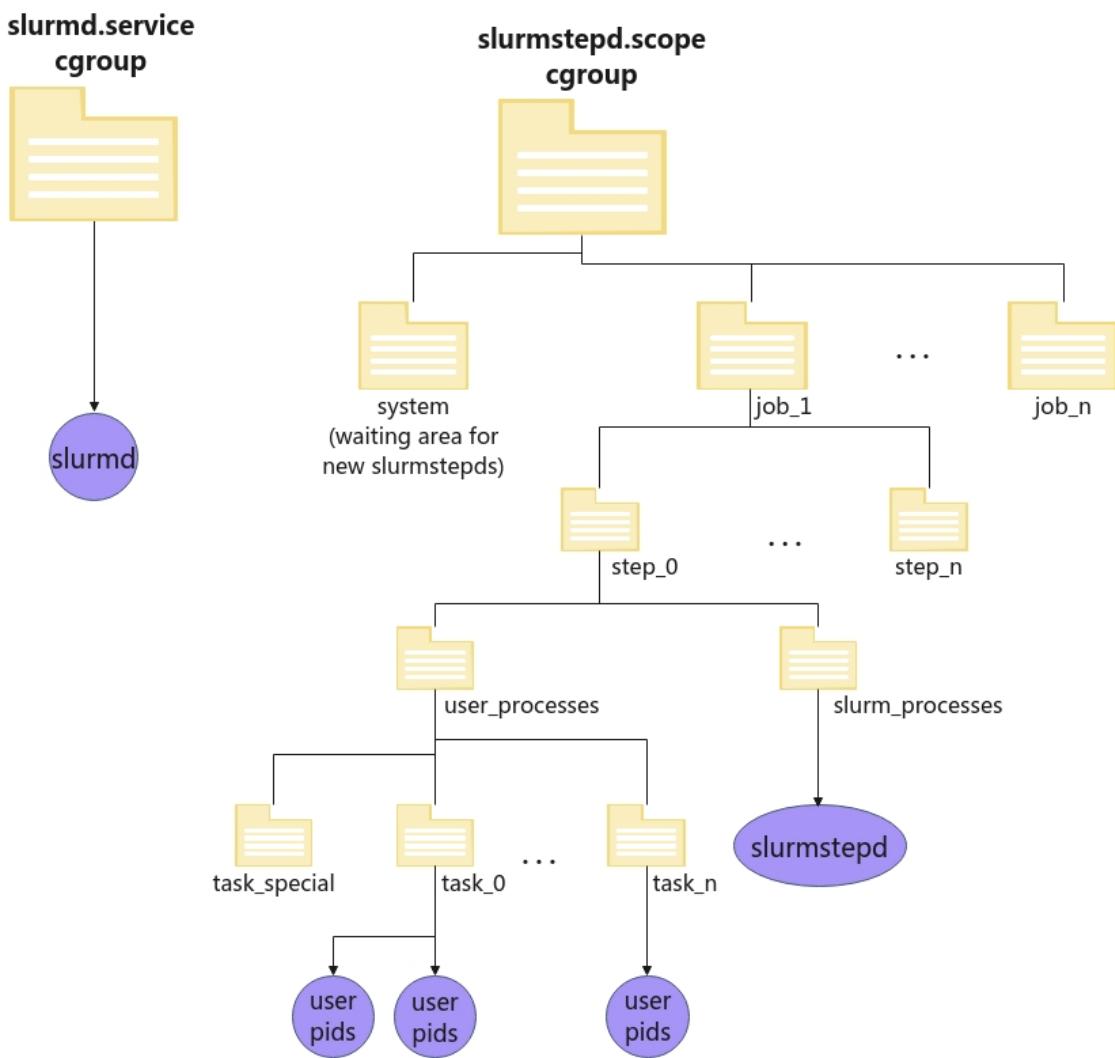


Figure 1. Slurm cgroup v2 hierarchy.

On the left side we have the slurmd service, started with systemd and living alone in its own delegated cgroup.

On the right side we see the slurmstepd scope, a directory in the cgroup tree also delegated where all slurmstepd and user jobs will reside. The slurmstepd is migrated initially in the waiting area for new stepds, system directory, and immediately, when it initializes the job hierarchy, it will move itself into the corresponding `job_x/step_y/slurm_processes` directory.

User processes will be spawned by slurmstepd and moved into the appropriate task directory.

At this point it should be possible to check which processes are running in a slurmstepd scope by issuing this command:

```
$ systemctl status slurmstepd.scope
● slurmstepd.scope
  Loaded: loaded (/run/systemd/transient/slurmstepd.scope; transient)
  Transient: yes
    Active: active (abandoned) since Wed 2022-04-06 14:17:46 CEST; 2min 47s ago
      Tasks: 24
     Memory: 18.7M
        CPU: 141ms
      CGroup: /system.slice/slurmstepd.scope
              └─job_3385
                  ├─step_0
                  │   ├─slurm
                  │   │   └─113630 slurmstepd: [3385.0]
                  │   └─user
                  │       ├─task_0
                  │           └─113635 /usr/bin/sleep 123
                  ├─step_extern
                  │   ├─slurm
                  │   │   └─113565 slurmstepd: [3385.extern]
                  │   └─user
                  │       ├─task_0
                  │           └─113569 sleep 100000000
                  └─step_interactive
                      ├─slurm
                      │   └─113584 slurmstepd: [3385.interactive]
                      └─user
                          ├─task_0
                          │   ├─113590 /bin/bash
                          │   ├─113620 srun sleep 123
                          │   └─113623 srun sleep 123
              └─system
                  └─113094 /home/lipi/slurm/master/inst/sbin/slurmstepd infinity
```

Working at the task level

There is a directory called `task_special` in the user job hierarchy. The `jobacctgather/cgroup` and `task/cgroup` plugins respectively get statistics and constrain resources at the task level. Other plugins like `proctrack/cgroup` just work at the step level. To unify the hierarchy and make it work for all different plugins, when a plugin asks to add a pid to a step but not to a task, the pid will be put into a special directory called `task_special`. If another plugin adds this pid to a task, it will be migrated from there. Normally this happens with the `proctrack` plugin when a call is done to add a pid to a step with `proctrack_g_add_pid`.

The eBPF based devices controller

In Control Group v2, the devices controller interfaces has been removed. Instead of controlling it through files, now it is required to create a bpf program of type `BPF_PROG_TYPE_CGROUP_DEVICE` and attach it to the desired

cgroup. This program is created by slurmtep dynamically and inserted into the kernel with a bpf syscall, and describes which devices are allowed or denied for the job, step and task.

The only devices that are managed are the ones described in the gres.conf file.

The insertion and removal of such programs will be logged in the system log:

```
apr 06 17:20:14 node1 audit: BPF prog-id=564 op=LOAD
apr 06 17:20:14 node1 audit: BPF prog-id=565 op=LOAD
apr 06 17:20:14 node1 audit: BPF prog-id=566 op=LOAD
apr 06 17:20:14 node1 audit: BPF prog-id=567 op=LOAD
apr 06 17:20:14 node1 audit: BPF prog-id=564 op=UNLOAD
apr 06 17:20:14 node1 audit: BPF prog-id=567 op=UNLOAD
apr 06 17:20:14 node1 audit: BPF prog-id=566 op=UNLOAD
apr 06 17:20:14 node1 audit: BPF prog-id=565 op=UNLOAD
```

Running different nodes with different cgroup versions

The cgroup version to be used is entirely dependent on the node. Because of this, it is possible to run the same job on different nodes with different cgroup plugins. The configuration is done per node in cgroup.conf.

What can not be done is to swap the version of cgroup plugin in cgroup.conf without rebooting and configuring the node. Since we do not support "hybrid" systems with mixed controller versions, a node must be booted with one specific cgroup version.

Configuration

In terms of configuration, setup does not differ much from the previous cgroup/v1 plugin, but the following considerations must be taken into account when configuring the cgroup plugin in cgroup.conf:

CGROUP PLUGIN

This option allows the sysadmin to specify which cgroup version will be run on the node. It is recommended to use autodetect and forget about it, but this can be forced to the plugin version too.

CgroupPlugin=[autodetect|cgroup/v1|cgroup/v2]

DEVELOPER OPTIONS

- IgnoreSystemd=[yes|no]: This option is used to avoid any call to dbus for contacting systemd. Instead of requesting the creation of a new scope when slurmd starts up, it will only use "mkdir" to prepare the cgroup directories for the slurmstepds. Use of this option in production systems with systemd is not supported for the reasons mentioned above. This option can be useful for systems without systemd though.
- IgnoreSystemdOnFailure=[yes|no]: This option will fallback to manual mode for creating the cgroup directories without creating a systemd "scope". This is only if a call to dbus returned an error, as it would be with IgnoreSystemd.
- CgroupAutomount=[yes|no]: This option is only used when IgnoreSystemd is set. If both are set slurmd will check all the available controllers in /sys/fs/cgroup and will enable them recursively until it reaches the slurmd level. This will imply that the manually created slurmstepd directories will also have these controllers set.

- CgroupMountPoint=/path/to/mount/point: In most cases with cgroup v2, this parameter should not be used because /sys/fs/cgroup will be the only cgroup directory.

IGNORED PARAMETERS

Since Cgroup v2 doesn't provide the Kmem* or swappiness interfaces anymore in the memory controller, the following parameters in cgroup.conf will be ignored:

```
AllowedKmemSpace=
MemorySwappiness=
MaxKmemPercent=
MinKmemSpace=
```

NOTE: The following have been deprecated in Slurm V23.02:

- AllowedKmemSpace
- ConstrainKmemSpace
- MaxKmemPercent
- MinKmemSpace

Cgroup Exercises

Exercise 1: Restricting memory Cgroup Example

The learning environment is hosted in a virtual machine using Linux Docker containers to present the controllers and compute nodes. Docker supports cgroup passthrough, so it is possible to demonstrate the constraints provided with cgroups.

One of the more common uses of using cgroup configuration is to ensure that the RAM allocated to a Slurm job does not grow to consume more than was granted in the allocation.

With Slurm compiled with cgroup support, **srun** can be the "Police" with regards to overconsuming memory resources. The slurmd can inform the Linux cgroup about the memory requested for the job, and let the Linux cgroup oom handle what happens when it goes over.

Edit the `slurm.conf`:

1. Edit `/etc/slurm/slurm.conf` and confirm that the cgroup plugins are configured:

```
TaskPlugin=task/affinity,task/cgroup
```

and

```
ProctrackType=proctrack/cgroup #The default is proctrack/pgid Command it out
```

and

```
JobAcctGatherType=jobacct_gather/cgroup
```

2. Since you made a change to a plugin in the config file, restart the controller (**as root user**):

```
/lab_scripts/restart.sh
```

3. Edit `/etc/slurm/cgroup.conf` and confirm that the following are in it:

```
CgroupAutomount=yes
ConstrainCores=yes
ConstrainDevices=no
ConstrainRAMSpace=yes
ConstrainSwapSpace=yes
```

NOTE: In this training environment, the `/etc/slurm` directory is mounted across all the compute nodes and controller nodes, so there is no need to copy the files across the cluster.

4. In order to test slurm cgroup capabilities, create the following file, call it `eatmem.c`. Compile it **as fred** (call the compiled binary `eatmem`):

```

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <stdint.h>
#include <inttypes.h>
#include <unistd.h>
#include <sys/types.h>

int main(int argc, char *argv[])
{
    uint64_t tot = 0;
    int grow_by = 10;
    setbuf(stdout, NULL);
    argv[0] = "blah";
    while (1) {
        int i = 0;
        char *m;
        if (!(m = calloc(grow_by * 1024 * 1024, 1)))
            printf("Failed to allocate memory");

        for (; i < grow_by *1024 *1024; i++)
            m[i] = 'a';

        tot += grow_by;
        printf("pid: %d, Tot mem=%" PRIu64 "mb\n", getpid() , tot);
        sleep(1);
    }
    return 0;
}

```

NOTE: This program will “Eat” memory in chunks until exhausted

5. Compile it:

```
gcc eatmem.c -o eatmem
```

6. As **fred**, submit this job requesting 8000M of memory:

```
sbatch -N1 --mem=1000 --wrap="srun eatmem"
```

7. In another terminal, you can watch the node’s memory grow to excess with free, top, or sacct:

```
watch sstat 1 -o jobid,maxvmsize,maxrss,maxpages,avevmsize
```

Should show:

JobID	MaxVMSize	MaxRSS	MaxPages	AveVMSize
1.0	184484K	184484K	0	184484K

Once it hits 1GB, it will OOMkill:

JobID	MaxVMSize	MaxRSS	MaxPages	AveVMSize
Sstat: error: No steps running for job 1				

NOTE: It may take a minute or two to fill up the memory

- Look at the job output:

```
cat slurm-1.out
```

You will eventually see this:

```
...
pid: 4800, Tot mem=960mb
pid: 4800, Tot mem=970mb
pid: 4800, Tot mem=980mb
slurmstepd-node00: error: Detected 1 oom-kill event(s) in StepId=1.0. Some of
your processes may have been killed by the cgroup out-of-memory handler.
srun: error: node00: task 0: Out Of Memory
```

Cleanup

- Reset the Docker Slurm cluster environment back to its original, pristine state by exiting the login Docker container back to the Ubuntu AWS prompt. And **as ubuntu**:

```
cd ~/docker-scale-out
make clean && make
```

Exercise 2: Restricting memory: Another way Example

Cgroup is the preferred method for limiting jobs so applications don't consume more memory than they are supposed to.

Another method can be accomplished by making certain policy changes in the controller. These policies are:

- JobAcctGatherParams=OverMemoryKill

From the man page:

OverMemoryKill

Kill processes that are being detected to use more memory than requested by steps every time accounting information is gathered by the JobAcctGather plugin. This parameter should be used with caution because a job exceeding its memory allocation may affect other processes and/or machine health.

NOTE: If available, it is recommended to limit memory by enabling task/cgroup as a TaskPlugin and making use of ConstrainRAMSpace=yes in the cgroup.conf instead of using this JobAcctGather mechanism for memory enforcement. With OverMemoryKill, memory limit is applied against each process individually and is not applied to the step as a whole as it is with ConstrainRAMSpace=yes. Using JobAcctGather is polling based and there is a delay before a job is killed, which could lead to system Out of Memory events.

Edit the slurm.conf:

1. As root user, add the following to the /etc/slurm/slurm.conf:

```
JobAcctGatherParams=OverMemoryKill
```

2. Comment out the following by putting #'s at the beginning of the lines:

```
#PrologFlags=X11  
#JobAcctGatherType=jobacct_gather/cgroup  
#ProctrackType=proctrack/cgroup  
#TaskPlugin=task/affinity,task/cgroup
```

3. And change the following from:

```
PrologFlags=X11
```

to:

```
PrologFlags=Contain
```

4. As root user, restart the controller:

```
/lab_scripts/restart.sh
```

5. As fred, recreate the eatmem program from exercise 1 (since you blew it away on cluster refresh)

6. Using the eatmem program, submit a job as fred:

```
sbatch -N1 --mem=1000 --wrap="srun eatmem"
```

NOTE: This may hang the VM. Don't panic, give it a few minutes and it should kill the process and return control. This is not nearly as clean as using cgroups to invoke the OOM-killer.

7. Look at the job output.

You will see this:

```
slurmstslurmstepd-node00: error: StepId=1.0 exceeded memory limit (1049071616 > 1048576000), being killed  
slurmstslurmstepd-node00: error: Exceeded job memory limit  
srun: Job step aborted: Waiting up to 32 seconds for job step to finish.  
slurmstslurmstepd-node00: error: *** STEP 1.0 ON node00 CANCELLED AT 2023-09-29T11:00:49 ***  
srun: error: node00: task 0: Killed
```

Like I said, it's not as elegant as using the cgroup management, but it will do the job

Cleanup

1. Reset the Docker Slurm cluster environment back to its original, pristine state by exiting the login Docker container back to the Ubuntu AWS prompt. And as ubuntu:

```
cd ~/docker-scale-out  
make clean && make
```

Quality of Service Discussion

One can specify a Quality of Service (QOS) for each job submitted to Slurm. The quality of service associated with a job will affect the job in three ways:

- Job Scheduling Priority
- Job Preemption
- Job Limits
- Partition QOS
- Other QOS Options

The QOS's are defined in the Slurm database using the `sacctmgr` utility.

Jobs request a QOS using the "`--qos=`" option to the `sbatch`, `salloc`, and `srun` commands.

Job Scheduling Priority

Job scheduling priority is made up of a number of factors as described in the priority/multifactor plugin. One of the factors is the QOS priority. Each QOS is defined in the Slurm database and includes an associated priority. Jobs that request and are permitted a QOS will incorporate the priority associated with that QOS in the job's multi-factor priority calculation.

To enable the QOS priority component of the multi-factor priority calculation, the "PriorityWeightQOS" configuration parameter must be defined in the `slurm.conf` file and assigned an integer value greater than zero.

A job's QOS only affects its scheduling priority when the multi-factor plugin is loaded.

Job Preemption Introduction

Slurm offers two ways for a queued job to preempt a running job, free-up the running job's resources and allocate them to the queued job. See the Preemption description for details.

The preemption method is determined by the "PreemptType" configuration parameter defined in `slurm.conf`. When the "PreemptType" is set to "preempt/qos", a queued job's QOS will be used to determine whether it can preempt a running job.

The QOS can be assigned (using `sacctmgr`) a list of other QOS's that it can preempt. When there is a queued job with a QOS that is allowed to preempt a running job of another QOS, the Slurm scheduler will preempt the running job.

Job Limits

Each QOS is assigned a set of limits which will be applied to the job. The limits mirror the limits imposed by the user/account/cluster/partition association defined in the Slurm database and described in the Resource Limits section. When limits for a QOS have been defined, they will take precedence over the association's limits.

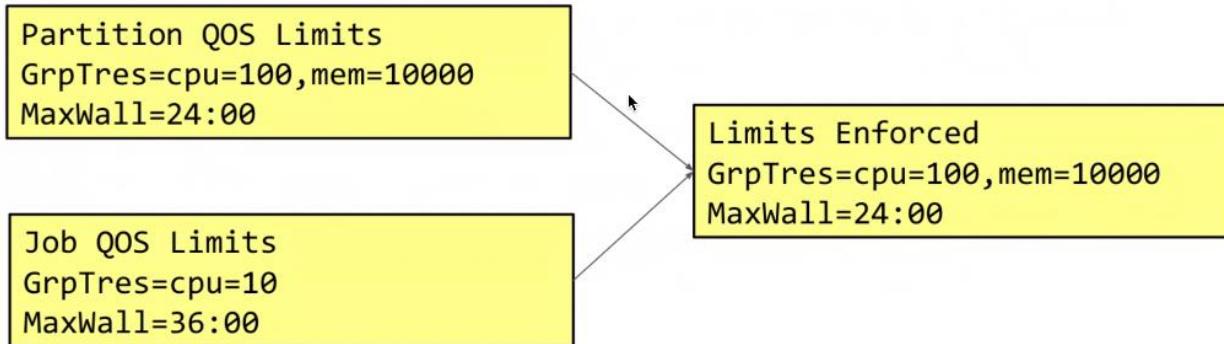
Partition QOS

A QOS can be attached to a partition. This means a partition will have all the same limits as a QOS. This also gives the ability of a true 'floating' partition, meaning if you assign all the nodes to a partition and then in the Partition's QOS limit the number of GrpCPUs or GrpNodes the partition will have access to all the nodes, but only be able to run on the number of resources in it.

By assigning a Partition QOS, you can have many more limits to choose from. For example, limits on Memory, GPUs, and others. So if you feel limited to partition restrictions, you can implement QOS limits on a partition.

The Partition QOS will override the job's QOS. If the opposite is desired you need to have the job's QOS have the 'OverPartQOS' flag which will reverse the order of precedence.

For example:



In this diagram, it shows limits and how they are enforced. You can have partition-specific QOS limits, and the job can be submitted to a specific QOS, what happens is that the QOS of the partition will override the job-specific QOS limit.

Floating Partition Example

1. Define a QOS with limit GrpCPU (not GrpNodes) set appropriately
2. Define partition with ALL nodes in the system with the Partition QOS set to the new QOS
3. This partition will only have access to a group of CPUs now on any of the nodes
4. Helpful for debug-like partitions with short run times
5. Makes it so you don't have nodes sitting idle when other jobs could run if they were not carved out in a different partition

If you have a condo model, this can help facilitate the constraints placed on the QOSs on the cluster. The benefit is that if you have carved up the cluster into multiple partitions, you won't have nodes sitting idle when workload is in demand

Partition vs Job QOS

When configuring limits for QOSes, not all the partitions NEED to have a QOS assigned to them. In fact, the following are QOS features that are NOT available on Partition QOSes:

- GraceTime
- UsageFactor
- UsageThreshold
- Preemption rules
- Priority
- All flags except "DenyOnLimit" (DenyOnLimit DOES work on partition QOS.)

Other QOS Options

Flags Used by the `slurmctld` to override or enforce certain characteristics. Valid options are:

DenyOnLimit	If set, jobs using this QOS will be rejected at submission time if they do not conform to the QOS 'Max' limits. Group limits will also be treated like 'Max' limits as well and will be denied if they go over. By default jobs that go over these limits will pend until they conform. This currently only applies to QOS and Association limits.
EnforceUsageThreshold	If set, and the QOS also has a UsageThreshold, any jobs submitted with this QOS that fall below the UsageThreshold will be held until their Fairshare Usage goes above the Threshold.
NoReserve	If this flag is set and backfill scheduling is used, jobs using this QOS will not reserve resources in the backfill schedule's map of resources allocated through time. This flag is intended for use with a QOS that may be preempted by jobs associated with all other QOS (e.g use with a "standby" QOS). If this flag is used with a QOS which can not be preempted by all other QOS, it could result in starvation of larger jobs.
PartitionMaxNodes	If set, jobs using this QOS will be able to override the requested partition's MaxNodes limit.
PartitionMinNodes	If set, jobs using this QOS will be able to override the requested partition's MinNodes limit.
OverPartQOS	If set, jobs using this QOS will be able to override any limits used by the requested partition's QOS limits.
PartitionTimeLimit	If set, jobs using this QOS will be able to override the requested partition's TimeLimit.
RequiresReservation	If set, jobs using this QOS must designate a reservation when submitting a job. This option can be useful in restricting usage of a QOS that may have greater preemptive capability or additional resources to be allowed only within a reservation.
NoDecay	If set, this QOS will not have its GrpTRESMins, GrpWall and UsageRaw decayed by the slurm.conf PriorityDecayHalfLife or PriorityUsageResetPeriod settings. This allows a QOS to provide aggregate limits that, once consumed, will not be replenished automatically. Such a QOS will act as a time-limited quota of resources for an association that has access to it. Account/user usage will still be decayed for associations using the QOS. The QOS GrpTRESMins and GrpWall limits can be increased or the QOS RawUsage value reset to 0 (zero) to again allow jobs submitted with this QOS to be queued (if DenyOnLimit is set) or run (pending with QOSGrp{TRES}MinutesLimit or QOSGrpWallLimit reasons, where {TRES} is some type of trackable resource).
GraceTime	Preemption grace time to be extended to a job which has been selected for preemption.

UsageFactor	Usage factor when running with this QOS (i.e. .5 would make it use only half the time as normal in accounting and 2 would make it use twice as much.) This only applies to the internal wall TRES usage used for limits and determining priority in the slurmctld and doesn't modify the real usage in the database.
UsageThreshold	A float representing the lowest fairshare of an association allowable to run a job. If an association falls below this threshold and has pending jobs or submits new jobs those jobs will be held until the usage goes back above the threshold. Use sshare to see current shares on the system.

Configuration

To summarize the above, the QOS's and their associated limits are defined in the Slurm database using the `sacctmgr` utility. The QOS will only influence job scheduling priority when the multi-factor priority plugin is loaded and a non-zero "PriorityWeightQOS" has been defined in the `slurm.conf` file. The QOS will only determine job preemption when the "PreemptType" is defined as "preempt/qos" in the `slurm.conf` file. Limits defined for a QOS (and described above) will override the limits of the user/account/cluster/partition association.

QOS Examples

QOS manipulation examples. All QOS operations are done using the `sacctmgr` command. The default output of `sacctmgr show qos` is very long given the large number of limits and options available so it is best to use the `format` option which filters the display.

By default when a cluster is added to the database a default qos named normal is created:

```
$ sacctmgr show qos format=name,priority
  Name      Priority
  ----- -----
    normal        0
```

Add a new QOS:

```
$ sacctmgr add qos zebra
Adding QOS(s)
  zebra
  Settings
  Description      = QOS Name

$ sacctmgr show qos format=name,priority
  Name      Priority
  ----- -----
    normal        0
    zebra         0
```

Set QOS priority:

```
$ sacctmgr modify qos zebra set priority=10
Modified qos...
```

```

zebra

$ sacctmgr show qos format=name,priority
  Name      Priority
  -----
  normal        0
  zebra         10

```

Some other limits:

```

$ sacctmgr modify qos zebra set GrpCPUs=24
Modified qos...
zebra

$ sacctmgr show qos format=name,priority,GrpCPUs
  Name      Priority  GrpCPUs
  -----
  normal        0
  zebra         10        24

```

Add a QOS to a user account:

```

$ sacctmgr modify user crock set qos=zebra

$ sacctmgr show assoc format=cluster,user,qos
  Cluster      User          QOS
  -----
canis_major           normal
canis_major       root       normal
canis_major           normal
canis_major       crock      zebra

```

Users can belong to multiple QOSs:

```

$ sacctmgr modify user crock set qos+=alligator
$ sacctmgr show assoc format=cluster,user,qos
  Cluster      User          QOS
  -----
canis_major           normal
canis_major       root       normal
canis_major           normal
canis_major       crock      alligator,zebra

```

Finally, delete a QOS:

```

$ sacctmgr delete qos alligator
Deleting QOS(s)...
alligator

```

Preemption

Slurm supports job preemption, the act of stopping one or more "low-priority" jobs to let a "high-priority" job run. Job preemption is implemented as a variation of Slurm's Scheduling logic. When a high-priority job has been allocated resources that have already been allocated to one or more low priority jobs, the low priority job(s) are preempted. The low priority job(s) can resume once the high priority job completes. Alternately, the low priority job(s) can be requeued and started using other resources if so configured in newer versions of Slurm.

The job's partition priority or its Quality Of Service (QOS) can be used to identify which jobs can preempt or be preempted by other jobs. Slurm offers the ability to configure the preemption mechanism used on a per partition or per QOS basis. For example, jobs in a low priority queue may get requeued, while jobs in a medium priority queue may get suspended.

Preemption configuration

There are a number of important configuration parameters relating to preemption:

SelectType: Slurm job preemption logic supports nodes allocated by the select/linear plugin, socket/core/CPU resources allocated by the select/cons_res plugin or select/cray for Cray systems without ALPS.

SelectTypeParameter: Since resources may be getting over-allocated with jobs (suspended jobs remain in memory), the resource selection plugin should be configured to track the amount of memory used by each job to ensure that memory page swapping does not occur. When select/linear is chosen, we recommend setting SelectTypeParameter=CR_Memory. When select/cons_res is chosen, we recommend including Memory as a resource (ex. SelectTypeParameter=CR_Core_Memory).

NOTE: Unless PreemptMode=SUSPEND,GANG these memory management parameters are not critical.

DefMemPerCPU: Since job requests may not explicitly specify a memory requirement, we also recommend configuring DefMemPerCPU (default memory per allocated CPU) or DefMemPerNode (default memory per allocated node). It may also be desirable to configure MaxMemPerCPU (maximum memory per allocated CPU) or MaxMemPerNode (maximum memory per allocated node) in slurm.conf. Users can use the --mem or --mem-per-cpu option at job submission time to specify their memory requirements.

NOTE: Unless PreemptMode=SUSPEND,GANG these memory management parameters are not critical.

GraceTime: Specifies a time period for a job to execute after it is selected to be preempted. This option can be specified by partition or QOS using the slurm.conf file or database respectively. This option is only honored if PreemptMode=CANCEL. The GraceTime is specified in seconds and the default value is zero, which results in no preemption delay. Once a job has been selected for preemption, its end time is set to the current time plus

GraceTime. The job is immediately sent SIGCONT and SIGTERM signals in order to provide notification of its imminent termination. This is followed by the SIGCONT, SIGTERM and SIGKILL signal sequence upon reaching its new end time.

JobAcctGatherType

JobAcctGatherFrequency: The "maximum data segment size" and "maximum virtual memory size" system limits will be configured for each job to ensure that the job does not exceed its requested amount of memory. If you wish to enable additional enforcement of memory limits, configure job accounting with the JobAcctGatherType and JobAcctGatherFrequency parameters. When accounting is enabled and a job exceeds its configured memory limits, it will be canceled in order to prevent it from adversely effecting other jobs sharing the same resources.

NOTE: Unless PreemptMode=SUSPEND,GANG these memory management parameters are not critical.

PreemptMode: Specifies the mechanism used to preempt low priority jobs. The PreemptMode should be specified for the cluster as a whole, although different values can be configured on each partition when PreemptType=preempt/partition_prio.

NOTE: that when specified on a partition, a compatible mode must also be specified system-wide; specifically if a PreemptMode is set to SUSPEND for any partition, then the system-wide PreemptMode must include the GANG and SUSPEND parameters so the module responsible for resuming jobs executes. Configure to CANCEL, CHECKPOINT, SUSPEND or REQUEUE depending on the desired action for low priority jobs. The GANG option must also be specified if gang scheduling is desired or a

PreemptMode of SUSPEND is used for any jobs.

A value of CANCEL will always cancel the job.

A value of CHECKPOINT will checkpoint (if possible) or kill low priority jobs.

Checkpointed jobs are not automatically restarted.

A value of REQUEUE will requeue (if possible) or kill low priority jobs. Requeued jobs are permitted to be restarted on different resources.

A value of SUSPEND will suspend and resume jobs. If PreemptType=preempt/partition_prio is configured then a value of SUSPEND will suspend and automatically resume the low priority jobs. If PreemptType=preempt/qos is configured, then the jobs sharing resources will always time slice rather than one job remaining suspended. The SUSPEND option must be used with the GANG option (e.g. "PreemptMode=SUSPEND,GANG").

A value of GANG may be used with any of the above values and will execute a module responsible for resuming jobs previously suspended for either gang scheduling or job preemption with suspension.

PreemptType: Configure to the desired mechanism used to identify which jobs can preempt other jobs.

preempt/none indicates that jobs will not preempt each other (default).

preempt/partition_prio indicates that jobs from one partition can preempt jobs from lower priority partitions.

preempt/qos indicates that jobs from one Quality Of Service (QOS) can preempt jobs from a lower QOS. These jobs can be in the same partition or different partitions.

PreemptMode must be set to CANCEL, CHECKPOINT, REQUEUE or SUSPEND. This option requires the use of a database identifying available QOS and their preemption rules.

This option is not compatible with PreemptMode=OFF and PreemptMode=SUSPEND is only supported by the select/cons_res plugin.

PriorityTier: Configure the partition's PriorityTier setting relative to other partitions to control the preemptive behavior when PreemptType=preempt/partition_prio. This option is not relevant if PreemptType=preempt/qos. If two jobs from two different partitions are allocated to the same resources, the job in the partition with the greater PriorityTier value will preempt the job in the partition with the lesser PriorityTier value. If the PriorityTier values of the two partitions are equal then no preemption will occur. The default PriorityTier value is 1.

OverSubscribe: Configure the partition's OverSubscribe setting to FORCE for all partitions in which job preemption using a suspend and resume mechanism is used or NO otherwise. The FORCE option supports an additional parameter that controls how many jobs can oversubscribe a compute resource (FORCE[:max_share]). By default the max_share value is 4. In order to preempt jobs (and not gang schedule them), always set max_share to 1. To allow up to 2 jobs from this partition to be allocated to a common resource (and gang scheduled), set OverSubscribe=FORCE:2.

NOTE: PreemptType=QOS will permit one additional job to be run on the partition if started due to job preemption. For example, a configuration of OverSubscribe=FORCE:1 will only permit one job per resources normally, but a second job can be started if done so through preemption based upon QOS. The use of PreemptType=QOS and PreemptType=Suspend only applies with SelectType=cons_res.

To enable preemption after making the configuration changes described above, restart Slurm if it is already running. Any change to the plugin settings in Slurm requires a full restart of the daemons. If you just change the partition PriorityTier or OverSubscribe setting, this can be updated with scontrol reconfig.

If a job request restricts Slurm's ability to run jobs from multiple users or accounts on a node by using the "--exclusive=user" or "--exclusive=mcs" job options, that may prevent preemption of jobs to start higher priority jobs. If preemption is used, it is generally advisable to disable the "--exclusive=user" and "--exclusive=mcs" job options by using a job_submit plugin (set the value of "shared" to "NO_VAL16").

Preemption Design and Operation

The select plugin will identify resources where a pending job can begin execution. When PreemptMode is configured to CANCEL, CHECKPOINT, SUSPEND or REQUEUE, the select plugin will also preempt running jobs as needed to initiate the pending job. When PreemptMode=SUSPEND,GANG the select plugin will initiate the pending job and rely upon the gang scheduling logic to perform job suspend and resume as described below.

The select plugin is passed an ordered list of preemptable jobs to consider for each pending job which is a candidate to start. This list is sorted by either:

1. QOS priority,
2. Partition priority and job size (to favor preempting smaller jobs), or
3. Job start time (with SchedulerParameters=preempt_youngest_first).

The select plugin will determine if the pending job can start without preempting any jobs and if so, starts the job using available resources. Otherwise, the select plugin will simulate the preemption of each job in the priority ordered list and test if the job can be started after each preemption. Once the job can be started, the higher priority jobs in the preemption queue will not be considered, but the jobs to be preempted in the original list may be sub-optimal. For example, to start an 8 node job, the ordered preemption candidates may be 2 node, 4 node and 8 node. Preempting all three jobs would allow the pending job to start, but by reordering the preemption candidates it is possible to start the pending job after preempting only one job. To address this issue, the preemption candidates are re-ordered with the final job requiring preemption placed first in the list and all of the other jobs to be preempted ordered by the number of nodes in their allocation which overlap the resources selected for the pending job. In the example above, the 8 node job would be moved to the first position in the list. The process of simulating the preemption of each job in the priority ordered list will then be repeated for the final decision of which jobs to preempt. This two stage process may preempt jobs which are not strictly in preemption priority order, but fewer jobs will be preempted than otherwise required. See the SchedulerParameters configuration parameter options of preempt_reordered_count and preempt_strict_order for preemption tuning parameters.

When enabled, the gang scheduling logic (which is also supports job preemption) keeps track of the resources allocated to all jobs. For each partition an "active bitmap" is maintained that tracks all concurrently running jobs in the Slurm cluster. Each partition also maintains a job list for that partition, and a list of "shadow" jobs. The "shadow" jobs are high priority job allocations that "cast shadows" on the active bitmaps of the low priority jobs. Jobs caught in these "shadows" will be preempted.

Each time a new job is allocated to resources in a partition and begins running, the gang scheduler adds a "shadow" of this job to all lower priority partitions. The active bitmap of these lower priority partitions are then rebuilt, with the shadow jobs added first. Any existing jobs that were replaced by one or more "shadow" jobs are suspended (preempted). Conversely, when a high priority running job completes, its "shadow" goes away and the active bitmaps of the lower priority partitions are rebuilt to see if any suspended jobs can be resumed.

The gang scheduler plugin is designed to be reactive to the resource allocation decisions made by the "select" plugins. The "select" plugins have been enhanced to recognize when job preemption has been configured, and to factor in the priority of each partition when selecting resources for a job. When choosing resources for each job, the selector avoids resources that are in use by other jobs (unless sharing has been configured, in which case it does some load-balancing). However, when job preemption is enabled, the select plugins may choose resources that are already in use by jobs from partitions with a lower priority setting, even when sharing is disabled in those partitions.

This leaves the gang scheduler in charge of controlling which jobs should run on the over-allocated resources. If PreemptMode=SUSPEND, jobs are suspended using the same internal functions that support scontrol suspend and scontrol resume. A good way to observe the operation of the gang scheduler is by running squeue -i<time> in a terminal window.

Limitations of Preemption During Backfill Scheduling

For performance reasons, the backfill scheduler reserves whole nodes for jobs, not partial nodes. If during backfill scheduling a job preempts one or more other jobs, the whole nodes for those preempted jobs are reserved for the preemptor job, even if the preemptor job requested fewer resources than that. These reserved nodes aren't available to other jobs during that backfill cycle, even if the other jobs could fit on the nodes. Therefore, jobs may preempt more resources during a single backfill iteration than they requested.

A Simple Example

The following example is configured with select/linear and PreemptMode=SUSPEND,GANG. This example takes place on a cluster of 5 nodes:

```
$ sinfo
PARTITION AVAIL  TIMELIMIT NODES  STATE NODELIST
active*      up    infinite      5  idle  n[12-16]
hipri        up    infinite      5  idle  n[12-16]
```

Here are the Partition settings:

```
$ grep PartitionName /shared/slurm/slurm.conf
PartitionName=DEFAULT OverSubscribe=FORCE:1 Nodes=n[12-16]
PartitionName=active PriorityTier=1 Default=YES
PartitionName=hipri PriorityTier=2
```

The runit.pl script launches a simple load-generating app that runs for the given number of seconds. Submit 5 single-node runit.pl jobs to run on all nodes:

```
$ sbatch -N1 ./runit.pl 300
sbatch: Submitted batch job 485

$ sbatch -N1 ./runit.pl 300
sbatch: Submitted batch job 486

$ sbatch -N1 ./runit.pl 300
sbatch: Submitted batch job 487

$ sbatch -N1 ./runit.pl 300
sbatch: Submitted batch job 488

$ sbatch -N1 ./runit.pl 300
sbatch: Submitted batch job 489

$ squeue -Si
JOBID PARTITION      NAME      USER      ST      TIME      NODES NODELIST
  485      active  runit.pl    user      R  0:06          1  n12
  486      active  runit.pl    user      R  0:06          1  n13
  487      active  runit.pl    user      R  0:05          1  n14
  488      active  runit.pl    user      R  0:05          1  n15
  489      active  runit.pl    user      R  0:04          1  n16
```

Now submit a short-running 3-node job to the hipri partition:

```
$ sbatch -N3 -p hipri ./runit.pl 30
sbatch: Submitted batch job 490

$ squeue -Si
JOBID PARTITION      NAME    USER   ST    TIME   NODES NODELIST
 485    active  runit.pl  user   S    0:27     1  n12
 486    active  runit.pl  user   S    0:27     1  n13
 487    active  runit.pl  user   S    0:26     1  n14
 488    active  runit.pl  user   R    0:29     1  n15
 489    active  runit.pl  user   R    0:28     1  n16
 490    hipri  runit.pl  user   R    0:03     3  n[12-14]
```

Job 490 in the hipri partition preempted jobs 485, 486, and 487 from the active partition. Jobs 488 and 489 in the active partition remained running.

This state persisted until job 490 completed, at which point the preempted jobs were resumed:

```
$ squeue
JOBID PARTITION      NAME    USER   ST    TIME   NODES NODELIST
 485    active  runit.pl  user   R    0:30     1  n12
 486    active  runit.pl  user   R    0:30     1  n13
 487    active  runit.pl  user   R    0:29     1  n14
 488    active  runit.pl  user   R    0:59     1  n15
 489    active  runit.pl  user   R    0:58     1  n16
```

Another Example

In this example we have three different partitions using three different job preemption mechanisms.

Excerpt from *slurm.conf*:

```
PartitionName=low Nodes=linux Default=YES
PartitionName=low OverSubscribe=NO PriorityTier=10 PreemptMode=requeue

PartitionName=med Nodes=linux Default=NO
PartitionName=med OverSubscribe=FORCE:1 PriorityTier=20 PreemptMode=suspend

PartitionName=hi  Nodes=linux Default=NO
PartitionName=hi  OverSubscribe=FORCE:1 PriorityTier=30 PreemptMode=off
```

```
$ sbatch tmp
Submitted batch job 94

$ sbatch -p med tmp
Submitted batch job 95

$ sbatch -p hi tmp
Submitted batch job 96
$ squeue
JOBID PARTITION      NAME    USER   ST    TIME   NODES NODELIST (REASON)
  96      hi      tmp    moe   R    0:04     1  linux
  94      low      tmp    moe   PD   0:00     1  (Resources)
  95      med      tmp    moe   S    0:02     1  linux
```

```
(after job 96 completes)

$ squeue
JOBID PARTITION      NAME      USER   ST          TIME  NODES NODELIST (REASON)
 94      low        tmp      moe   PD          0:00      1  (Resources)
 95      med        tmp      moe     R          0:24      1  linux
```

Another Example

Excerpt from *slurm.conf*:

```
PreemptMode=Suspend,Gang
PreemptType=preempt/qos
PartitionName=normal Nodes=linux Default=YES OverSubscribe=FORCE:1
```

Quality of Service (QOS) Lab Exercises

In this set of labs, you will learn how QOS-based preemption is configured. In another training module (Job Priority), we talk about prioritizing jobs based on QOS assignment. In that discussion, we create several QOSs and assign priority to them, in order to see how the scheduler manages the jobs

Lab Setup-DO THIS BEFORE UNDERTAKING THE EXERCISES

1. In order to get that environment setup, as root user, create a script called `qos.sh` and add the following:

NOTE: If you are copying and pasting from this lab manual, and using vi or vim, use “:set paste” to preserve formatting. Compare this document with what you have in your .sh file to make sure it is appropriately formatted.

NOTE: This script is also found in the `/lab_scripts` folder in the Login Docker container.

```
#!/bin/bash
# Script to setup for the QOS-based preemption lab
echo "PriorityWeightQOS=1000000" >> /etc/slurm/slurm.conf
/lab_scripts/restart.sh
sleep 3
#Create the QOSs:
sacctmgr -i add qos high
sacctmgr -i add qos medium
sacctmgr -i add qos low
# Assign all users, cluster and accounts to the low QOS, and make the low
# QOS the default for all users, cluster and accounts:
sacctmgr -i modify account bedrock set qos=low
sacctmgr -i modify user \
root,arnold,bambam,barney,betty,chip,dino,edna,fred,gazoo,pebbles, \
slurm,wilma set qos=low
sacctmgr -i modify cluster cluster set qos=low
sacctmgr -i modify account bedrock set defaultqos=low
sacctmgr -i modify cluster cluster set defaultqos=low

#Assign users to be able to use the QOSs:
sacctmgr -i modify user fred,barney set qos=+high,+medium,+low
sacctmgr -i modify user fred,barney,wilma,betty set qos=+medium,+low
```

2. Flag it executable.
3. As root, run the qos.sh script:
`qos.sh`
4. Verify the configuration looks right by running:
`sacctmgr list assoc format=Cluster,Account,User,QOS,defaultqos`

Should show:

Cluster	Account	User	QOS	Def QOS
cluster	root		low	low
cluster	root	root	low	low
cluster	bedrock		low	low
cluster	bedrock	arnold	low	low
cluster	bedrock	bambam	low	low
cluster	bedrock	barney	high,low,medium	low
cluster	bedrock	betty	low,medium	low
cluster	bedrock	chip	low	low
cluster	bedrock	dino	low	low
cluster	bedrock	edna	low	low
cluster	bedrock	fred	high,low,medium	low
cluster	bedrock	gazoo	low	low
cluster	bedrock	pebbles	low	low
cluster	bedrock	slurm	low	low
cluster	bedrock	wilma	low,medium	low

Exercise 1: Configure for QOS-based job preemption

- As root, edit the `/etc/slurm/slurm.conf` file and add the global preemption policy by inserting the following lines in the file:

```
PreemptMode=requeue,gang
PreemptType=preempt/qos
```

And uncommenting:

```
#PriorityType=priority/multifactor
```

To:

```
PriorityType=priority/multifactor
```

- Save and exit the `slurm.conf` file.
- Commit the qos priority to the database: (The “-i” on the following `sacctmgr` command means to accept the yes/no prompt automatically):


```
sacctmgr -i modify qos high set priority=10000
sacctmgr -i modify qos medium set priority=5000
sacctmgr -i modify qos low set priority=100
```
- Reload the configuration file into memory:

```
/lab_scripts/restart.sh
```

- In order for Slurm to actually preempt workload, there needs to be a change in sacctmgr. To see the default, type the following:

```
sacctmgr show qos format=name,priority,preempt
```

You should see:

Name	Priority	Preempt
normal	0	
high	10000	
medium	5000	
low	100	

- Make the change in sacctmgr to allow preemption of high over low and medium qos jobs, and for medium over low qos jobs:

```
sacctmgr -i mod qos high set preempt=medium,low  
sacctmgr -i mod qos medium set preempt=low
```

- Run sacctmgr again and you should see the *Preempt* column filled out:

```
sacctmgr show qos format=name,priority,preempt
```

You should see:

Name	Priority	Preempt
normal	0	
high	10000	low,medium
medium	5000	low
low	100	

- In a secondary terminal window, launch the following (which is all one line, but wraps in this document):

```
watch -c "squeue;sinfo -pdebug -N1;sprio -l;sacctmgr show qos  
format=name,priority,preempt"
```

- As **pebbles**, submit a 10-node exclusive sleep job to the low QOS:

```
sbatch -N10 --mem=1000 --requeue --exclusive --wrap="sleep 100"
```

- As **fred**, submit a 10-node exclusive sleep job to the high QOS:

```
sbatch -N10 --mem=1000 --requeue --qos=high --exclusive --wrap="sleep 100"
```

You will see the preemption happen.

- In the logs, you can also see the preemption results.

12. Exit from user fred.
13. As root, view the logs over on the mgmtnode, using SSH:
`ssh mgmtnode cat /var/log/slurmctld.log | grep --color=auto preempted`

You will see a log entry something similar to this:

```
[2021-10-01T12:50:45.389] preempted JobId=1 has been requeued to reclaim  
resources for JobId=2
```

14. Repeat steps 1-2, but change the PreemptMode

from:

```
PreemptMode=requeue,gang
```

to:

```
PreemptMode=cancel,gang
```

15. Repeat steps 4-14, to see what happens because of the configuration change to *cancel,gange*.

The log entry will change:

```
[2021-10-01T12:50:45.389] preempted JobId=1 has been requeued to reclaim  
resources for JobId=2  
[2021-10-01T13:01:12.517] preempted JobId=3 has been killed to reclaim resources  
for JobId=4
```

Cleanup

1. Reset the Docker Slurm cluster environment back to its original, pristine state by exiting the login Docker container back to the Ubuntu AWS prompt. And as **ubuntu**:
`cd ~/docker-scale-out
make clean && make`

The Infrastructure for Building Slurm

Slurm is relatively simple to build and install.

The link to download the source code for Slurm is:

<https://slurm.schedmd.com/download.html>

It is also available on github for downloading.

The required infrastructure to support Slurm on any platform are:

- **MUNGE:** MUNGE stands for Munge Uid "n" Gid Emporium. It was developed at Lawrence Livermore Labs. It provides the authentication and digital signature and encryption mechanism for secure communication. It is very high throughput, 40-50k records checked in on each node. It does require a key on each node, as well as the munge-dev package with headers and library.
- **NTP:** Time is crucial to maintain the proper communication between the controller and the slurm daemons. If time does get out of sync, then timestamps on scheduler objects become wrong. The Munge provides encryption on the scheduler objects. That key can only be un-encrypted within a certain, configurable time. If that time has expired, or the dates between the scheduler and slurmd's view of them is incorrect, then it will mess up the controller.
- **MySQL or MariaDB:** A relational database is required for accounting records to be stored. The slurmdbd communicates with the database to store and retrieve, among other things, job history information.
- **hwloc** Portal Hardware Locality library and tools: This set of tools allow the configuration topology (Sockets, cores, memory, threads, cache, etc) to be read into the controller. This package also installs lstopo, a tool for viewing the hardware layout of the compute nodes.

Database

When using either MySQL or MariaDB, defining the database and configuring access for users is crucial for Slurm to function correctly. The slurmdbd will create and populate the tables as needed when it runs for the first time. In addition, slurmdbd will update the tables as needed when performing an upgrade to Slurm.

Depending on how you install Slurm, you may need to manually bring it up before proper function. An example of how to grant access to the database to the slurm_db user in the database, you would do something like this:

```
$ mysql  
MariaDB [(none)]> grant all on slurm_acct_db.* TO 'slurm_db@localhost';  
MariaDB [(none)]> grant all on slurm_acct_db.* to 'slurm_db@<hostname>';  
MariaDB [(none)]> create database slurm_acct_db;  
MariaDB [(none)]> exit
```

Slurm Versions

Slurm has a major release about every 9 months. To identify which version of Slurm you are using, look at the name of the downloaded Slurm source. The first two digits represent the major release: Version 17.11.# was released November 2017 (2017-11). Version 18.08.# was released August 2018 (2018-08).

The third digit represents the bug fix version. 17.11.1 contains bug fixes from 17.11.0. Version 18.08.4 contains bug fixes from 18.08.1 through 18.08.4. The bug fix releases are typically released about once a month. For

optimal reliability after a major release, wait for several releases with bug fixes before deploying. This lets other people sort out any problems that may have been missed in previous patch releases.

Building Slurm

As previously mentioned, you can download the Slurm code from here:

<https://slurm.schedmd.com/download.html>

This includes the full source in a tarball distribution. It is possible to create an RPM from source, using rpmbuild, for example. Once you have downloaded and extracted the source tarball, you can issue rpmbuild like this:

```
# rpmbuild -ta slurm.#.#.#.tar.bz2  
# rpm --install <the rpm files>
```

Some of the RPMs are system specific and optional. For example, the distribution contains "Wrapper" scripts for pbs and torque users, that allow them to continue to submit as they have in those environments. Also, "sjobexit" contains a tool for job exit code management.

Configuring and Starting Slurm

Slurm is managed by configuring a flat text file called slurm.conf. When you install Slurm, there is an example slurm.conf that you can modify for your own purposes. If you want to configure yours using some helps, there is a web-based configuration tool called the configurator. You can access this by pointing your browser to:

<https://slurm.schedmd.com/configurator>

and

<https://slurm.schedmd.com/configurator.easy.html>

Once you have answered all the fields in the configurator, you copy and paste the contents into your *slurm.conf* and restart Slurm.

Once you have a configuration file built, you can start the Slurm daemons. The order in which you start them is not so important. But here is the suggested order:

1. Start the `slurmdbd` on the site-wide server. The first time this is ever run it will build the tables in the database.
2. Start `slurmctld` on a head node. The first thing the Slurm Control daemon does is talk to the database, to find out where the database is and if it's running. If it can, it will pull information from the database and cache it. It does this so that if the slurmdbd is ever down, it can read from the cache file and at least get a starting point from which to launch.
3. Start `slurmd` on a compute node. There is a special switch for the slurmd called "-C". This will print the configuration about the compute node it sends to the control daemon.

There are some options to these daemons, which can be used for basic troubleshooting. If you run any of them with -D option, it will run the daemon in foreground and all messages output to the terminal. If you start the

daemons with the `-v` switch (verbose), then you get more messaging. You can also append more `-v`'s, such as `-vvv` for very verbose output.

Building and Installing Slurm Exercises

The learning environment has already been installed. However, the instructor may wish to reset the learning environment so that you can install the Slurm software. There are 2 methods: 1) Configure from source method, and 2) RPM Build method

Configure Method

Using this script, you can install the slurm software in the training environment from source.

1. Copy and paste the following script as root into the mgmtnode container:

```
#!/bin/bash
#
# This is the Slurm Installation script
# Run it on mgmtnode as root
#
#####
# Install epel-release and other software
#####

mkdir /etc/PDSH
echo "node00
node01
node02
node03
node04
node05
node06
node07
node08
node09" > /etc/PDSH/hosts
export WCOLL=/etc/PDSH/hosts
yum install -y epel-release
yum install -y pdsh bind-utils expect gcc make git gtk2-devel hdf5-devel
hwloc hwloc-devel hwloc-gui libibmad libibumad libcurl libssh2-devel lua
lua-devel man man2html mariadb mariadb-devel mariadb-server munge munge-
devel munge-libs net-tools ncurses-devel nfs-utils numactl numactl-devel
openssh-server openssl-devel pam-devel perl-ExtUtils-MakeMaker python-pip
readline-devel rpm-build rrdtool-devel screen vim wget rpm-devel
pdsh "yum install -y epel-release"
pdsh "yum install -y pdsh bind-utils expect gcc make git gtk2-devel hdf5-
devel hwloc hwloc-devel hwloc-gui libibmad libibumad libcurl libssh2-devel
lua lua-devel man man2html mariadb mariadb-devel mariadb-server munge
munge-devel munge-libs net-tools ncurses-devel nfs-utils numactl numactl-
devel openssh-server openssl-devel pam-devel perl-ExtUtils-MakeMaker
python-pip readline-devel rpm-build rrdtool-devel screen vim wget"
```

(script content continues on the next page)

```

#####
# Install munge, make key, distribute, start
#####

systemctl enable munge
pdsh "systemctl enable munge"
dd if=/dev/urandom bs=1 count=1024 > /etc/munge/munge.key
chown munge:munge /etc/munge/munge.key
chmod 400 /etc/munge/munge.key
pdcp /etc/munge/munge.key /etc/munge
pdsh "chown munge:munge /etc/munge/munge.key"
pdsh "chmod 400 /etc/munge/munge.key"

systemctl start munge
pdsh "systemctl start munge"

#####
# Enable and start MariaDB
#####

systemctl enable mariadb
systemctl start mariadb

#####
# Download Slurm
#####

cd ~/Software
wget https://download.schedmd.com/slurm/slurm-22.05.1.tar.bz2
pdcp slurm-22.05.1.tar.bz2 /root/Software

#####
# Extract Slurm
#####

tar jxvf slurm-22.05.1.tar.bz2
pdsh "tar jxvf ~/Software/slurm-22.05.1.tar.bz2 -C ~/Software"

#####
# Configure and install Slurm
#####


```

(script content continues on the next page)

```

cd ~/Software/slurm-22.05.1
./configure
make && make install
pdsh "cd ~/Software/slurm-22.05.1 ; ./configure"
pdsh "cd ~/Software/slurm-22.05.1 ; make && make install"
cp ~/Software/slurm-22.05.1/etc/slurmctld.service /usr/lib/systemd/system
cp ~/Software/slurm-22.05.1/etc/slurmdbd.service /usr/lib/systemd/system

mkdir -p /var/log/slurm
touch /var/log/slurm/slurmdbd.log
chown slurm:slurm /var/log/slurm/slurmdbd.log
mkdir -p /var/spool/slurmctld
chown slurm:slurm /var/spool/slurmctld

systemctl enable slurmctld
systemctl enable slurmdbd

cd /usr/local/etc/
wget --user=guest ==password=slurm http://hoopesfamily.net/slurmdbd.conf
wget --user=guest ==password=slurm http://hoopesfamily.net/slurm.conf

pdcp /usr/local/etc/slurm.conf /usr/local/etc
pdcp ~/Software/slurm-22.05.1/etc/slurmd.service /usr/lib/systemd/system

#####
# Enable and start MariaDB
#####

systemctl enable mariadb
systemctl start mariadb
mysql -u root -Bse "grant all on slurm_acct_db.* TO 'slurm'@'localhost' identified by 'slurm' with grant option;" 

echo "#"
# slurmDBD info
DbdAddr=localhost
DbdHost=localhost
#DbdPort=7031
SlurmUser=slurm
#MessageTimeout=300
DebugLevel=4

```

(script content continues on the next page)

```

#DefaultQOS=normal,standby
LogFile=/var/log/slurm/slurmdbd.log
PidFile=/var/run/slurmdbd.pid
#PluginDir=/usr/lib/slurm
#PrivateData=accounts,users,usage,jobs
#TrackWCKey=yes
#
# Database info
StorageType=accounting_storage/mysql
#StorageHost=localhost
#StoragePort=1234
StoragePass=slurm
StorageUser=slurm
#StorageLoc=slurm_acct_db
" > /usr/local/etc/slurmdbd.conf

systemctl start slurmdbd
systemctl start slurmctld

pdsh "systemctl enable slurmd"
pdsh "systemctl start slurmd"

#####
# Fetch the clean.sh script and run it
#####

cd /usr/local/bin
wget http://hoopesfamily.net/clean_CONFIGURE.sh
chmod +x clean_CONFIGURE.sh
mv clean_CONFIGURE clean.sh
/usr/local/bin/clean.sh

#####
# Add WCOLL env to bashrc
#####
echo -e "export WCOLL=/etc/PDSH/hosts">>/etc/bashrc

```

2. Flag it executable and run it.

RPM Method

Using this script, you can install the slurm software in the training environment from RPM Build.

1. Copy and paste the following script as root into the mgmtnode container:

```

#!/bin/bash
#
# This is the Slurm Installation script
# Run it on mgmtnode as root
#
#####
# Install epel-release and other software
#####

mkdir /etc/PDSH
echo "node00
node01
node02
node03
node04
node05
node06
node07
node08
node09" > /etc/PDSH/hosts
export WCOLL=/etc/PDSH/hosts

yum install -y epel-release

yum install -y pdsh bind-utils expect gcc make git gtk2-devel hdf5-devel hwloc \
hwloc-devel hwloc-gui libibmad libibumad libcurl libssh2-devel lua lua-devel man \
man2html mariadb mariadb-devel mariadb-server munge munge-devel munge-libs \
net-tools ncurses-devel nfs-utils numactl numactl-devel openssh-server \
openssl-devel pam-devel perl-ExtUtils-MakeMaker python-pip readline-devel \
rpm-build rrdtool-devel screen vim wget rpm-devel perl-Switch postfix mailx

pdsh "yum install -y epel-release"

pdsh "yum install -y pdsh bind-utils expect gcc make git gtk2-devel hdf5-devel \
hwloc hwloc-devel hwloc-gui libibmad libibumad libcurl libssh2-devel lua \
lua-devel man man2html mariadb mariadb-devel mariadb-server munge munge-devel \
munge-libs net-tools ncurses-devel nfs-utils numactl numactl-devel \
openssl-server openssl-devel pam-devel perl-ExtUtils-MakeMaker python-pip \
readline-devel rpm-build rrdtool-devel screen vim wget mailx"

systemctl enable postfix
systemctl start postfix

```

(script content continues on the next page)

```

#####
# Install munge, make key, distribute, start
#####

systemctl enable munge
pdsh "systemctl enable munge"
dd if=/dev/urandom bs=1 count=1024 > /etc/munge/munge.key
chown munge:munge /etc/munge/munge.key
chmod 400 /etc/munge/munge.key
pdcp /etc/munge/munge.key /etc/munge
pdsh "chown munge:munge /etc/munge/munge.key"
pdsh "chmod 400 /etc/munge/munge.key"

systemctl start munge
pdsh "systemctl start munge"

#####
# Download Slurm
#####

cd ~/Software
wget https://download.schedmd.com/slurm/slurm-22.05.1.tar.bz2

#####
# RPM Build Slurm and install
#####

rpmbuild -ta slurm-22.05.1.tar.bz2
ls -l ~/rpmbuild/RPMS/x86_64/*.rpm
rpm -Uvh ~/rpmbuild/RPMS/x86_64/*.rpm
pdcp ~/rpmbuild/RPMS/x86_64/slurm-slurmd-22.05.1.el7.x86_64.rpm /root/Software
pdcp ~/rpmbuild/RPMS/x86_64/slurm-22.05.1.el7.x86_64.rpm /root/Software
pdsh "yum install -y /root/Software/slurm-22.05.1.el7.x86_64.rpm"
pdsh "yum install -y /root/Software/slurm-slurmd-22.05.1.el7.x86_64.rpm"
pdsh "systemctl enable slurmd"

```

(script content continues on the next page)

```

#####
# Enable and start MariaDB
#####

systemctl enable mariadb
systemctl start mariadb
mysql -u root -Bse "grant all on slurm_acct_db.* TO 'slurm'@'localhost'
identified by 'slurm' with grant option;"

echo "#"
# slurmDBD info
DbdAddr=localhost
DbdHost=localhost
#DbdPort=7031
SlurmUser=slurm
#MessageTimeout=300
DebugLevel=4
#DefaultQOS=normal,standby
LogFile=/var/log/slurm/slurmdbd.log
PidFile=/var/run/slurmdbd.pid
#PluginDir=/usr/lib/slurm
#PrivateData=accounts,users,usage,jobs
#TrackWCKey=yes
#
# Database info
StorageType=accounting_storage/mysql
#StorageHost=localhost
#StoragePort=1234
StoragePass=slurm
StorageUser=slurm
#StorageLoc=slurm_acct_db
" > /etc/slurm/slurmdbd.conf

mkdir /var/log/slurm
chown slurm:slurm /var/log/slurm
touch /var/log/slurm/slurmdbd.log
chown slurm:slurm /var/log/slurm/slurmdbd.log

systemctl enable slurmdbd
systemctl start slurmdbd

```

(script content continues on the next page)

```

#####
# Configure and start Slurm
#####
mkdir -p /var/spool/slurmctld
chown slurm:slurm /var/spool/slurmctld
echo "
SlurmctldHost=mgmtnode
#PrologFlags=x11
Mpidefault=none
ProctrackType=proctrack/linuxproc
ReturnToService=1
SlurmctldPidFile=/var/run/slurmctld.pid
SlurmdPidFile=/var/run/slurmd.pid
SlurmdSpoolDir=/var/spool/slurmd
SlurmUser=slurm
StateSaveLocation=/var/spool/slurmctld
SwitchType=switch/none
TaskPlugin=task/affinity
FastSchedule=1
SchedulerType=sched/backfill
SelectType=select/cons_res
SelectTypeParameters=CR_Core
AccountingStorageType=accounting_storage/slurmdbd
AccountingStorageEnforce=assoc,limits,qos,safe
ClusterName=cluster
JobAcctGatherType=jobacct_gather/linux
JobCompType=jobcomp/none
SlurmctldLogFile=/var/log/slurmctld.log
SlurmdLogFile=/var/log/slurmd.log
SlurmctldDebug=debug
PartitionName=debug Nodes=node0[0-9] Default=YES MaxTime=INFINITE State=UP
NodeName=node0[0-9] CPUs=4 Boards=1 SocketsPerBoard=1 CoresPerSocket=4
ThreadsPerCore=1 RealMemory=16046
PriorityType=priority/multifactor
" > /etc/slurm/slurm.conf

systemctl start slurmctld
systemctl status slurmctld
systemctl enable slurmctld

```

(script content continues on the next page)

```
#####
# Fetch the clean.sh script and run it
#####

cd /usr/local/bin
wget http://hoopesfamily.net/clean_RPM.sh
chmod +x clean_RPM.sh
mv clean_RPM.sh clean.sh
/usr/local/bin/clean.sh

#####
# Add WCOLL env to bashrc
#####
echo -e "export WCOLL=/etc/PDSH/hosts">>>/etc/bashrc
```

Building The Learning Environment for Yourselves

The Slurm cluster and accompanying supporting software can be installed in your own environment. Either on a stand-alone computer or in a Virtual Machine. The requirements are:

- **Ubuntu 20.04 LTS, 4 CPUs, 16GB, 25GB disk, User=ubuntu, Password=ubuntu**

- Once you have your Ubuntu VM/Computer installed, log in as user: **root**
- Open a terminal, and enter the following (**all on one line**):
`sudo apt -y update && apt -y upgrade && apt -y install curl gcc make
wget jq`
- Next, the following command (**all on one line**), to begin setting up the docker-compose environment:
`sudo curl -L sudo curl -SL
https://github.com/docker/compose/releases/download/v2.14.2/docker-
compose-linux-x86_64 -o /usr/local/bin/docker-compose`

- Now, enter all of the following command (**in a series**):

```
curl -sSL https://get.docker.com/ | sudo bash  
  
rm -rf /usr/bin/docker-compose  
  
ln -s /usr/local/bin/docker-compose /usr/bin/docker-compose  
  
chmod +x /usr/local/bin/docker-compose  
  
usermod -a -G docker ubuntu  
  
reboot
```

- After **reboot**, login as:

User: **ubuntu**
Password: **ubuntu**

- Switch to the root user:

```
sudo su -
```

- As **root**, enter the following (as all one command -- the quotes will allow everything to be entered at the same time -- copy/paste the whole block or type it exactly as it appears [quotes are in **blue** to make them easy to see exactly where they begin and end]):

```
echo "  
net.ipv4.tcp_max_syn_backlog=4096  
net.core.netdev_max_backlog=1000  
net.core.somaxconn=15000  
fs.file-max=992832  
  
# Force gc to clean-up quickly  
net.ipv4.neigh.default.gc_interval = 3600  
  
# Set ARP cache entry timeout  
net.ipv4.neigh.default.gc_stale_time = 3600  
  
# Setup DNS threshold for arp  
net.ipv4.neigh.default.gc_thresh3 = 8096
```

```
net.ipv4.neigh.default.gc_thresh2 = 4048
net.ipv4.neigh.default.gc_thresh1 = 1024

# Increase map count for elasticsearch
vm.max_map_count=262144

# Avoid running out of file descriptors
fs.file-max=10000000
fs.inotify.max_user_instances=65535
fs.inotify.max_user_watches=1048576
" >> /etc/sysctl.conf
```

8. Update sysctl:

```
sysctl --system
```

9. Exit from root, back to **ubuntu** user.

10. Enter the following as **all one line**:

```
git clone --depth 1 --single-branch -b master
https://gitlab.com/SchedMD/training/docker-scale-out.git
```

11. Switch to the docker-scale-out directory:

```
cd docker-scale-out
```

12. Change the version of Slurm you want to build

```
export SLURM_RELEASE=slurm-23-02-4-1
```

13. Build the docker containers using docker-compose and make:

```
make build
```

NOTE: This takes a minute or two. (Or 20?). Once the command line prompt re-appears, the docker scale out (DSO) environment should be all done and ready to use.

15. See the Docker containers:

```
cd docker-scale-out
docker ps --format "{{.Names}}: {{.Status}}" | column -t
```

You should see:

docker-scale-out-proxy-1:	Up	About	an	hour
docker-scale-out-node08-1:	Up	About	an	hour
docker-scale-out-node09-1:	Up	About	an	hour
docker-scale-out-node05-1:	Up	About	an	hour
docker-scale-out-node01-1:	Up	About	an	hour
docker-scale-out-node02-1:	Up	About	an	hour
docker-scale-out-node06-1:	Up	About	an	hour
docker-scale-out-node00-1:	Up	About	an	hour
docker-scale-out-mgmtnode2-1:	Up	About	an	hour
docker-scale-out-node07-1:	Up	About	an	hour
docker-scale-out-rest-1:	Up	About	an	hour
docker-scale-out-node03-1:	Up	About	an	hour
docker-scale-out-node04-1:	Up	About	an	hour
docker-scale-out-mgmtnode-1:	Up	About	an	hour
docker-scale-out-open-on-demand-1:	Up	About	an	hour
docker-scale-out-kibana-1:	Up	About	an	hour
docker-scale-out-slurmdbd-1:	Up	About	an	hour
docker-scale-out-es03-1:	Up	About	an	hour
docker-scale-out-influxdb-1:	Up	About	an	hour
docker-scale-out-login-1:	Up	About	an	hour
docker-scale-out-grafana-1:	Up	About	an	hour
docker-scale-out-es02-1:	Up	About	an	hour
docker-scale-out-es01-1:	Up	About	an	hour
docker-scale-out-db-1:	Up	About	an	hour
docker-scale-out-xdmod-1:	Up	About	an	hour

16. To connect to the login Docker container, use the following method:

docker-compose exec login bash

You should see:

```
[root@login ~]#
```

You can now use the DSO environment to do the other labs

Node and Partition Configuration Discussion

Slurm schedules workload in High Performance Computing (HPC) systems by determining the best place for a job to run and then sending the job to those resources. One way Slurm manages the location of hardware is by way of partition and nodes.

Nodes

The slurmd presents information to the scheduler about the state and configuration of each node in the cluster. There are multiple ways to refer to nodes. The configuration for nodes can be stored in the `slurm.conf`. The entries that make up the node configuration are:

- **NodeName:** This is the label that Slurm will reference a node in the cluster. It may or not be the actual hostname of the node.
- **NodeHostName:** This is the name that returns from running `/bin/hostname` on the node. It's the legal DNS name of the node in the cluster.
- **NodeAddr:** This is the name or IP address that Slurm attempts to communicate with the node on. If you are doing testing, it is possible to simulate multiple nodes on a single host by referencing a unique NodeAddr for each node.
- **Port:** This is the port over which slurmd uses to communicate with the controller

There are many node configuration parameters that can be placed in the `slurm.conf` which let Slurm know about the attributes of each node. Again, for testing and simulation purposes, you can artificially inflate these numbers to see how scheduling may affect your workflow. Here are a few examples:

- Boards=#
- CoresPerSocket=#
- CPUS=#
- RealMemory=#
- Sockets=#
- SocketsPerBoard=#
- ThreadsPerCore=#

When configuring these parameters on real nodes, make sure to make them match the physical properties on those nodes. For example, if you configure `RealMemory=256GB`, but the node actually has less, then Slurm will mark that node as DOWN. That way you won't have your job looking for 256GB but landing on a job with less and thus breaking your application.

Another option would be to configure a node in the configuration file with less memory than it physically has, thus leaving memory for various system processes. The values listed in the config file are used for determining job placement.

Other node configuration parameters include:

Features An arbitrary string used to place a label on a node, such as a node having a PHI or NVidia GPU.

Gres	Generic resource and counts. This can be used to reference a license when scheduling workload on a node running an application that requires one.
Reason	A string describing the reason for a node state. For example, why this node has been marked DOWN or DRAINED. This is required when an administrator is marking a node DOWN or DRAINED.
State	A healthy, unused node will report as IDLE. A node can be reported as UNKNOWN to recover from a saved state. It could also be marked explicitly as DOWN, DRAINED, FAIL, and others
Weight	This is the node scheduling weight. When jobs are allocated to resources, and all other things being equal, they are going to go to the nodes that have the lower weight. You may want to assign a lower weight to a node that has less memory than others, so that jobs that do not explicitly request lots of memory will be sent to them. Those nodes that are considered more valuable, like having GPUs or more memory, you could assign them a higher weight value. That would tend to leave those nodes more idle unless the less-desirable are filled up or someone asks for the special nodes specifically.

Sample Node Configuration

Here is an example of how node entries may look in the `slurm.conf` file:

```
NodeName=DEFAULT Sockets=2 CoresPerSocket=8 ThreadsPerCore=2
NodeName=tux[0-1023] RealMemory=2048 Weight=2
NodeName=tux[1024-2000] RealMemory=4096 Weight=4
DownNodes=tux123 State=DOWN Reason=PDU
DownNodes=tux126 State=DOWN Reason=Fan
```

Whenever you label a node name with DEFAULT, the policies will carry down to all nodes.

With the Weight policy enforced here, all other things being equal, the jobs will fill up nodes 0-1023 first, because they have the lower weight value of 2. Of course, if a user requests more than 2048, then it will fall on nodes 1024-2000.

The DownNodes Parameter

It is possible to add entries in the `slurm.conf` file to identify which nodes may be problematic. Normally you would use the scontrol command to mark the nodes offline. For example, to accomplish the same thing, you would enter the following in the `slurm.conf`:

```
# scontrol update NodeName=node1 State=DOWN Reason="Maintenance"

# scontrol show node node1
  NodeName=node1 Arch=x86_64 CoresPerSocket=4
  CPUAlloc=0 CPUTot=4 CPULoad=0.15
  AvailableFeatures=(null)
  ActiveFeatures=(null)
  Gres=(null)
  NodeAddr=localhost NodeHostName=node1 Port=7001 Version=18.08
  OS=Linux 3.10.0-957.1.3.el7.x86_64 #1 SMP Thu Nov 29 14:49:43 UTC 2018
  RealMemory=980 AllocMem=0 FreeMem=423 Sockets=1 Boards=1
  State=DOWN ThreadsPerCore=1 TmpDisk=0 Weight=1 Owner=N/A MCS_label=N/A
```

```

Partitions=debug
BootTime=2019-02-06T14:36:29 SlurmdStartTime=2019-02-06T14:36:39
CfgTRES=cpu=4,mem=980M,billing=4
AllocTRES=
CapWatts=n/a
CurrentWatts=0 LowestJoules=0 ConsumedJoules=0
ExtSensorsJoules=n/s ExtSensorsWatts=0 ExtSensorsTemp=n/s
Reason=Maintenance [user1@2019-02-06T14:39:27]

```

To accomplish the same thing, you would enter the following in the `slurm.conf`:

```

# echo 'DownNodes=node1 Reason=Maintenance State=DOWN' >> /etc/slurm/slurm.conf

# systemctl restart slurcmctld

# scontrol show node node1
NodeName=node1 Arch=x86_64 CoresPerSocket=4
CPUAlloc=0 CPUTot=4 CPULoad=0.08
AvailableFeatures=(null)
ActiveFeatures=(null)
Gres=(null)
NodeAddr=localhost NodeHostName=node1 Port=7001 Version=18.08
OS=Linux 3.10.0-957.1.3.el7.x86_64 #1 SMP Thu Nov 29 14:49:43 UTC 2018
RealMemory=980 AllocMem=0 FreeMem=409 Sockets=1 Boards=1
State=DOWN ThreadsPerCore=1 TmpDisk=0 Weight=1 Owner=N/A MCS_label=N/A
Partitions=debug
BootTime=2019-02-06T14:36:29 SlurmdStartTime=2019-02-06T14:36:39
CfgTRES=cpu=4,mem=980M,billing=4
AllocTRES=
CapWatts=n/a
CurrentWatts=0 LowestJoules=0 ConsumedJoules=0
ExtSensorsJoules=n/s ExtSensorsWatts=0 ExtSensorsTemp=n/s
Reason=Maintenance [slurm@2019-02-06T14:45:33]

```

Partitions

Slurm partitions group nodes into logical (possibly overlapping) sets of nodes. The partitions can be considered job queues, each of which has an assortment of constraints such as job size limit, job time limit, users permitted to use it, etc. Priority-ordered jobs are allocated nodes within a partition until the resources (nodes, processors, memory, etc.) within that partition are exhausted

One use of partitions is to apply access rights to them. For example, you could restrict certain accounts from using nodes allocated in a partition. Or allow certain groups to use a specific partition.

A partition in Slurm functions similarly to a queue in torque/PBS or a class in Moab.

Here are a list of common policies regarding partitions that you can declare in the `slurm.conf`:

AllocNodes	Only jobs submitted from the listed nodes can use partition
AllowAccounts	A comma-separated list of accounts permitted to use this partition
DenyAccounts	A comma-separated list of accounts prevented from using this partition

AllowGroups	A comma-separated list of groups permitted to use this partition
AllowQos	A QOS that is prevented from using this partition
DisableRootJobs	Root is not permitted to run jobs in this partition

It is also possible to apply default memory management resource assignments, in `slurm.conf`, for jobs submitted to a partition:

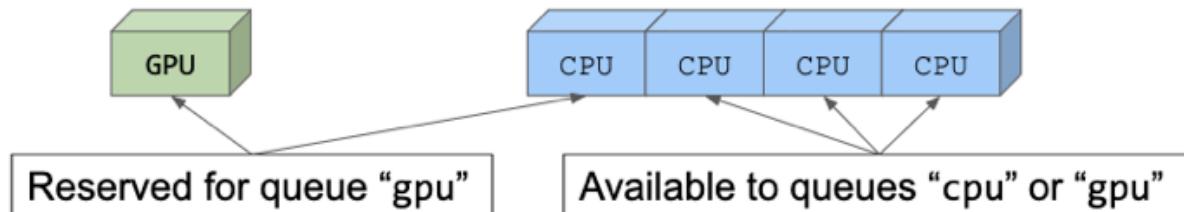
DefMemPerCPU	The default amount of memory allocated to jobs per CPU in this partition
DefMemPerNode	The default amount of memory allocated to jobs per node in this partition
MaxMemPerCPU	The maximum amount of memory allocated to jobs per CPU in this partition
MaxMemPerNode	The maximum amount of memory allocated to jobs per node in this partition

NOTE: Choose either PerCPU or PerNode values, not both.

Some additional partition configuration policies:

DefaultTime	It is possible to assign, through policy, a default time limit for a partition. The advantage would be taking away the need for the end-user to supply a default time limit
GraceTime	This is the default amount of time between SIGTERM and SIGKILL calls when preempting a job in this partition. Once the time limit for the job has been reached, slurmd will issue a SIGTERM to the running application(s). After GraceTime values has been reached, then a SIGKILL is sent. The default time for this is 30 seconds. It is possible for an end-user to catch the SIGTERM and do something about it, like save the state of the application. SIGKILL is not something a user can catch and do anything with, however.
MaxCPUsPerNode	This is useful for nodes that are assigned to multiple partitions. For example, jobs using GPUs on a node can use all of its CPUs, but jobs NOT using the GPUs must leave some CPUs available for jobs in the partition using GPUs

For example, with a cluster with 1 GPU and 4 CPUs per node, configure 2 queues: cpu and gpu. Then configure the cpu queue with MaxCPUsPerNode=3.



Let's say you have a node with a GPU on it. In order to use the GPU, you will also need to use a CPU. You don't want to fill all the CPUs on the node with jobs and leave the GPU idle. So the way you could handle this is that you might configure 2 queues: a queue named "gpu" that only runs GPU jobs, and another named "cpu" that can run CPU and GPU jobs. In this example, you will want to limit the max number of CPUs per job to 3, leaving one of the CPUs available. Anybody who wants to use the GPU, will also have a CPU available at the same time.

You can restrict the gpu queue to jobs using GPUs by using a job_submit plugin

Many sites will assign a very small time limit, and force users to specify a time limit that would be appropriate for their job.

NOTE: If you set the default time limit to the max time limit, then it makes it very difficult for the backfill scheduler to be efficient.

It's worth mentioning that it is possible to evaluate the job at submit time to see whether or not it has a time limit set (or memory request, or CPU count, etc). Using a job submit plugin, you can assign a time limit if not set or force the time limit to fall within an acceptable range.

Other configuration parameters on a partition are (where '#' is an integer value):

MaxNodes=#	The most nodes a job sent to this partition can be allocated by the scheduler
MaxTime=#	The longest time a job sent to this partition can be allocated by the scheduler. A good use case for this would be to create a partition (queue) that limits the time a job can run in. Make another for longer running jobs, etc
MinNodes=#	The minimum number of nodes a job sent to this partition can be allocated by the scheduler
PriorityJobFactor==#	This is used to calculate job priority. Using a Job Priority Factor, jobs that land in this partition will be given a calculated job priority
PriorityTier=#	When jobs are scheduled, partitions in a higher scheduling tier get scheduled first no matter what the job priority is. Partition Priority tier can also be used for job preemption. For example, if you create a "standby" partition, you could define it to be a low Priority Tier value. Then, when "regular" jobs come in, it could requeue the standby jobs. Since the partitions are scheduled independently, this is only useful if you have nodes in multiple partitions.
PreemptMode	This policy determines what happens when a higher-priority job comes in and preemption is enabled. The most common values are: suspend, requeue, and cancel
ReqResv	This policy configures the partition so that only jobs requesting an advanced reservation can be run
RootOnly	This policy dates back to Moab and Maui schedulers. They used to be able to work on Slurm, and use Slurm as a resource manager much like Toque acted under Moab/Maui. That functionality is not available any more. When it was, it was possible to submit jobs to queues that were managed by Moab. This policy, when enabled, prohibited submitting jobs as root user
SelectTypeParameters	With this policy, it is possible to specify different resource allocation units. With this, it's possible to allocate whole nodes, and on others allocate at the core level. A use case would be as follows: Assume you are a site that runs mostly large, highly parallel jobs. And for a debug partition on that system, users can submit jobs to request individual cores for their jobs. But for their really large batch jobs, resources can be allocated at a whole node level so there is no interference with the parallel jobs running on an individual node

OverSubscribe	Mostly needed for gang scheduling. You can decide that you want multiple jobs allocated on each core, for example, and time slice them
State	This policy sets the state of the partition. Valid values are: Up, Down, Drain, and Inactive
TRESBillingWeights	This policy allows you to define billing weights for each TRES. This lets you define, on a per-partition basis, how much you would like to charge for the various resources, such as memory, cpu, and cores. This applies to not only billing in the accounting system, but also for fairshare calculation charges

An example of how you add these policies in the `slurm.conf`:

```
# Partition configuration in slurm.conf

PartitionName=DEFAULT State=UP GraceTime=10 DefMemPerCPU=512 PriorityTier=1

PartitionName=debug Nodes=tux[0-31] MaxTime=30 MaxNodes=4 Default=yes
PartitionName=batch Nodes=tux[32-2000] MaxTime=infinite MaxNodes=infinite
PartitionName=all Nodes=tux[0-2000] MaxTime=30 AllowQos=critical PriorityTier=8
```

In this example, the word "DEFAULT" applies to all partitions.

The "debug" partition is made up of nodes tux0 through tux31, and allows a max job runtime of 30 minutes, allocating no more than 4 nodes.

A "batch" partition made of up nodes tux32 through tux2000 that allows infinite wlttime and does not max out on nodes assigned to jobs.

Another partition called "all" made up of all the compute nodes, with a max time of 30 minutes. Limited to jobs that are submitted to the critical QOS, and a higher Priority Tier so jobs destined for this partition should be scheduled first.

If there is a job in the "all" partition, the backfill scheduler will block the jobs submitted to the debug and batch partitions.

Partition-Based Preemption

Slurm supports job preemption, the act of stopping one or more "low-priority" jobs to let a "high-priority" job run. Job preemption is implemented as a variation of Slurm's Gang Scheduling logic.

When a high-priority job has been allocated resources that have already been allocated to one or more low priority jobs, the low priority job(s) are preempted. The low priority job(s) can resume once the high priority job completes.

Alternatively, the low priority job(s) can be requeued and started using other resources if so configured in newer versions of Slurm.

The job's partition priority or its Quality Of Service (QOS) can be used to identify which jobs can preempt or be preempted by other jobs. Slurm offers the ability to configure the preemption mechanism used on a per partition or per QOS basis. For example, jobs in a low priority queue may get requeued, while jobs in a medium priority queue may get suspended.

Configuration for Preemption

There are several important configuration parameters relating to preemption:

SelectType	Slurm job preemption logic supports nodes allocated by the select/linear plugin, socket/core/CPU resources allocated by the select/cons_res plugin or select/cray_aries for Cray XC systems without ALPS.
SelectTypeParameter	Since resources may be getting over-allocated with jobs (suspended jobs remain in memory), the resource selection plugin should be configured to track the amount of memory used by each job to ensure that memory page swapping does not occur. When select/linear is chosen, we recommend setting SelectTypeParameter=CR_Memory. When select/cons_res is chosen, we recommend including Memory as a resource (e.g. SelectTypeParameter=CR_Core_Memory).
	NOTE: Unless PreemptMode=SUSPEND,GANG these memory management parameters are not critical.
DefMemPerCPU	Since job requests may not explicitly specify a memory requirement, we also recommend configuring DefMemPerCPU (default memory per allocated CPU) or DefMemPerNode (default memory per allocated node). It may also be desirable to configure MaxMemPerCPU (maximum memory per allocated CPU) or MaxMemPerNode (maximum memory per allocated node) in <code>slurm.conf</code> . Users can use the --mem or --mem-per-cpu option at job submission time to specify their memory requirements.
	NOTE: Unless PreemptMode=SUSPEND,GANG these memory management parameters are not critical.
GraceTime	Specifies a time period for a job to execute after it is selected to be preempted. This option can be specified by partition or QOS using the <code>slurm.conf</code> file or database respectively. This option is only honored if PreemptMode=CANCEL. The GraceTime is specified in seconds and the default value is zero, which results in no preemption delay. Once a job has been selected for preemption, its end time is set to the current time plus GraceTime. The job is immediately sent SIGCONT and SIGTERM signals in order to provide notification of its imminent termination. This is followed by the SIGCONT, SIGTERM and SIGKILL signal sequence upon reaching its new end time.
JobAcctGatherType and JobAcctGatherFrequency	The "maximum data segment size" and "maximum virtual memory size" system limits will be configured for each job to ensure that the job does not exceed its requested amount of memory. If you wish to enable additional enforcement of memory limits, configure job accounting with the JobAcctGatherType and JobAcctGatherFrequency parameters. When accounting is enabled and a job exceeds its configured memory limits, it will be canceled in order to prevent it from adversely affecting other jobs sharing the same resources.
	NOTE: Unless PreemptMode=SUSPEND,GANG these memory management parameters are not critical.
PreemptMode	Specifies the mechanism used to preempt low priority jobs.

The PreemptMode should be specified for the cluster as a whole, although different values can be configured on each partition when PreemptType=preempt/partition_prio.

NOTE: When specified on a partition, a compatible mode must also be specified

system-wide; specifically if a PreemptMode is set to SUSPEND for any partition, then the system-wide PreemptMode must include the GANG and SUSPEND parameters so the module responsible for resuming jobs executes. Configure to CANCEL, CHECKPOINT, SUSPEND or REQUEUE depending on the desired action for low priority jobs.

A value of **GANG** must also be specified if gang scheduling is desired or a PreemptMode of SUSPEND is used for any jobs.

A value of **CANCEL** will always cancel the job.

A value of **CHECKPOINT** will checkpoint (if possible) or kill low priority jobs.

Checkpointed jobs are not automatically restarted.

A value of **REQUEUE** will requeue (if possible) or kill low priority jobs.

Requeued jobs are permitted to be restarted on different resources.

A value of **SUSPEND** will suspend and resume jobs.

If PreemptType=preempt/partition_prio is configured then a value of SUSPEND will suspend and automatically resume the low priority jobs. If PreemptType=preempt/qos is configured, then the jobs sharing resources will always time slice rather than one job remaining suspended. The SUSPEND option must be used with the GANG option (e.g. "PreemptMode=SUSPEND,GANG").

A value of **GANG** may be used with any of the above values and will execute a module responsible for resuming jobs previously suspended for either gang scheduling or job preemption with suspension.

PreemptType: Configure the desired mechanism used to identify which jobs can preempt other jobs.

preempt/none Indicates that jobs will not preempt each other (default).

preempt/partition_prio Indicates that jobs from one partition can preempt jobs from lower priority partitions.

preempt/qos Indicates that jobs from one Quality Of Service (QOS) can preempt jobs from a lower QOS. These jobs can be in the same partition or different partitions. PreemptMode (see above) must be set to CANCEL, CHECKPOINT, REQUEUE or SUSPEND. This option requires the use of a database identifying available QOS and their preemption rules. This option is not compatible with PreemptMode=OFF and PreemptMode=SUSPEND is only supported by the select/cons_res plugin.

PreemptExemptTime Specifies minimum run time of jobs before they are considered for preemption. Unlike GraceTime, this is honored for all values of PreemptMode.

It is specified as a time string: A time of -1 disables the option, equivalent to 0.

Acceptable time formats include:

- minutes
- minutes:seconds
- hours:minutes:seconds
- days\hours
- days\hours:minutes
- days\hours:minutes:seconds

PreemptEligibleTime is shown in the output of "scontrol show job <job id>".

PriorityTier Configure the partition's PriorityTier setting relative to other partitions to control the preemptive behavior when PreemptType=preempt/partition_prio. This option is not relevant if PreemptType=preempt/qos. If two jobs from two different partitions are allocated to the same resources, the job in the partition with the greater PriorityTier value will preempt the job in the partition with the lesser PriorityTier value. If the PriorityTier values of the two partitions are equal then no preemption will occur. The default PriorityTier value is 1.

OverSubscribe Configure the partition's OverSubscribe setting to FORCE for all partitions in which job preemption using a suspend/resume mechanism is used or NO otherwise. The FORCE option supports an additional parameter that controls how many jobs can oversubscribe a compute resource (FORCE[:max_share]). By default the max_share value is 4. In order to preempt jobs (and not gang schedule them), always set max_share to 1. To allow up to 2 jobs from this partition to be allocated to a common resource (and gang scheduled), set OverSubscribe=FORCE:2.

NOTE: PreemptType=QOS will permit one additional job to be run on the partition if started due to job preemption. For example, a configuration of OverSubscribe=FORCE:1 will only permit one job per resources normally, but a second job can be started if done so through preemption based upon QOS. The use of PreemptType=QOS and PreemptType=Suspend only applies with SelectType=cons_res in versions 18.08 and older or with SelectType=cons_tres in version 19.05.

To enable preemption after making the configuration changes described above, restart Slurm if it is already running. Any change to the plugin settings in Slurm requires a full restart of the daemons. If you just change the partition PriorityTier or OverSubscribe setting, this can be updated with scontrol reconfig.

If a job request restricts Slurm's ability to run jobs from multiple users or accounts on a node by using the "--exclusive=user" or "--exclusive=mcs" job options, that may prevent preemption of jobs to start higher priority jobs. If preemption is used, it is generally advisable to disable the "--exclusive=user" and "--exclusive=mcs" job options by using a job_submit plugin (set the value of "shared" to "NO_VAL16").

Preemption Design and Operation

The select plugin will identify resources where a pending job can begin execution. When PreemptMode is configured to CANCEL, CHECKPOINT, SUSPEND or REQUEUE, the select plugin will also preempt running jobs as needed to initiate the pending job. When PreemptMode=SUSPEND,GANG the select plugin will initiate the pending job and rely upon the gang scheduling logic to perform job suspend and resume as described below.

The select plugin is passed an ordered list of preemptable jobs to consider for each pending job which is a candidate to start.

This list is sorted by either:

1. QOS priority,
2. Partition priority and job size (to favor preempting smaller jobs), or
3. Job start time (with SchedulerParameters=preempt_youngest_first).

The *select* plugin will determine if the pending job can start without preempting any jobs and if so, starts the job using available resources. Otherwise, the select plugin will simulate the preemption of each job in the priority ordered list and test if the job can be started after each preemption. Once the job can be started, the higher priority jobs in the preemption queue will not be considered, but the jobs to be preempted in the original list may be sub-optimal. For example, to start an 8 node job, the ordered preemption candidates may be 2 node, 4 node and 8 node. Preempting all three jobs would allow the pending job to start, but by reordering the preemption candidates it is possible to start the pending job after preempting only one job. To address this issue, the preemption candidates are re-ordered with the final job requiring preemption placed first in the list and all of the other jobs to be preempted ordered by the number of nodes in their allocation which overlap the resources selected for the pending job. In the example above, the 8 node job would be moved to the first position in the list. The process of simulating the preemption of each job in the priority ordered list will then be repeated for the final decision of which jobs to preempt. This two stage process may preempt jobs which are not strictly in preemption priority order, but fewer jobs will be preempted than otherwise required. See the SchedulerParameters configuration parameter options of preempt_reordered_count and preempt_strict_order for preemption tuning parameters.

When enabled, the gang scheduling logic (which is also supports job preemption) keeps track of the resources allocated to all jobs. For each partition an "active bitmap" is maintained that tracks all concurrently running jobs in the Slurm cluster. Each partition also maintains a job list for that partition, and a list of "shadow" jobs. The "shadow" jobs are high priority job allocations that "cast shadows" on the active bitmaps of the low priority jobs. Jobs caught in these "shadows" will be preempted.

Each time a new job is allocated to resources in a partition and begins running, the gang scheduler adds a "shadow" of this job to all lower priority partitions. The active bitmap of these lower priority partitions are then rebuilt, with the shadow jobs added first. Any existing jobs that were replaced by one or more "shadow" jobs are suspended (preempted). Conversely, when a high priority running job completes, its "shadow" goes away and the active bitmaps of the lower priority partitions are rebuilt to see if any suspended jobs can be resumed.

The gang scheduler plugin is designed to be reactive to the resource allocation decisions made by the "select" plugins. The "select" plugins have been enhanced to recognize when job preemption has been configured, and to factor in the priority of each partition when selecting resources for a job. When choosing resources for each job, the selector avoids resources that are in use by other jobs (unless sharing has been configured, in which case it does some load-balancing). However, when job preemption is enabled, the select plugins may choose resources

that are already in use by jobs from partitions with a lower priority setting, even when sharing is disabled in those partitions.

This leaves the gang scheduler in charge of controlling which jobs should run on the over-allocated resources. If PreemptMode=SUSPEND, jobs are suspended using the same internal functions that support scontrol suspend and scontrol resume. A good way to observe the operation of the gang scheduler is by running squeue -i<time> in a terminal window.

Limitations of Preemption During Backfill Scheduling

For performance reasons, **the backfill scheduler reserves whole nodes for jobs, not partial nodes**. If during backfill scheduling a job preempts one or more other jobs, the whole nodes for those preempted jobs are reserved for the preemptor job, even if the preemptor job requested fewer resources than that. These reserved nodes aren't available to other jobs during that backfill cycle, even if the other jobs could fit on the nodes. Therefore, jobs may preempt more resources during a single backfill iteration than they requested.

Slurm Dynamic Nodes

Starting in Slurm 22.05, nodes can be dynamically added and removed from Slurm.

Dynamic Node Communications

For regular, non-dynamically created nodes, Slurm knows how to communicate with nodes by reading in the `slurm.conf`. This is why it is important for a non-dynamic setup that the `slurm.conf` is synchronized across the cluster. For dynamically created nodes, other than the `slurmctld`, the rest of the Slurm components (e.g. `srun`, daemons) don't know about the dynamically created nodes. In order for `srun` and the `slurmds` to know how to communicate with the other nodes in a job allocation, `slurmctld` passes each node's address information (`NodeName`, `NodeAddr`, `NodeHostname`) -- known as the alias list -- to the `srun` and the `srun` forwards the list to the `slurmds`. This list is seen in the job's environment as the `SLURM_NODE_ALIASES` environment variable.

The controller automatically grabs the node's `NodeAddr` and `NodeHostname` for dynamic `slurmd` registrations. For cloud nodes created with `scontrol`, if the nodename is not resolvable, then either 1) the node's `NodeAddr` and `NodeHostname` need to be updated with the `scontrol update` command before the node registers or 2) use the `cloud_reg_addrs` `SlurmctldParameter`.

Slurm Configuration Related to Dynamic Nodes

`MaxNodeCount=#` Set to the number of possible nodes that can be active in a system at a time.
See the `slurm.conf` man page for more details.

`SelectType=select/cons_tres` Dynamic nodes are only supported with `cons_tres`.

`TreeWidth=65533` Fanning out of controller pings and application launches through `slurmds` are not supported with dynamic nodes. `TreeWidth` must be disabled (i.e. set to 65533) for dynamic environments. However, the reverse fanout of step completions through `slurmds` does happen due to the job's alias list.

NOTE: The `cloud_dns` `SlurmctldParameter` must not be set as this disables the alias list.

PARTITION ASSIGNMENT for Dynamic Nodes

Dynamic nodes can be automatically assigned to partitions at creation by using the partition's nodes *ALL* keyword or *NodeSets* and specifying a feature on the nodes.

e.g. in *slurm.conf*

```
Nodeset=ns1 Feature=f1
Nodeset=ns2 Feature=f2

PartitionName=all Nodes=ALL Default=yes
PartitionName=dyn1 Nodes=ns1
PartitionName=dyn2 Nodes=ns2
PartitionName=dyn3 Nodes=ns1,ns2
```

Creating Nodes

Nodes can be created two ways:

1. Dynamic slurmd registration

Using the slurmd -Z and --conf options a slurmd will register with the controller and will automatically be added to the system.

e.g.

```
slurmd -Z --conf "RealMemory=80000 Gres=gpu:2 Feature=f1"
```

2. scontrol create NodeName= ...

Create nodes using scontrol by specifying the same NodeName line that you would define in the *slurm.conf*. See *slurm.conf* man page for node options. Only State=CLOUD and State=FUTURE are supported. The node configuration should match what the slurmd will register with (e.g. slurmd -C) plus any additional attributes.

e.g.

```
scontrol create NodeName=d[1-100] CPUs=16 Boards=1 SocketsPerBoard=1 CoresPerSocket=8
ThreadsPerCore=2 RealMemory=31848 Gres=gpu:2 Feature=f1 State=cloud
```

Deleting Nodes

Nodes can be deleted using **scontrol delete nodename=<nodelist>**.

Nodes can only be deleted if they have no jobs running on them and aren't part of a reservation.

Node and Partition Exercises

In this set of labs, you will learn how to modify nodes and partitions.

Exercise 1: Group nodes into partitions

In this exercise, you will create 2 additional partitions called "batch" and "all." the debug partition will contain nodes node00-node04, the batch partition will contain nodes node05-node09, and the "all" partition will contain all the nodes in the cluster. Each partition will also have a different priority. With different priorities, jobs will land on a higher priority destination partition.

1. As root user, edit `/etc/slurm/slurm.conf` and configure the partition entries to look like this:

```
PartitionName=DEFAULT Nodes=node[00-09]
PartitionName=debug Nodes=node[00-04] Default=YES MaxTime=30:00 State=UP PriorityTier=1
PartitionName=batch Nodes=node[05-09] Default=NO MaxTime=30:00 State=UP PriorityTier=2
PartitionName=all Nodes=node[00-09] Default=NO MaxTime=10:00 State=UP PriorityTier=3

#NodeName=cloud[0000-1024] Weight=8 Feature=cloud State=CLOUD
#PartitionName=cloud Nodes=cloud[0000-1024] Default=no
```

Note: Comment out the cloud partition and nodes entries. Also check the MaxTime values

2. Restart the controller:
`/lab_scripts/restart.sh`
3. Verify that the scheduler knows about the new partitions:
`scontrol show partitions`

It should show:

```

PartitionName=debug
  AllowGroups=ALL AllowAccounts=ALL AllowQos=ALL
  AllocNodes=ALL Default=YES QoS=N/A
  DefaultTime=NONE DisableRootJobs=NO ExclusiveUser=NO GraceTime=0 Hidden=NO
  MaxNodes=UNLIMITED MaxTime=30:00 MinNodes=0 LLN=NO MaxCPUsPerNode=UNLIMITED
  MaxCPUsPerSocket=UNLIMITED
  Nodes=node[00-04]
  PriorityJobFactor=1 PriorityTier=1 RootOnly=NO ReqResv=NO OverSubscribe=NO
  OverTimeLimit=NONE PreemptMode=OFF
  State=UP TotalCPUs=20 TotalNodes=5 SelectTypeParameters=NONE
  JobDefaults=(null)
  DefMemPerNode=UNLIMITED MaxMemPerNode=UNLIMITED
  TRES(cpu=20,mem=79835M,node=5,billing=20,gres/gpu=15,gres/gpu:gtx=15
  ResumeTimeout=GLOBAL SuspendTimeout=GLOBAL SuspendTime=GLOBAL PowerDownOnIdle=NO

PartitionName=batch
  AllowGroups=ALL AllowAccounts=ALL AllowQos=ALL
  AllocNodes=ALL Default=NO QoS=N/A
  DefaultTime=NONE DisableRootJobs=NO ExclusiveUser=NO GraceTime=0 Hidden=NO
  MaxNodes=UNLIMITED MaxTime=30:00 MinNodes=0 LLN=NO MaxCPUsPerNode=UNLIMITED
  MaxCPUsPerSocket=UNLIMITED
  Nodes=node[05-09]
  PriorityJobFactor=1 PriorityTier=2 RootOnly=NO ReqResv=NO OverSubscribe=NO
  OverTimeLimit=NONE PreemptMode=OFF
  State=UP TotalCPUs=20 TotalNodes=5 SelectTypeParameters=NONE
  JobDefaults=(null)
  DefMemPerNode=UNLIMITED MaxMemPerNode=UNLIMITED
  TRES(cpu=20,mem=79835M,node=5,billing=20,gres/gpu=15,gres/gpu:gtx=15
  ResumeTimeout=GLOBAL SuspendTimeout=GLOBAL SuspendTime=GLOBAL PowerDownOnIdle=NO

PartitionName=all
  AllowGroups=ALL AllowAccounts=ALL AllowQos=ALL
  AllocNodes=ALL Default=NO QoS=N/A
  DefaultTime=NONE DisableRootJobs=NO ExclusiveUser=NO GraceTime=0 Hidden=NO
  MaxNodes=UNLIMITED MaxTime=10:00 MinNodes=0 LLN=NO MaxCPUsPerNode=UNLIMITED
  MaxCPUsPerSocket=UNLIMITED
  Nodes=node[00-09]
  PriorityJobFactor=1 PriorityTier=3 RootOnly=NO ReqResv=NO OverSubscribe=NO
  OverTimeLimit=NONE PreemptMode=OFF
  State=UP TotalCPUs=40 TotalNodes=10 SelectTypeParameters=NONE
  JobDefaults=(null)
  DefMemPerNode=UNLIMITED MaxMemPerNode=UNLIMITED
  TRES(cpu=40,mem=159670M,node=10,billing=40,gres/gpu=30,gres/gpu:gtx=30
  ResumeTimeout=GLOBAL SuspendTimeout=GLOBAL SuspendTime=GLOBAL PowerDownOnIdle=NO

```

4. As fred, submit a job to the cluster:

```
srun --mem=1000 -N5 hostname
```

Should show:

```

node02
node04
node03
node01
node00

```

Note: These nodes are part of the default partition of debug

5. Submit another job to the cluster but to the batch partition:

```
srun --mem=1000 -N5 -pbatch hostname
```

Should show:

```
node07  
node09  
node08  
node06  
node05
```

Note: These nodes are part of the batch partition

6. Submit another job to the cluster but to the either the batch or debug partition:

```
srun --mem=1000 -N5 -pbatch,debug hostname
```

Should show:

```
node07  
node09  
node08  
node06  
node05
```

Note: The reason this landed in the batch partition is because it's a higher priority partition (PriorityTier=2 vs 1)

Cleanup

1. Reset the Docker Slurm cluster environment back to its original, pristine state by exiting the login Docker container back to the Ubuntu AWS prompt. And **as ubuntu**:

```
cd ~/docker-scale-out  
make clean && make
```

Exercise 2: Configure Slurm for Partition-Based Preemption

In this exercise, you will create 3 partitions: **debug**, **hiprio**, and **lowprio**. All nodes will belong to the debug partition. Nodes node00-node04 will belong to the hiprio partition. And Nodes node05-09 will belong to the lowprio partition. Each will have a different priority tier. By configuring Slurm for partition-based preemption, jobs sent to the debug (default) partition will be preempted by the hiprio-destined nodes.

1. **As root**, add (or change) the policies in **/etc/slurm/slurm.conf** as follows:

```
PreemptType=preempt/partition_prio  
PreemptMode=REQUEUE  
PreemptExemptTime=0:5
```

2. Change the partition setup in **/etc/slurm/slurm.conf** to reflect the following partition settings:

```
PartitionName=debug Nodes=node[00-09] Default=YES MaxTime=INFINITE State=UP PriorityTier=1  
PartitionName=lowprio Nodes=node[00-04] MaxTime=INFINITE State=UP PriorityTier=2  
PartitionName=hiprio Nodes=node[05-09] MaxTime=INFINITE State=UP PriorityTier=3
```

3. Restart the scheduler:

```
/lab_scripts/restart.sh
```

4. Confirm the partition assignments by typing:

```
sinfo -pdebug,lowprio,hiprio -Nel
```

It should show:

NODELIST	NODES	PARTITION	STATE	CPUS	S:C:T	MEMORY	TMP_DISK	WEIGHT	AVAIL_FE	REASON
node00	1	debug*	idle	4	1:4:1	16011	0	1	(null)	none
node00	1	lowprio	idle	4	1:4:1	16011	0	1	(null)	none
node01	1	debug*	idle	4	1:4:1	16011	0	1	(null)	none
node01	1	lowprio	idle	4	1:4:1	16011	0	1	(null)	none
node02	1	debug*	idle	4	1:4:1	16011	0	1	(null)	none
node02	1	lowprio	idle	4	1:4:1	16011	0	1	(null)	none
node03	1	debug*	idle	4	1:4:1	16011	0	1	(null)	none
node03	1	lowprio	idle	4	1:4:1	16011	0	1	(null)	none
node04	1	hiprio	idle	4	1:4:1	16011	0	1	(null)	none
node04	1	lowprio	idle	4	1:4:1	16011	0	1	(null)	none
node05	1	debug*	idle	4	1:4:1	16011	0	1	(null)	none
node05	1	hiprio	idle	4	1:4:1	16011	0	1	(null)	none
node06	1	debug*	idle	4	1:4:1	16011	0	1	(null)	none
node06	1	hiprio	idle	4	1:4:1	16011	0	1	(null)	none
node07	1	debug*	idle	4	1:4:1	16011	0	1	(null)	none
node07	1	hiprio	idle	4	1:4:1	16011	0	1	(null)	none
node08	1	debug*	idle	4	1:4:1	16011	0	1	(null)	none
node08	1	hiprio	idle	4	1:4:1	16011	0	1	(null)	none
node09	1	debug*	idle	4	1:4:1	16011	0	1	(null)	none
node09	1	hiprio	idle	4	1:4:1	16011	0	1	(null)	none

(Sorry for the small font)

5. Open a second terminal.

6. Run squeue and sinfo together in a watch command:

```
watch -t "squeue -la ; sinfo -pdebug,lowprio,hiprio -Nel"
```

It should show:

Thu Feb 20 22:27:16 2020										
JOBID	PARTITION	NAME	USER	STATE	TIME	TIME_LIMI	NODES	NODELIST(REASON)		
NODELIST	NODES	PARTITION	STATE	CPUS	S:C:T	MEMORY	TMP_DISK	WEIGHT	AVAIL_FE	REASON
node00	1	debug*	idle	4	1:4:1	16011	0	1	(null)	none
node00	1	lowprio	idle	4	1:4:1	16011	0	1	(null)	none
node01	1	debug*	idle	4	1:4:1	16011	0	1	(null)	none
node01	1	lowprio	idle	4	1:4:1	16011	0	1	(null)	none
node02	1	debug*	idle	4	1:4:1	16011	0	1	(null)	none
node02	1	lowprio	idle	4	1:4:1	16011	0	1	(null)	none
node03	1	debug*	idle	4	1:4:1	16011	0	1	(null)	none
node03	1	lowprio	idle	4	1:4:1	16011	0	1	(null)	none
node04	1	hiprio	idle	4	1:4:1	16011	0	1	(null)	none
node04	1	lowprio	idle	4	1:4:1	16011	0	1	(null)	none
node05	1	debug*	idle	4	1:4:1	16011	0	1	(null)	none
node05	1	hiprio	idle	4	1:4:1	16011	0	1	(null)	none
node06	1	debug*	idle	4	1:4:1	16011	0	1	(null)	none
node06	1	hiprio	idle	4	1:4:1	16011	0	1	(null)	none
node07	1	debug*	idle	4	1:4:1	16011	0	1	(null)	none
node07	1	hiprio	idle	4	1:4:1	16011	0	1	(null)	none
node08	1	debug*	idle	4	1:4:1	16011	0	1	(null)	none
node08	1	hiprio	idle	4	1:4:1	16011	0	1	(null)	none
node09	1	debug*	idle	4	1:4:1	16011	0	1	(null)	none
node09	1	hiprio	idle	4	1:4:1	16011	0	1	(null)	none

7. As fred, submit a job requiring 4 nodes:

```
sbatch --mem=1000 --exclusive -N4 -t5 --wrap="sleep 300"
```

In the watch window, you should see:

Fri Jan 22 21:46:19 2021											
	JOBID	PARTITION	NAME	USER	STATE	TIME	TIME_LIMI	NODES	NODELIST(REASON)		
	1	debug	wrap	fred	RUNNING	0:20	5:00	4	node[00-03]		
Fri Jan 22 21:46:19 2021											
NODELIST	NODES	PARTITION		STATE	CPUS	S:C:T	MEMORY	TMP_DISK	WEIGHT	AVAIL_FE	REASON
node00	1	debug*	allocated	4	1:4:1	16011	0	1	(null)	none	
node00	1	lowprio	allocated	4	1:4:1	16011	0	1	(null)	none	
node01	1	debug*	allocated	4	1:4:1	16011	0	1	(null)	none	
node01	1	lowprio	allocated	4	1:4:1	16011	0	1	(null)	none	
node02	1	debug*	allocated	4	1:4:1	16011	0	1	(null)	none	
node02	1	lowprio	allocated	4	1:4:1	16011	0	1	(null)	none	
node03	1	debug*	allocated	4	1:4:1	16011	0	1	(null)	none	
node03	1	lowprio	allocated	4	1:4:1	16011	0	1	(null)	none	
node04	1	debug*	idle	4	1:4:1	16011	0	1	(null)	none	
node04	1	lowprio	idle	4	1:4:1	16011	0	1	(null)	none	
node05	1	debug*	idle	4	1:4:1	16011	0	1	(null)	none	
node05	1	hiprio	idle	4	1:4:1	16011	0	1	(null)	none	
node06	1	debug*	idle	4	1:4:1	16011	0	1	(null)	none	
node06	1	hiprio	idle	4	1:4:1	16011	0	1	(null)	none	
node07	1	debug*	idle	4	1:4:1	16011	0	1	(null)	none	
node07	1	hiprio	idle	4	1:4:1	16011	0	1	(null)	none	
node08	1	debug*	idle	4	1:4:1	16011	0	1	(null)	none	
node08	1	hiprio	idle	4	1:4:1	16011	0	1	(null)	none	
node09	1	debug*	idle	4	1:4:1	16011	0	1	(null)	none	
node09	1	hiprio	idle	4	1:4:1	16011	0	1	(null)	none	

8. Send another job:

```
sbatch --mem=1000 --exclusive -N4 -t35 --wrap="sleep 300"
```

It should now show:

Fri Jan 22 21:48:03 2021										
JOBID	PARTITION	NAME	USER	STATE	TIME	TIME_LIMI	NODES	NODELIST(REASON)		
1	debug	wrap	fred	RUNNING	2:04	5:00	4	node[00-03]		
2	debug	wrap	fred	RUNNING	0:11	35:00	4	node[04-07]		
Fri Jan 22 21:48:03 2021										
NODELIST	NODES	PARTITION	STATE	CPUS	S:C:T	MEMORY	TMP_DISK	WEIGHT	AVAIL_FE	REASON
node00	1	debug*	allocated	4	1:4:1	16011	0	1	(null)	none
node00	1	lowprio	allocated	4	1:4:1	16011	0	1	(null)	none
node01	1	debug*	allocated	4	1:4:1	16011	0	1	(null)	none
node01	1	lowprio	allocated	4	1:4:1	16011	0	1	(null)	none
node02	1	debug*	allocated	4	1:4:1	16011	0	1	(null)	none
node02	1	lowprio	allocated	4	1:4:1	16011	0	1	(null)	none
node03	1	debug*	allocated	4	1:4:1	16011	0	1	(null)	none
node03	1	lowprio	allocated	4	1:4:1	16011	0	1	(null)	none
node04	1	debug*	allocated	4	1:4:1	16011	0	1	(null)	none
node04	1	lowprio	allocated	4	1:4:1	16011	0	1	(null)	none
node05	1	debug*	allocated	4	1:4:1	16011	0	1	(null)	none
node05	1	hiprio	allocated	4	1:4:1	16011	0	1	(null)	none
node06	1	debug*	allocated	4	1:4:1	16011	0	1	(null)	none
node06	1	hiprio	allocated	4	1:4:1	16011	0	1	(null)	none
node07	1	debug*	allocated	4	1:4:1	16011	0	1	(null)	none
node07	1	hiprio	allocated	4	1:4:1	16011	0	1	(null)	none
node08	1	debug*	idle	4	1:4:1	16011	0	1	(null)	none
node08	1	hiprio	idle	4	1:4:1	16011	0	1	(null)	none
node09	1	debug*	idle	4	1:4:1	16011	0	1	(null)	none
node09	1	hiprio	idle	4	1:4:1	16011	0	1	(null)	none

9. Now, submit a job to the hiprio partition:

```
sbatch --mem=1000 --exclusive -p hiprio -N4 -t5 --wrap="sleep 300"
```

You should see job 4 preempt a job in the lower-priority debug partition:

Fri Jan 22 21:50:01 2021										
JOBID	PARTITION	NAME	USER	STATE	TIME	TIME_LIMI	NODES	NODELIST(REASON)		
1	debug	wrap	fred	PENDING	0:00	35:00	4	(BeginTime)		
2	debug	wrap	fred	RUNNING	4:02	5:00	4	node[00-03]		
3	hiprio	wrap	fred	RUNNING	0:15.	5:00.	4	node[05-08]		
Fri Jan 22 21:50:01 2021										
NODELIST	NODES	PARTITION	STATE	CPUS	S:C:T	MEMORY	TMP_DISK	WEIGHT	AVAIL_FE	REASON
node00	1	debug*	allocated	4	1:4:1	16011	0	1	(null)	none
node00	1	lowprio	allocated	4	1:4:1	16011	0	1	(null)	none
node01	1	debug*	allocated	4	1:4:1	16011	0	1	(null)	none
node01	1	lowprio	allocated	4	1:4:1	16011	0	1	(null)	none
node02	1	debug*	allocated	4	1:4:1	16011	0	1	(null)	none
node02	1	lowprio	allocated	4	1:4:1	16011	0	1	(null)	none
node03	1	debug*	allocated	4	1:4:1	16011	0	1	(null)	none
node03	1	lowprio	allocated	4	1:4:1	16011	0	1	(null)	none
node04	1	debug*	idle	4	1:4:1	16011	0	1	(null)	none
node04	1	lowprio	idle	4	1:4:1	16011	0	1	(null)	none
node05	1	debug*	allocated	4	1:4:1	16011	0	1	(null)	none
node05	1	hiprio	allocated	4	1:4:1	16011	0	1	(null)	none
node06	1	debug*	allocated	4	1:4:1	16011	0	1	(null)	none
node06	1	hiprio	allocated	4	1:4:1	16011	0	1	(null)	none
node07	1	debug*	allocated	4	1:4:1	16011	0	1	(null)	none
node07	1	hiprio	allocated	4	1:4:1	16011	0	1	(null)	none
node08	1	debug*	allocated	4	1:4:1	16011	0	1	(null)	none
node08	1	hiprio	allocated	4	1:4:1	16011	0	1	(null)	none
node09	1	debug*	idle	4	1:4:1	16011	0	1	(null)	none
node09	1	hiprio	idle	4	1:4:1	16011	0	1	(null)	none

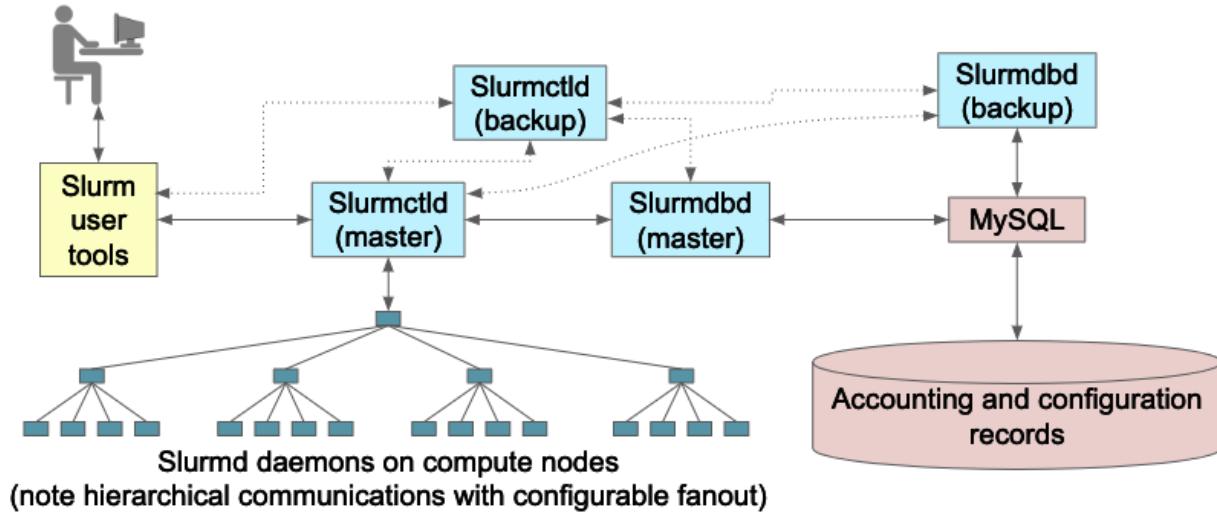
Cleanup

1. Reset the Docker Slurm cluster environment back to its original, pristine state by exiting the login Docker container back to the Ubuntu AWS prompt. And **as ubuntu**:

```
cd ~/docker-scale-out  
make clean && make
```

Daemons and Logging Discussion

The following diagram lists the common daemons that you will find running on a typical Slurm deployment:



- **Slurm user tools:** These are the user-oriented commands from which jobs are launched and tracked
- **slurmctld (master):** This is the master Slurm control daemon. In an HA configuration you may have a secondary Slurm control daemon running on other hardware
- **slurmdbd (master):** This is the master Slurm database daemon. If configured and running, it provides access to a database which can do job accounting, enforcement, and other tracking
- **MySQL:** The database that tracks job accounting, enforcement, and other activities
- **slurmd:** The Slurm daemons which run on the compute nodes. They provide information to the slurm control daemon on the configuration and utilization of the compute resources

Daemons

The documentation for the daemons listed in the above graphic are all provided in man pages. When starting the daemons, there are switches that can be used to control the startup/shutdown of them. These are:

- **-D** Run the daemon in foreground, with output sent to the terminal. If you are debugging the daemons and trying to see why they are/or are not starting
- **-v** Verbose logging. The daemons support multiple v's like this: -vvv
- **-c** Cold start the daemon. This is used to ignore the previously saved state of the daemon. This throws away the old state. When starting this way, it will lose track of all the previously-running jobs, reservations, and node state information and start clean. Use this switch carefully. Only really used when making code changes to the daemons
- **-h** Prints a summary of the daemon startup options

slurmctld Daemon

This is where all the current node, partition, job, and step data is maintained. It decides when and where to initiate jobs. With the exception of the accounting records, this daemon processes all of the user-fed commands. It is highly multi-threaded. If you look at it on a busy cluster, it could have several hundred threads active at any point in time. And, that would not be unusual. It typically runs as a special user, which is usually the "slurm" user that is created during installation time. Most sites do NOT need to run as user "root" for security reasons. Normally, there will only be one slurmctld process running at a given site. Unless you are doing HA, then you might have another running on another host.

slurmdbd Daemon

This is the Slurm Database Daemon. This works very similar to the "Gold" package that was developed in Moab/Maui. One of the major differences is that slurmdbd is a thousand times faster. Large volumes of tracking data would cause Gold to bring down the performance of the system. This daemon sits between the user and the database. Which, by the way, we do NOT recommend editing by hand. Even if you ARE a seasoned MySQL programmer.

Users in a Slurm cluster do not have direct access to the database, so the slurmdbd is how the commands are relayed to the database. In this way, users don't need to worry about remembering a database password and the fields that are relevant.

The slurmdbd is responsible for storing accounting information to the database. It securely returns accounting information, including things like access control permissions. It also stores information about your job, such as when it was launched, when it ended, how long it ran, etc. If there were any job steps that took place as part of the job run, then those are also tracked. It also tracks information about nodes: When they were brought on-line, offline, etc. Also, information about reservations start, end, utilized, etc. are stored in the database.

From a Slurm architecture perspective, if the slurmdbd is down or un-responsive, and the slurmctld is running, then jobs can still continue to be submitted and run, just not tracked in the database. It references the job information from its own cache if the slurmdbd is not running.

If the slurmdbd is down, then changes made to access control, job updates or changes, etc cannot be made because those are stored in the database.

The slurmdbd service does not need to be run as root user. Just like the slurmctld, it normally runs as the Slurm user.

slurmd

This is the service daemon that runs on the compute nodes. It is responsible for launching jobs. It pretty much gets out of the way once the job is running, other than minor updates to the accounting records about the job. It is very small and lightweight piece of code.

The slurmd supports hierarchical communications with configuration fanout, so as to quickly launch parallel jobs on the cluster. This daemon must be run as the "root" user in order to spawn jobs as various users on the system.

slurmd can be started with the -C option which will print actual hardware configuration and then exits. This is helpful when filling in node definitions in the slurm.conf. For example:

The slurm daemon writes its own log file. The location of the log files is listed in the slurm.conf file:

```
$ sbin/slurmd -C
NodeName=node1 CPUs=8 Boards=1 SocketsPerBoard=8 CoresPerSocket=1 ThreadsPerCore=1
RealMemory=3771
UpTime=0-00:44:28
```

slurmstepd

This is what is considered the "Job step shepherd." If you have a job submit script that launches seven srun's, then it launches them in the correct order. Each stepd that runs is given a jobidstep number, which is appended to the end of the jobid.

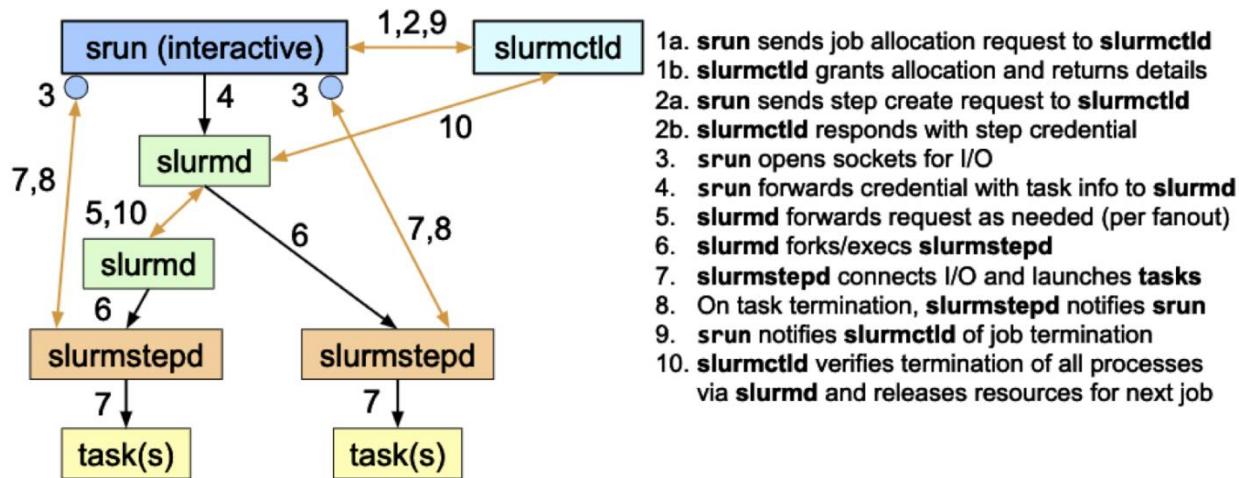
If you are submitting a batch MPI job, then the slurmstepd(s) will launch the application(s).

This starts as user root, then very quickly becomes owned by the job submitters' UID.

It manages the I/O required for the job, as well as any signal handling. This daemon is very compact as well, just as the slurmd is.

Linux Job Launch Sequence

The following chart outlines the job submit sequence:



1a: When submitting an srun, not from within a resource allocation, srun sends an allocation request to the slurmctld. 1b: The slurmctld then responds with what you have been allocated.

2a: If you are running within a batch script, or an interactive salloc request, then srun sends a step create request to the slurmctld. 2b: The slurmctld response with a credential with the allocated resources.

3: The srun then opens a socket(s) for I/O and 4: forwards the job credential to one of the slurmd's in the job allocation

5: The slurmd(s) will forward the request to additions slurmd as needed.

The hierarchical distribution is not like a fixed tree format. For example, node123 doesn't always talk to node124 (next in sequence). The tree that's used to launch a job depends on how the job was submitted. If a 100-

node jobs is submitted and they are spread throughout the cluster, that tree is going to be responsible for launching that job step on the 100 nodes, and not interfere with other running jobs on those nodes. The hierarchical communication is basically the hierarchy as determined on a per-request basis. If there is a failure in the communication, the message will surround that failure. For example, if there is a failure, such as in a job cancel or something like that, the slurmd above that will notice the failure, resend the message to lower slurmd's in the hierarchy. So it's fault tolerance and self-healing communication.

6: The slurmds will launch a slurmstepd for that job step.

7: That slurmstepd will connect over the I/O back to the srun command and launch the task.

8: At task termination, the slurmstepd notifies the srun that initiated the task.

9: srun then tells the slurmctld of job termination

10: slurmctld verifies termination of all processes thorough slurmd and releases resources for the next job. This is the first point where the slurmctld talks directly to the slurmds on the compute nodes to release resources, run the epilog, and make sure the node is cleaned up.

Log Files

In Slurm, each of the daemons writes its own log file. The location of the log files is listed in the *slurm.conf* file:

In *slurm.conf*:

```
SlurmctldLogFile=/var/log/slurmctld.log  
SlurmdLogFile=/var/log/slurmd.log
```

In *slurmdbd.conf*:

```
LogFile=/var/log/slurm/slurmdbd.log
```

How much information goes into the log file depends on the verbosity of the configuration.

We recommend writing the slurmd log files on the local node, for performance reasons. Also, write them using the %n syntax which references the node itself.

The Slurm daemons are only writing to these log files, so it doesn't have any logic built in for maintenance of the logs. So we also recommend using the logrotate tools to compress and save older log files. This will move the log file to a new one, and then send a SIGHUP to the relevant daemon which will start a new log file. Then it compresses what you just did

For troubleshooting purposes, like a user questions what happened to the job, you can search for the jobid or nodename in the logs to see what happened

Setting the log level for the scheduler (slurmctld) is done in the *slurm.conf*:

```
SlurmctldDebug=info
```

The values for SlurmctldDebug are:

- **quiet** Log nothing

- **fatal** Log only fatal errors
- **error** Log only errors
- **info** Log errors and general informational messages (The default)
- **verbose** Log errors and verbose informational messages
- **debug** Log errors and verbose informational messages and debugging messages
- **debug2** Log errors and verbose informational messages and more debugging messages
- **debug3** Log errors and verbose informational messages and even more debugging messages
- **debug4** Log errors and verbose informational messages and even more debugging messages
- **debug5** Log errors and verbose informational messages and even more debugging messages

Setting the log level for the slurm daemons (slurmd) is done in the slurm.conf:

```
SlurmdDebug=info
```

The values for SlurmdDbug are:

- **quiet** Log nothing
- **fatal** Log only fatal errors
- **error** Log only errors
- **info** Log errors and general informational messages (The default)
- **verbose** Log errors and verbose informational messages
- **debug** Log errors and verbose informational messages and debugging messages
- **debug2** Log errors and verbose informational messages and more debugging messages
- **debug3** Log errors and verbose informational messages and even more debugging messages
- **debug4** Log errors and verbose informational messages and even more debugging messages
- **debug5** Log errors and verbose informational messages and even more debugging messages

Daemons and Logging Exercises

In this set of labs, you will learn how to change logging level, and monitor the results of logging afterwards

Exercise 1: Change the control daemon logging level

1. As root, edit the `/etc/slurm/slurm.conf` file. Change whatever value the following entry is to:

```
SlurmctldDebug=info
```

2. Restart the controller:

```
scontrol reconfigure
```

NOTE: “scontrol reconfigure” will re-read the controller/slurmd configuration. But, if there is a change to a plugin, it won’t pick that change up. So if you are making changes to plugins (adding/modifying/removing) you will have to run “/lab_scripts/restart shutdown”

3. Submit a job and watch the queue as fred as follows (**the following lines should be entered as all one command ... notice the semi-colon (;) at the end**):

```
sbatch --mem=1000 --exclusive -N10 -t10 -pdebug --wrap="sleep 1000" ;  
watch squeue
```

4. Note the **JOBID** _____ (most likely it will be JobID=1)

5. Break out of watch with **Ctrl-C**

6. Look for occurrences of the **JOBID** in the controller log file by connecting to the mgmtnode container:

```
ssh mgmtnode
```

```
grep -E "JobId=1[:\: ]" /var/log/slurmctld.log #for me it's JobId=1
```

You should see:

```
[2021-09-30T14:46:36.403] _slurm_rpc_submit_batch_job: JobId=1 InitPrio=4294901759 usec=696  
[2021-09-30T14:46:36.542] sched: Allocate JobId=1 NodeList=node[00-09] #CPUs=40 Partition=debug
```

7. As root again, edit the `/etc/slurm/slurm.conf` file. Change info to debug5:

```
SlurmctldDebug=debug5
```

8. Restart the controller:

```
scontrol reconfig
```

9. Submit another job as fred as follows (**again, this is a single command ... it is in fred's history, as well, so you should be able to just arrow up to it**):

```
sbatch --mem=1000 --exclusive -N10 -t10 -pdebug --wrap="sleep 1000" ;  
watch squeue
```

10. Note the **JOBID** _____
11. Break out of watch with **Ctrl-C**.
12. Look for occurrences of the **JOBID** in the controller log file by connecting to the mgmtnode container:
`ssh mgmtnode`
`grep -E "JobId=2[:\]" /var/log/slurmctld.log` #for me it's JobId=2

Now, you should see something like this:

```
2021-09-30T14:49:28.509] debug2: _build_node_list: JobId=2 matched 0 nodes (cloud[0000-1024])  
due to job partition or features  
[2021-09-30T14:49:28.509] debug3: _pick_best_nodes: JobId=2 idle_nodes 1025 share_nodes 1025  
[2021-09-30T14:49:28.509] _slurm_rpc_submit_batch_job: JobId=2 InitPrio=4294901758 usec=739  
[2021-09-30T14:49:28.872] debug2: _build_node_list: JobId=2 matched 0 nodes (cloud[0000-1024])  
due to job partition or features  
[2021-09-30T14:49:28.872] debug3: _pick_best_nodes: JobId=2 idle_nodes 1025 share_nodes 1025
```

Cleanup

1. Reset the Docker Slurm cluster environment back to its original, pristine state by exiting the login Docker container back to the Ubuntu AWS prompt. And **as ubuntu**:

```
cd ~/docker-scale-out  
make clean && make
```

Slurm Scheduling Discussion

Slurm is designed to perform a quick and simple scheduling attempt at events such as job submission or completion and configuration changes. During these event-triggered scheduling events, `default_queue_depth` (default is 100) number of jobs will be considered.

The interval for quick scheduling attempts to start at the following conditions:

- At each job submission
- At job completion on each of its compute nodes
- At configuration changes

The advantage is that jobs don't need to wait around for the more comprehensive scheduling iteration when requests come into the scheduler which can be accommodated. Jobs can therefore be scheduled quickly.

At less frequent intervals, defined by `sched_interval`, all jobs will be considered for scheduling.

In either case, once any job or job array task in a partition is left pending, no other jobs in that partition will be scheduled.

A more comprehensive scheduling attempt is done by the backfill scheduling plugin. This would be the case if you have thousands of jobs in the queue and they need to re-ordered and sorted, it takes longer to accomplish this scheduling pass. This involves a lot of overhead to determine when and where that many jobs will be scheduled. It's not milliseconds, like the short scheduling pass, but up to many seconds or minutes in the thorough scheduling pass. The heavy part of scheduling is attempting to figure out the space and time for the jobs in the queue.

Scheduling Configuration

The `SchedulerType` configuration parameter specifies the scheduler plugin to use. Options are `sched/backfill`, which performs backfill scheduling, and `sched/builtin`, which attempts to schedule jobs in a strict priority order within each partition/queue.

There is also a `SchedulerParameters` configuration parameter which can specify a wide range of parameters as described below. This first set of parameters applies to all scheduling configurations. See the `slurm.conf(5)` man page for more details.

- `default_queue_depth=#` - Specifies the number of jobs to consider for scheduling on each event that may result in a job being scheduled. Default value is 100 jobs. Since this happens frequently, a relatively small number is generally best.
- `defer` - Do not attempt to schedule jobs individually at submit time. Can be useful for high-throughput computing.
- `defer_batch` - Like `defer`, but only will defer scheduling for batch jobs. Interactive allocations from `salloc/srun` will still attempt to schedule immediately upon submission.
- `max_switch_wait=#` - Specifies the maximum time a job can wait for desired number of leaf switches. Default value is 300 seconds. This is valuable for different topology clusters.
- `partition_job_depth=#` - Specifies how many jobs are tested in any single partition, default value is 0 (no limit).
- `sched_interval=#` - Specifies how frequently, in seconds, the main scheduling loop will execute and test all pending jobs. The default value is 60 seconds.

Main Scheduler

In order to maintain a quick scheduling process, the controller analyzes the jobs destined for a particular partition. The queued jobs are sorted in priority order, with the higher priority jobs at the top. The scheduler attempts to place the jobs on that priority order. 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.

Many of our customers tune this "Quick scheduling" for their benefit. If your jobs are going to stick around in the queue for a day, then there is no need to "Tune" this behavior, so turn it off. However, let's assume that your goal is to get instantaneous response from the scheduler. So let's assume you have an interactive queue (high priority queue) for short-lived, smaller jobs, and a batch queue for longer-lived jobs. If you have default_queue_depth=5, and the partition_job_depth=5 on the debug queue (partition), then the scheduler will never look at the debug queue jobs because it stops after the 5 it evaluates from the higher-priority interactive queue.

Backfill Scheduling

The backfill scheduling plugin is loaded by default:

```
slurm.conf
SchedulerType=sched/backfill
```

In order to disable backfill, you turn on the built in "Strict Priority" method which, in essence, turns the scheduler into a pure FIFO mechanism:

```
slurm.conf
SchedulerType=sched/builtin
```

Without backfill scheduling, each partition is scheduled strictly in priority order, which typically results in significantly lower system utilization and responsiveness than is otherwise possible. Backfill scheduling will start lower priority jobs if doing so does not delay the expected start time of any higher priority jobs. Since the expected start time of pending jobs depends upon the expected completion time of running jobs, reasonably accurate time limits are important for backfill scheduling to work well.

Slurm's backfill scheduler takes into consideration every running job. It then considers pending jobs in priority order, determining when and where each will start, taking into consideration the possibility of job preemption, gang scheduling, generic resource (GRES) requirements, memory requirements, etc. If the job under consideration can start immediately without impacting the expected start time of any higher priority job, then it does so. Otherwise the resources required by the job will be reserved during the job's expected execution time. The backfill plugin will set the expected start time for pending jobs. A job's expected start time can be seen using the squeue --start command.

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

- **DefaultTime** - Default job time limit (specify value by partition)

- **MaxTime** - Maximum job time limit (specify value by partition)
- **OverTimeLimit** - Amount by which a job can exceed its time limit before it is killed. A system-wide configuration parameter.

Backfill scheduling is a time consuming operation. Locks are released briefly every two seconds so that other options can be processed, for example to process new job submission requests. Backfill scheduling can optionally continue execution after the lock release and ignore newly submitted jobs (SchedulerParameters=bf_continue). Doing so will permit consideration of more jobs, but may result in the delayed scheduling of newly submitted jobs. A list of SchedulerParameters configuration parameters related to backfill scheduling follows. See the `slurm.conf(5)` man page for more details.

- **bf_continue** - If set, then continue backfill scheduling after periodically releasing locks for other operations.
- **bf_ignore_newly_avail_nodes** - If set, then only resources available at the beginning of a backfill cycle will be considered for use. Otherwise resources made available during that backfill cycle may be used for lower priority jobs, delaying the initiation of higher priority jobs when using bf_continue. Disabled by default.
- **bf_interval=#** - Interval between backfill scheduling attempts. Default value is 30 seconds.
- **bf_max_job_part=#** - Maximum number of jobs to initiate per partition in each backfill cycle. Default value is 0 (no limit).
- **bf_max_job_start=#** - Maximum number of jobs to initiate in each backfill cycle. Default value is 0 (no limit).
- **bf_max_job_test=#** - Maximum number of jobs consider for backfill scheduling in each backfill cycle. Default value is 100 jobs.
- **bf_max_job_user=#** - Maximum number of jobs to initiate per user in each backfill cycle. Default value is 0 (no limit).
- **bf_resolution=#** - Time resolution of backfill scheduling. Default value is 60 seconds. Larger values are appropriate if job time limits are imprecise and/or small delays in starting pending jobs in order to achieve higher system utilization is desired.
- **bf_window=#** - How long, in minutes, into the future to look when determining when and where jobs can start. Higher values result in more overhead and less responsiveness. A value at least as long as the highest allowed time limit is generally advisable to prevent job starvation. In order to limit the amount of data managed by the backfill scheduler, if the value of bf_window is increased, then it is generally advisable to also increase bf_resolution. The default value is 1440 minutes (one day).
- **bf_yield_interval=#** - The backfill scheduler will periodically relinquish locks in order for other pending operations to take place. This specifies the times when the locks are relinquish in microseconds. The default value is 2,000,000 microseconds (2 seconds). Smaller values may be helpful for high throughput computing when used in conjunction with the bf_continue option.
- **bf_yield_sleep=#** - The backfill scheduler will periodically relinquish locks in order for other pending operations to take place. This specifies the length of time for which the locks are relinquish in microseconds. The default value is 500,000 microseconds (0.5 seconds).

Consumable Resources

Slurm, using the default node allocation plug-in (SelectType=select/linear), allocates nodes to jobs in exclusive mode. This means that even when all the resources within a node are not utilized by a given job, another job will not have access to these resources. Nodes possess resources such as processors, memory, swap, local disk, etc.

and jobs consume these resources. The exclusive use default policy in Slurm can result in inefficient utilization of the cluster and of its nodes resources. Slurm's cons_tres or consumable resource plugin is available to manage resources on a much more fine-grained basis as described below. .

Using the Consumable Resource Allocation Plugin: select/cons_tres

- Consumable resources has been enhanced with several new resources --namely CPU (same as in previous version), Socket, Core, Memory as well as any combination of the logical processors with Memory:
 - CPU (CR_CPU): CPU as a consumable resource.
 - No notion of sockets, cores, or threads.
 - On a multi-core system CPUs will be cores.
 - On a multi-core/hyperthread system CPUs will be threads.
 - On a single-core systems CPUs are CPUs. ;-)
 - Board (CR_Board): Baseboard as a consumable resource.
 - Socket (CR_Socket): Socket as a consumable resource.
 - Core (CR_Core): Core as a consumable resource.
 - Memory (CR_Memory) Memory only as a consumable resource. **Note!** CR_Memory assumes OverSubscribe=Yes
 - Socket and Memory (CR_Socket_Memory): Socket and Memory as consumable resources.
 - Core and Memory (CR_Core_Memory): Core and Memory as consumable resources.
 - CPU and Memory (CR_CPU_Memory) CPU and Memory as consumable resources.
 - GPU (New in 19.05 release with cons_tres)
- In the cases where Memory is the consumable resource or one of the two consumable resources the RealMemory parameter, which defines a node's amount of real memory in `slurm.conf`, must be set when FastSchedule=1.
- srun's -E extension for sockets, cores, and threads are ignored within the node allocation mechanism when CR_CPU or CR_CPU_MEMORY is selected. It is considered to compute the total number of tasks when -n is not specified.
- The job submission commands (salloc, sbatch and srun) support the options --mem=MB and --mem-per-cpu=MB permitting users to specify the maximum amount of real memory per node or per allocated required. This option is required in the environments where Memory is a consumable resource. It is important to specify enough memory since Slurm will not allow the application to use more than the requested amount of real memory. The default value for --mem is 1 MB. see srun man page for more details.
- All CR_s assume OverSubscribe=No or OverSubscribe=Force EXCEPT for CR_MEMORY which assumes OverSubscribe=Yes
- The consumable resource plugin is enabled via SelectType and SelectTypeParameter in the `slurm.conf`.

```
#  
# Excerpts from sample slurm.conf file  
#  
SelectType=select/cons_tres  
SelectTypeParameters=CR_Core_Memory
```

- Using --overcommit or -O is allowed. When the process to logical processor pinning is enabled by using an appropriate TaskPlugin configuration parameter, the extra processes will time share the allocated resources.

General Comments

- Slurm's default select/linear plugin is using a best fit algorithm based on number of consecutive nodes. The same node allocation approach is used in select/cons_tres for consistency.
- The select/cons_tres plugin is enabled or disabled cluster-wide.
- In the case where select/cons_tres is not enabled, the normal Slurm behaviors are not disrupted. The only changes, users see when using the select/cons_tres plugin, are that jobs can be co-scheduled on nodes when resources permit it. The rest of Slurm, such as srun and its options (except srun -s ...), etc. are not affected by this plugin. Slurm is, from a user point of view, working the same way as when using the default node selection scheme.
- The --exclusive srun option allows users to request nodes in exclusive mode even when consumable resources is enabled. see "man srun" for details.
- srun's -s or --oversubscribe is incompatible with the consumable resource environment and will therefore not be honored. Since in this environment nodes are shared by default, --exclusive allows users to obtain dedicated nodes.

Examples of CR_Memory, CR_Socket_Memory, and CR_CPU_Memory type consumable resources

```
# sinfo -lNe
NODELIST      NODES PARTITION   STATE    CPUS   S:C:T MEMORY
hydra[12-16]      5 allNodes*   ...        4 2:2:1    2007
```

Using select/cons_tres plug-in with CR_Memory

```
Example:
# srun -N 5 -n 20 --mem=1000 sleep 100 &    <-- running
# srun -N 5 -n 20 --mem=10 sleep 100 &    <-- running
# srun -N 5 -n 10 --mem=1000 sleep 100 &    <-- queued and waiting for resources

# squeue
JOBID PARTITION      NAME      USER ST    TIME   NODES NODELIST (REASON)
 1820  allNodes      sleep  sballe PD  0:00       5 (Resources)
 1818  allNodes      sleep  sballe R   0:17       5 hydra[12-16]
 1819  allNodes      sleep  sballe R   0:11       5 hydra[12-16]
```

Using select/cons_tres plug-in with CR_Socket_Memory (2 sockets/node)

```
Example 1:
# srun -N 5 -n 5 --mem=1000 sleep 100 &    <-- running
# srun -n 1 -w hydra12 --mem=2000 sleep 100 &    <-- queued and waiting for resources

# squeue
JOBID PARTITION      NAME      USER ST    TIME   NODES NODELIST (REASON)
 1890  allNodes      sleep  sballe PD  0:00       1 (Resources)
```

```

1889 allNodes sleep sballe R 0:08          5 hydra[12-16]

Example 2:
# srun -N 5 -n 10 --mem=10 sleep 100 & <-- running
# srun -n 1 --mem=10 sleep 100 & <-- queued and waiting for resources

# squeue
JOBID PARTITION NAME USER ST TIME NODES NODELIST(REASON)
1831 allNodes sleep sballe PD 0:00      1 (Resources)
1830 allNodes sleep sballe R 0:07      5 hydra[12-16]

```

Using select/cons_tres plug-in with CR_CPU_Memory (4 CPUs/node)

```

Example 1:
# srun -N 5 -n 5 --mem=1000 sleep 100 & <-- running
# srun -N 5 -n 5 --mem=10 sleep 100 & <-- running
# srun -N 5 -n 5 --mem=1000 sleep 100 & <-- queued and waiting for resources

# squeue
JOBID PARTITION NAME USER ST TIME NODES NODELIST(REASON)
1835 allNodes sleep sballe PD 0:00      5 (Resources)
1833 allNodes sleep sballe R 0:10      5 hydra[12-16]
1834 allNodes sleep sballe R 0:07      5 hydra[12-16]

Example 2:
# srun -N 5 -n 20 --mem=10 sleep 100 & <-- running
# srun -n 1 --mem=10 sleep 100 & <-- queued and waiting for resources

# squeue
JOBID PARTITION NAME USER ST TIME NODES NODELIST(REASON)
1837 allNodes sleep sballe PD 0:00      1 (Resources)
1836 allNodes sleep sballe R 0:11      5 hydra[12-16]

```

Example of Node Allocations Using Consumable Resource Plugin

The following example illustrates the different ways four jobs are allocated across a cluster using (1) Slurm's default allocation (exclusive mode) and (2) a processor as consumable resource approach.

It is important to understand that the example listed below is a contrived example and is only given here to illustrate the use of CPU as consumable resources. Job 2 and Job 3 call for the node count to equal the processor count. This would typically be done because that one task per node requires all of the memory, disk space, etc. The bottleneck would not be processor count.

Trying to execute more than one job per node will almost certainly severely impact parallel job's performance. The biggest beneficiary of CPUs as consumable resources will be serial jobs or jobs with modest parallelism, which can effectively share resources. On many systems with larger processor count, jobs typically run one fewer task than there are processors to minimize interference by the kernel and daemons.

The example cluster is composed of 4 nodes (10 CPUs in total):

- linux01 (with 2 processors),
- linux02 (with 2 processors),

- linux03 (with 2 processors), and
- linux04 (with 4 processors).

The four jobs are the following:

- [2] srun -n 4 -N 4 sleep 120 &
- [3] srun -n 3 -N 3 sleep 120 &
- [4] srun -n 1 sleep 120 &
- [5] srun -n 3 sleep 120 &

The user launches them in the same order as listed above.

Using Slurm's Default Node Allocation (Non-shared Mode)

The four jobs have been launched and 3 of the jobs are now pending, waiting to get resources allocated to them. Only Job 2 is running since it uses one CPU on all 4 nodes. This means that linux01 to linux03 each have one idle CPU and linux04 has 3 idle CPUs.

# squeue							
JOBID	PARTITION	NAME	USER	ST	TIME	NODES	NODELIST (REASON)
3	lsf	sleep	root	PD	0:00	3	(Resources)
4	lsf	sleep	root	PD	0:00	1	(Resources)
5	lsf	sleep	root	PD	0:00	1	(Resources)
2	lsf	sleep	root	R	0:14	4	xc14n[13-16]

Once Job 2 is finished, Job 3 is scheduled and runs on linux01, linux02, and linux03. Job 3 is only using one CPU on each of the 3 nodes. Job 4 can be allocated onto the remaining idle node (linux04) so Job 3 and Job 4 can run concurrently on the cluster.

Job 5 has to wait for idle nodes to be able to run.

# squeue							
JOBID	PARTITION	NAME	USER	ST	TIME	NODES	NODELIST (REASON)
5	lsf	sleep	root	PD	0:00	1	(Resources)
3	lsf	sleep	root	R	0:11	3	xc14n[13-15]
4	lsf	sleep	root	R	0:11	1	xc14n16

Once Job 3 finishes, Job 5 is allocated resources and can run.

The advantage of the exclusive mode scheduling policy is that the job gets all the resources of the assigned nodes for optimal parallel performance. The drawback is that jobs can tie up large amount of resources that it does not use and which cannot be shared with other jobs.

Using a Processor Consumable Resource Approach

The output of squeue shows that we have 3 out of the 4 jobs allocated and running. This is a 2 running job increase over the default Slurm approach.

Job 2 is running on nodes linux01 to linux04. Job 2's allocation is the same as for Slurm's default allocation which is that it uses one CPU on each of the 4 nodes. Once Job 2 is scheduled and running, nodes linux01, linux02 and

linux03 still have one idle CPU each and node linux04 has 3 idle CPUs. The main difference between this approach and the exclusive mode approach described above is that idle CPUs within a node are now allowed to be assigned to other jobs.

It is important to **note** that assigned doesn't mean oversubscription. The consumable resource approach tracks how much of each available resource (in our case CPUs) must be dedicated to a given job. This allows us to prevent per node oversubscription of resources (CPUs).

Once Job 2 is running, Job 3 is scheduled onto node linux01, linux02, and Linux03 (using one CPU on each of the nodes) and Job 4 is scheduled onto one of the remaining idle CPUs on Linux04.

Job 2, Job 3, and Job 4 are now running concurrently on the cluster.

```
# squeue
JOBID PARTITION NAME USER ST TIME NODES NODELIST(REASON)
 5     lsf sleep root  PD 0:00      1 (Resources)
 2     lsf sleep root   R 0:13      4 linux[01-04]
 3     lsf sleep root   R 0:09      3 linux[01-03]
 4     lsf sleep root   R 0:05      1 linux04

# sinfo -lNe
NODELIST      NODES PARTITION          STATE CPUS MEMORY TMP_DISK WEIGHT FEATURES REASON
linux[01-03]      3     lsf* allocated    2   2981      1       1 (null) none
linux04          1     lsf* allocated    4   3813      1       1 (null) none
```

Once Job 2 finishes, Job 5, which was pending, is allocated available resources and is then running as illustrated below:

```
# squeue
JOBID PARTITION NAME USER ST TIME NODES NODELIST(REASON)
 3     lsf sleep root   R 1:58      3 linux[01-03]
 4     lsf sleep root   R 1:54      1 linux04
 5     lsf sleep root   R 0:02      3 linux[01-03]
# sinfo -lNe
NODELIST      NODES PARTITION          STATE CPUS MEMORY TMP_DISK WEIGHT FEATURES REASON
linux[01-03]      3     lsf* allocated    2   2981      1       1 (null) none
linux04          1     lsf*     idle     4   3813      1       1 (null) none
```

Job 3, Job 4, and Job 5 are now running concurrently on the cluster.

```
# squeue
JOBID PARTITION NAME USER ST TIME NODES NODELIST(REASON)
 5     lsf sleep root   R 1:52      3 linux[01-03]
```

Job 3 and Job 4 have finished and Job 5 is still running on nodes linux[01-03].

The advantage of the consumable resource scheduling policy is that the job throughput can increase dramatically. The overall job throughput/productivity of the cluster increases thereby reducing the amount of time users have to

wait for their job to complete as well as increasing the overall efficiency of the use of the cluster. The drawback is that users do not have the entire node dedicated to their job since they have to share nodes with other jobs if they do not use all of the resources on the nodes.

We have added a "--exclusive" option to srun which allow users to specify that they would like their allocated nodes in exclusive mode. For more information see "man srun". The reason for that is if users have mpi/threaded/openMP programs that will take advantage of all the CPUs within a node but only need one mpi process per node.

Generic Resource (GRES) Scheduling

Generic resource (GRES) scheduling is supported through a flexible plugin mechanism. Support is currently provided for Graphics Processing Units (GPUs), CUDA Multi-Process Service (MPS), and Intel® Many Integrated Core (MIC) processors.

GRES Configuration

Slurm supports no generic resources in the default configuration. One must explicitly specify which resources are to be managed in the `slurm.conf` configuration file. The configuration parameters of interest are `GresTypes` and `Gres`.

For more details, see `GresTypes` and `Gres` in the `slurm.conf` man page.

Note that the GRES specification for each node works in the same fashion as the other resources managed. Depending upon the value of the `FastSchedule` parameter, nodes which are found to have fewer resources than configured will be placed in a DOWN state.

Snippet from an example `slurm.conf` file:

```
# Configure support for our four GPUs (with MPS), plus bandwidth
GresTypes=gpu,mps,bandwidth

NodeName=tux[0-7] Gres=gpu:tesla:2,gpu:kepler:2,mps:400,bandwidth:lustre:no_consume:4
```

Each compute node with generic resources typically contain a `gres.conf` file describing which resources are available on the node, their count, associated device files and cores which should be used with those resources.

In the case of GPUs, if `AutoDetect=nvml` in `gres.conf` and the NVML library is installed on the node and was present during Slurm configuration, the missing configuration details will be automatically gathered using the NVML library. Configuration information about all other generic resource must explicitly be described in the `gres.conf` file.

To view available `gres.conf` configuration parameters, see the `gres.conf` man page.

Example `gres.conf` file:

```
## Configure support for four GPUs (with MPS), plus bandwidth
AutoDetect=nvml
Name=gpu Type=gp100 File=/dev/nvidia0 Cores=0,1
```

```
Name=gpu Type=gp100 File=/dev/nvidia1 Cores=0,1
Name=gpu Type=p6000 File=/dev/nvidia2 Cores=2,3
Name=gpu Type=p6000 File=/dev/nvidia3 Cores=2,3
Name=mps Count=200 File=/dev/nvidia0
Name=mps Count=200 File=/dev/nvidia1
Name=mps Count=100 File=/dev/nvidia2
Name=mps Count=100 File=/dev/nvidia3
Name=bandwidth Type=lustre Count=4G
```

Running GRES Jobs

Jobs will not be allocated any generic resources unless specifically requested at job submit time using the options:

--gres: Generic resources required per node

--gpu: GPUs required per job

--gpu-per-node: GPUs required per node. Equivalent to the --gres option for GPUs.

--gpu-per-socket: GPUs required per socket. Requires the job to specify a task socket.

--gpu-per-task: GPUs required per task. Requires the job to specify a task count.

All of these options are supported by the salloc, sbatch and srun commands. **Note** that all of the --gpu* options are only supported by Slurm's select/cons_tres plugin (found in Slurm 19.05.X). Jobs requesting these options when the select/cons_tres plugin is not configured will be rejected. The --gres option requires an argument specifying which generic resources are required and how many resources using the form name[:type:count] while all of the --gpu* options require an argument of the form [type]:count. The name is the same name as specified by the GresTypes and Gres configuration parameters. type identifies a specific type of that generic resource (e.g. a specific model of GPU). count specifies how many resources are required and has a default value of 1. For example:

```
sbatch --gres=gpu:kepler:2 ....
```

Several addition resource requirement specifications are available specifically for GPUs and detailed descriptions about these options are available in the man pages for the job submission commands. As for the --gpu* option, these options are only supported by Slurm's select/cons_tres plugin.

--cpus-per-gpu: Count of CPUs allocated per GPU.

--gpu-bind: Define how tasks are bound to GPUs.

--gpu-freq: Specify GPU frequency and/or GPU memory frequency.

--mem-per-gpu: Memory allocated per GPU.

Jobs will be allocated specific generic resources as needed to satisfy the request. If the job is suspended, those resources do not become available for use by other jobs.

Job steps can be allocated generic resources from those allocated to the job using the --gres option with the srun command as described above. By default, a job step will be allocated all of the generic resources allocated to the

job. If desired, the job step may explicitly specify a different generic resource count than the job. This design choice was based upon a scenario where each job executes many job steps. If job steps were granted access to all generic resources by default, some job steps would need to explicitly specify zero generic resource counts, which we considered more confusing. The job step can be allocated specific generic resources and those resources will not be available to other job steps. A simple example is shown below.

```
#!/bin/bash
#
# gres_test.bash
# Submit as follows:
# sbatch --gres=gpu:4 -n4 -N1-1 gres_test.bash
#
srun --gres=gpu:2 -n2 --exclusive show_device.sh &
srun --gres=gpu:1 -n1 --exclusive show_device.sh &
srun --gres=gpu:1 -n1 --exclusive show_device.sh &
wait
```

GPU Management

In the case of Slurm's GRES plugin for GPUs, the environment variable CUDA_VISIBLE_DEVICES is set for each job step to determine which GPUs are available for its use on each node. This environment variable is only set when tasks are launched on a specific compute node (no global environment variable is set for the salloc command and the environment variable set for the sbatch command only reflects the GPUs allocated to that job on that node, node zero of the allocation). CUDA version 3.1 (or higher) uses this environment variable in order to run multiple jobs or job steps on a node with GPUs and ensure that the resources assigned to each are unique. In the example above, the allocated node may have four or more graphics devices. In that case, CUDA_VISIBLE_DEVICES will reference unique devices for each file and the output might resemble this:

```
JobStep=1234.0 CUDA_VISIBLE_DEVICES=0,1
JobStep=1234.1 CUDA_VISIBLE_DEVICES=2
JobStep=1234.2 CUDA_VISIBLE_DEVICES=3
```

NOTE: Be sure to specify the File parameters in the `gres.conf` file and ensure they are in the increasing numeric order.

The CUDA_VISIBLE_DEVICES environment variable will also be set in the job's Prolog and Epilog programs. **Note** that the environment variable set for the job may differ from that set for the Prolog and Epilog if Slurm is configured to constrain the device files visible to a job using Linux cgroup. This is because the Prolog and Epilog programs run outside of any Linux cgroup while the job runs inside of the cgroup and may thus have a different set of visible devices. For example, if a job is allocated the device "/dev/nvidia1", then CUDA_VISIBLE_DEVICES will be set to a value of "1" in the Prolog and Epilog while the job's value of CUDA_VISIBLE_DEVICES will be set to a value of "0" (i.e.the first GPU device visible to the job). For more information see the Prolog and Epilog Guide.

When possible, Slurm automatically determines the GPUs on the system using NVIDIA's NVML library. The NVML library (which powers the `nvidia-smi` tool) numbers GPUs in order by their PCI bus IDs. For this numbering to match the numbering reported by CUDA, the CUDA_DEVICE_ORDER environmental variable must be set to `CUDA_DEVICE_ORDER=PCI_BUS_ID`.

GPU device files (e.g./dev/nvidia1) are based on the Linux minor number assignment, while NVML's device numbers are assigned via PCI bus ID, from lowest to highest. Mapping between these two is indeterministic and system dependent, and could vary between boots after hardware or OS changes. For the most part, this assignment seems fairly stable. However, an after-bootup check is required to guarantee that a GPU device is assigned to a specific device file.

Please consult the NVIDIA CUDA documentation for more information about the CUDA_VISIBLE_DEVICES and CUDA_DEVICE_ORDER environmental variables.

MPS Management

CUDA Multi-Process Service (MPS) provides a mechanism where GPUs can be shared by multiple jobs, where each job is allocated some percentage of the GPU's resources. The total count of MPS resources available on a node should be configured in the `slurm.conf` file (e.g."NodeName=tux[1-16] Gres=gpu:2,mps:200"). Several options are available for configuring MPS in the `gres.conf` file as listed below with examples following that:

1. No MPS configuration: The count of gres/mps elements defined in the `slurm.conf` will be evenly distributed across all GPUs configured on the node. For the example, "NodeName=tux[1-16] Gres=gpu:2,mps:200" will configure a count of 100 gres/mps resources on each of the two GPUs.
2. MPS configuration includes only the Name and Count parameters: The count of gres/mps elements will be evenly distributed across all GPUs configured on the node. This is similar to case 1, but places duplicate configuration in the `gres.conf` file.
3. MPS configuration includes the Name, File and Count parameters: Each File parameter should identify the device file path of a GPU and the Count should identify the number of gres/mps resources available for that specific GPU device. This may be useful in a heterogeneous environment. For example, some GPUs on a node may be more powerful than others and thus be associated with a higher gres/mps count. Another use case would be to prevent some GPUs from being used for MPS (i.e.they would have an MPS count of zero).

Note that Type and Cores parameters for gres/mps are ignored. That information is copied from the gres/gpu configuration.

Note the Count parameter is translated to a percentage, so the value would typically be a multiple of 100.

Note that if NVIDIA's NVML library is installed, the GPU configuration (i.e.Type, File, Cores and Links data) will be automatically gathered from the library and need not be recorded in the `gres.conf` file.

Note the same GPU can be allocated either as a GPU type of GRES or as an MPS type of GRES, but not both. In other words, once a GPU has been allocated as a gres/gpu resource it will not be available as a gres/mps. Likewise, once a GPU has been allocated as a gres/mps resource it will not be available as a gres/gpu. However the same GPU can be allocated as MPS generic resources to multiple jobs belonging to multiple users, so long as the total count of MPS allocated to jobs does not exceed the configured count. Some example configurations for Slurm's `gres.conf` file are shown below.

```
# Example 1 of gres.conf
# Configure support for four GPUs (with MPS)
AutoDetect=nvml
Name=gpu Type=gp100 File=/dev/nvidia0 Cores=0,1
Name=gpu Type=gp100 File=/dev/nvidia1 Cores=0,1
Name=gpu Type=p6000 File=/dev/nvidia2 Cores=2,3
```

```
Name=gpu Type=p6000 File=/dev/nvidia3 Cores=2,3  
# Set gres/mps Count value to 100 on each of the 4 available GPUs  
Name=mps Count=400]
```

```
# Example 2 of gres.conf  
# Configure support for four differernt GPU types (with MPS)  
AutoDetect=nvml  
Name=gpu Type=gtx1080 File=/dev/nvidia0 Cores=0,1  
Name=gpu Type=gtx1070 File=/dev/nvidia1 Cores=0,1  
Name=gpu Type=gtx1060 File=/dev/nvidia2 Cores=2,3  
Name=gpu Type=gtx1050 File=/dev/nvidia3 Cores=2,3  
Name=mps Count=1300 File=/dev/nvidia0  
Name=mps Count=1200 File=/dev/nvidia1  
Name=mps Count=1100 File=/dev/nvidia2  
Name=mps Count=1000 File=/dev/nvidia3
```

NOTE: Slurm support for gres/mps requires the use of the select/cons_tres plugin.

Job requests for MPS will be processed the same as any other GRES except that the request must be satisfied using only one GPU per node and only one GPU per node may be configured for use with MPS. For example, a job request for "--gres=mps:50" will not be satisfied by using 20 percent of one GPU and 30 percent of a second GPU on a single node. Multiple jobs from different users can use MPS on a node at the same time. **Note** that GRES types of GPU and MPS can not be requested within a single job. Also jobs requesting MPS resources can not specify a GPU frequency.

A prolog program should be used to start and stop MPS servers as needed. A sample prolog script to do this is included with the Slurm distribution in the location etc/prolog.example. Its mode of operation is if a job is allocated gres/mps resources then the Prolog will have the CUDA_VISIBLE_DEVICES, CUDA_MPS_ACTIVE_THREAD_PERCENTAGE, and SLURM_JOB_UID environment variables set. The Prolog should then make sure that an MPS server is started for that GPU and user (UID == User ID). It also records the GPU device ID in a local file. If a job is allocated gres/gpu resources then the Prolog will have the CUDA_VISIBLE_DEVICES and SLURM_JOB_UID environment variables set (no CUDA_MPS_ACTIVE_THREAD_PERCENTAGE). The Prolog should then terminate any MPS server associated with that GPU. It may be necessary to modify this script as needed for the local environment. For more information see the Prolog and Epilog Guide.

Jobs requesting MPS resources will have the CUDA_VISIBLE_DEVICES and CUDA_DEVICE_ORDER environment variables set. The device ID is relative to those resources under MPS server control and will always have a value of zero in the current implementation (only one GPU will be usable in MPS mode per node). The job will also have the CUDA_MPS_ACTIVE_THREAD_PERCENTAGE environment variable set to that job's percentage of MPS resources available on the assigned GPU. The percentage will be calculated based upon the portion of the configured Count on the Gres is allocated to a job of step. For example, a job requesting "--gres=gpu:200" and using configuration example 2 above would be allocated

15% of the gtx1080 (File=/dev/nvidia0, $200 \times 100 / 1300 = 15$), or

16% of the gtx1070 (File=/dev/nvidia0, $200 \times 100 / 1200 = 16$), or

18% of the gtx1060 (File=/dev/nvidia0, 200 x 100 / 1100 = 18), or

20% of the gtx1050 (File=/dev/nvidia0, 200 x 100 / 1000 = 20).

An alternate mode of operation would be to permit jobs to be allocated whole GPUs then trigger the starting of an MPS server based upon comments in the job. For example, if a job is allocated whole GPUs then search for a comment of "mps-per-gpu" or "mps-per-node" in the job (using the "scontrol show job" command) and use that as a basis for starting one MPS daemon per GPU or across all GPUs repectively.

Please consult the NVIDIA Multi-Process Service documentation for more information about MPS.

MIC Management

Slurm can be used to provide resource management for systems with the Intel® Many Integrated Core (MIC) processor. Slurm sets an OFFLOAD_DEVICES environment variable, which controls the selection of MICs available to a job step. The OFFLOAD_DEVICES environment variable is used by both Intel LEO (Language Extensions for Offload) and the MKL (Math Kernel Library) automatic offload. (This is very similar to how the CUDA_VISIBLE_DEVICES environment variable is used to control which GPUs can be used by CUDA™ software.) If no MICs are reserved via GRES, the OFFLOAD_DEVICES variable is set to -1. This causes the code to ignore the offload directives and run MKL routines on the CPU. The code will still run but only on the CPU. This also gives a somewhat cryptic warning:

```
offload warning: OFFLOAD_DEVICES device number -1 does not correspond  
to a physical device
```

The offloading is automatically scaled to all the devices, (e.g. if --gres=mic:2 is defined) then all offloads use two MICs unless explicitly defined in the offload pragmas.

CPU Management User and Administrator Guide

The purpose of this guide is to assist Slurm users and administrators in selecting configuration options and composing command lines to manage the use of CPU resources by jobs, steps and tasks. The document is divided into the following sections:

- CPU Management Steps performed by Slurm
- Getting Information about CPU usage by Jobs/Steps/Tasks
- CPU Management and Slurm Accounting
- CPU Management Examples

CPU Management through user commands is constrained by the configuration parameters chosen by the Slurm administrator. The interactions between different CPU management options are complex and often difficult to predict. Some experimentation may be required to discover the exact combination of options needed to produce a desired outcome. Users and administrators should refer to the man pages for `slurm.conf`, `cgroup.conf`, `salloc`, `sbatch` and `srun` for detailed explanations of each option.

CPU Management Steps performed by Slurm

Slurm uses four basic steps to manage CPU resources for a job/step:

- Step 1: Selection of Nodes
- Step 2: Allocation of CPUs from the selected Nodes
- Step 3: Distribution of Tasks to the selected Nodes
- Step 4: Optional Distribution and Binding of Tasks to CPUs within a Node

STEP 1: SELECTION OF NODES

In Step 1, Slurm selects the set of nodes from which CPU resources are to be allocated to a job or job step. Node selection is therefore influenced by many of the configuration and command line options that control the allocation of CPUs (Step 2 below). If SelectType=select/linear is configured, all resources on the selected nodes will be allocated to the job/step. This is the default. If SelectType=select/cons_tres (or cons_tres) is configured, individual sockets, cores and threads may be allocated from the selected nodes as consumable resources. The consumable resource type is defined by SelectTypeParameters.

`slurm.conf` options that control Step 1

<code>slurm.conf</code> parameter	Possible values	Description
NodeName	<name of the node> Plus additional parameters. See man page for details.	Defines a node. This includes the number and layout of boards, sockets, cores, threads and processors (logical CPUs) on the node.
PartitionName	<name of the partition> Plus additional parameters. See man page for details.	Defines a partition. Several parameters of the partition definition affect the selection of nodes (e.g., Nodes, OverSubscribe, MaxNodes)
SlurmdParameters	config_overrides	Controls how the information in a node definition is used
SelectType	select/linear select/cons_tres select/cons_tres	Controls whether CPU resources are allocated to jobs and job steps in units of whole nodes or as consumable resources (sockets, cores or threads).
SelectTypeParameters	CR_CPU CR_CPU_Memory CR_Core CR_Core_Memory CR_Socket CR_Socket_Memory	Defines the consumable resource type and controls other aspects of CPU resource allocation by the select plugin.

	Plus additional options. See man page for details.	
--	--	--

srun/salloc/sbatch command line options that control Step 1

Command line option	Possible values	Description
-B, --extra-node-info	<sockets[:cores[:threads]]>	Restricts node selection to nodes with a specified layout of sockets, cores and threads.
-C, --constraint	<list>	Restricts node selection to nodes with specified attributes
--contiguous	N/A	Restricts node selection to contiguous nodes
--cores-per-socket	<cores>	Restricts node selection to nodes with at least the specified number of cores per socket
-c, --cpus-per-task	<ncpus>	Controls the number of CPUs allocated per task
--exclusive	N/A	Prevents sharing of allocated nodes with other jobs. Suballocates CPUs to job steps.
-F, --nodefile	<node file>	File containing a list of specific nodes to be selected for the job (salloc and sbatch only)
--hint	compute_bound memory_bound [no]multithread	Additional controls on allocation of CPU resources
--mincpus	<n>	Controls the minimum number of CPUs allocated per node
-N, --nodes	<minnodes[-maxnodes]>	Controls the minimum/maximum number of nodes allocated to the job
-n, --ntasks	<number>	Controls the number of tasks to be created for the job

--ntasks-per-core	<number>	Controls the maximum number of tasks per allocated core
--ntasks-per-socket	<number>	Controls the maximum number of tasks per allocated socket
--ntasks-per-node	<number>	Controls the maximum number of tasks per allocated node
-O, --overcommit	N/A	Allows fewer CPUs to be allocated than the number of tasks
-p, --partition	<partition_names>	Controls which partition is used for the job
-s, --oversubscribe	N/A	Allows sharing of allocated nodes with other jobs
--sockets-per-node	<sockets>	Restricts node selection to nodes with at least the specified number of sockets
--threads-per-core	<threads>	Restricts node selection to nodes with at least the specified number of threads per core
-w, --nodelist	<host1,host2,...or filename>	List of specific nodes to be allocated to the job
-x, --exclude	<host1,host2,...or filename>	List of specific nodes to be excluded from allocation to the job
-Z, --no-allocate	N/A	Bypass normal allocation (privileged option available to users "SlurmUser" and "root" only)

STEP 2: ALLOCATION OF CPUS FROM THE SELECTED NODES

In Step 2, Slurm allocates CPU resources to a job/step from the set of nodes selected in Step 1. CPU allocation is therefore influenced by the configuration and command line options that relate to node selection. If SelectType=select/linear is configured, all resources on the selected nodes will be allocated to the job/step. If SelectType=select/cons_tres is configured, individual sockets, cores and threads may be allocated from the selected nodes as consumable resources. The consumable resource type is defined by SelectTypeParameters.

When using SelectType=select/cons_tres, the default allocation method across nodes is block allocation (allocate all available CPUs in a node before using another node). The default allocation method within a node is cyclic

allocation (allocate available CPUs in a round-robin fashion across the sockets within a node). Users may override the default behavior using the appropriate command line options described below. The choice of allocation methods may influence which specific CPUs are allocated to the job/step.

Step 2 is performed by slurmctld and the select plugin.

`slurm.conf` options that control Step 2

slurm.conf parameter	Possible values	Description
NodeName	<name of the node> Plus additional parameters. See man page for details.	Defines a node. This includes the number and layout of boards, sockets, cores, threads and processors (logical CPUs) on the node.
PartitionName	<name of the partition> Plus additional parameters. See man page for details.	Defines a partition. Several parameters of the partition definition affect the selection of nodes (e.g., Nodes, OverSubscribe, MaxNodes)
SlurmdParameters	config_overrides	Controls how the information in a node definition is used
SelectType	select/linear select/cons_res select/cons_tres	Controls whether CPU resources are allocated to jobs and job steps in units of whole nodes or as consumable resources (sockets, cores or threads).
SelectTypeParameters	CR_CPU CR_CPU_Memory CR_Core CR_Core_Memory CR_Socket CR_Socket_Memory Plus additional options. See man page for details.	Defines the consumable resource type and controls other aspects of CPU resource allocation by the select plugin.

`srun/salloc/sbatch` command line options that control Step 2

Command line option	Possible values	Description
-B, --extra-node-info	<sockets[:cores[:threads]]>	Restricts node selection to nodes with a specified layout of sockets, cores and threads.
-C, --constraint	<list>	Restricts node selection to nodes with specified attributes
--contiguous	N/A	Restricts node selection to contiguous nodes
--cores-per-socket	<cores>	Restricts node selection to nodes with at least the specified number of cores per socket
-c, --cpus-per-task	<ncpus>	Controls the number of CPUs allocated per task
--distribution, -m	block cyclic arbitrary plane=<options>[:block cyclic]	The second specified distribution (after the ":") can be used to override the default allocation method within nodes
--exclusive	N/A	Prevents sharing of allocated nodes with other jobs. Suballocates CPUs to job steps.
-F, --nodefile	<node file>	File containing a list of specific nodes to be selected for the job (salloc and sbatch only)
--hint	compute_bound memory_bound [no]multithread	Additional controls on allocation of CPU resources
--mincpus	<n>	Controls the minimum number of CPUs allocated per node
-N, --nodes	<minnodes[-maxnodes]>	Controls the minimum/maximum number of nodes allocated to the job

-n, --ntasks	<number>	Controls the number of tasks to be created for the job
--ntasks-per-core	<number>	Controls the maximum number of tasks per allocated core
--ntasks-per-socket	<number>	Controls the maximum number of tasks per allocated socket
--ntasks-per-node	<number>	Controls the maximum number of tasks per allocated node
-O, --overcommit	N/A	Allows fewer CPUs to be allocated than the number of tasks
-p, --partition	<partition_names>	Controls which partition is used for the job
-s, --oversubscribe	N/A	Allows sharing of allocated nodes with other jobs
--sockets-per-node	<sockets>	Restricts node selection to nodes with at least the specified number of sockets
--threads-per-core	<threads>	Restricts node selection to nodes with at least the specified number of threads per core
-w, --nodelist	<host1,host2,...or filename>	List of specific nodes to be allocated to the job
-x, --exclude	<host1,host2,...or filename>	List of specific nodes to be excluded from allocation to the job
-Z, --no-allocate	N/A	Bypass normal allocation (privileged option available to users "SlurmUser" and "root" only)

STEP 3: DISTRIBUTION OF TASKS TO THE SELECTED NODES

In Step 3, Slurm distributes tasks to the nodes that were selected for the job/step in Step 1. Each task is distributed to only one node, but more than one task may be distributed to each node. Unless overcommitment of CPUs to tasks is specified for the job, the number of tasks distributed to a node is constrained by the number of CPUs allocated on the node and the number of CPUs per task. If consumable resources is configured, or resource sharing is allowed, tasks from more than one job/step may run on the same node concurrently.

Step 3 is performed by slurmctld.

slurm.conf options that control Step 3

slurm.conf parameter	Possible values	Description
MaxTasksPerNode	<number>	Controls the maximum number of tasks that a job step can spawn on a single node

srun/salloc/sbatch command line options that control Step 3

Command line option	Possible values	Description
--distribution, -m	block cyclic arbitrary plane=<options>[:block cyclic]	The first specified distribution (before the ":") controls the sequence in which tasks are distributed to each of the selected nodes. Note that this option does not affect the number of tasks distributed to each node, but only the sequence of distribution.
--ntasks-per-core	<number>	Controls the maximum number of tasks per allocated core
--ntasks-per-socket	<number>	Controls the maximum number of tasks per allocated socket
--ntasks-per-node	<number>	Controls the maximum number of tasks per allocated node
-r, --relative	N/A	Controls which node is used for a job step

STEP 4: OPTIONAL DISTRIBUTION AND BINDING OF TASKS TO CPUS WITHIN A NODE

In optional Step 4, Slurm distributes and binds each task to a specified subset of the allocated CPUs on the node to which the task was distributed in Step 3. Different tasks distributed to the same node may be bound to the same subset of CPUs or to different subsets. This step is known as task affinity or task/CPU binding.

Step 4 is performed by slurmd and the task plugin.

`slurm.conf` options that control Step 4

slurm.conf parameter	Possible values	Description
TaskPlugin	task/none task/affinity task/cgroup	Controls whether this step is enabled and which task plugin to use

`cgroup.conf` options that control Step 4 (task/cgroup plugin only)

cgroup.conf parameter	Possible values	Description
ConstrainCores	yes/no	Controls whether jobs are constrained to their allocated CPUs

`srun/salloc/sbatch` command line options that control Step 4

Command line option	Possible values	Description
--cpu-bind	See man page	Controls binding of tasks to CPUs
--ntasks-per-core	<number>	Controls the maximum number of tasks per allocated core
--distribution, -m	block cyclic arbitrary plane=<options>[:block cyclic]	The second specified distribution (after the ":") controls the sequence in which tasks are distributed to allocated CPUs within a node for binding of tasks to CPUs

Additional Notes on CPU Management Steps

For consumable resources, it is important for users to understand the difference between cpu allocation (Step 2) and task affinity/binding (Step 4). Exclusive (unshared) allocation of CPUs as consumable resources limits the number of jobs/steps/tasks that can use a node concurrently. But it does not limit the set of CPUs on the node that each task distributed to the node can use. Unless some form of CPU/task binding is used (e.g., a task or spank plugin), all tasks distributed to a node can use all of the CPUs on the node, including CPUs not allocated to their job/step. This may have unexpected adverse effects on performance, since it allows one job to use CPUs allocated exclusively to another job. For this reason, it may not be advisable to configure consumable resources without also configuring task affinity. **Note** that task affinity can also be useful when select/linear (whole node allocation) is configured, to improve performance by restricting each task to a particular socket or other subset of CPU resources on a node.

Getting Information about CPU usage by Jobs/Steps/Tasks

There is no easy way to generate a comprehensive set of CPU management information for a job/step (allocation, distribution and binding). However, several commands/options provide limited information about CPU usage.

Command/Option	Information
scontrol show job option: --details	This option provides a list of the nodes selected for the job and the CPU ids allocated to the job on each node. Note that the CPU ids reported by this command are Slurm abstract CPU ids, not Linux/hardware CPU ids (as reported by, for example, /proc/cpuinfo).
Linux command: env	Man.Slurm environment variables provide information related to node and CPU usage: SLURM_JOB_CPUS_PER_NODE SLURM_CPUS_PER_TASK SLURM_CPU_BIND SLURM_DISTRIBUTION SLURM_JOB_NODELIST SLURM_TASKS_PER_NODE SLURM_STEP_NODELIST SLURM_STEP_NUM_NODES SLURM_STEP_NUM_TASKS

	SLURM_STEP_TASKS_PER_NODE SLURM_JOB_NODES SLURM_NTASKS SLURM_NPROCS SLURM_CPUS_ON_NODE SLURM_NODEID SLURMD_NODENAME
srun option: --cpu-bind=verbose	This option provides a list of the CPU masks used by task affinity to bind tasks to CPUs. Note that the CPU ids represented by these masks are Linux/hardware CPU ids, not Slurm abstract CPU ids as reported by scontrol, etc.
srun/salloc/sbatch option: -l	This option adds the task id as a prefix to each line of output from a task sent to stdout/stderr. This can be useful for distinguishing node-related and CPU-related information by task id for multi-task jobs/steps.
Linux command: cat /proc/<pid>/status grep Cpus_allowed_list	Given a task's pid (or "self" if the command is executed by the task itself), this command produces a list of the CPU ids bound to the task. This is the same information that is provided by --cpu-bind=verbose, but in a more readable format.

A NOTE ON CPU NUMBERING

The number and layout of logical CPUs known to Slurm is described in the node definitions in `slurm.conf`. This may differ from the physical CPU layout on the actual hardware. For this reason, Slurm generates its own internal, or "abstract", CPU numbers. These numbers may not match the physical, or "machine", CPU numbers known to Linux.

CPU Management Examples

The following examples illustrate some scenarios for managing CPU resources using Slurm. Many additional scenarios are possible. In each example, it is assumed that all CPUs on each node are available for allocation.

EXAMPLE NODE AND PARTITION CONFIGURATION

For these examples, the Slurm cluster contains the following nodes:

Nodename	n0	n1	n2	n3
Number of Sockets	2	2	2	2
Number of Cores per Socket	4	4	4	4
Total Number of Cores	8	8	8	8
Number of Threads (CPUs) per Core	1	1	1	2
Total Number of CPUs	8	8	8	16

And the following partitions:

PartitionName	regnodes	hypernode
Nodes	n0 n1 n2	n3
Default	YES	-

These entities are defined in `slurm.conf` as follows:

```

Nodename=n0 NodeAddr=node0 Sockets=2 CoresPerSocket=4 ThreadsPerCore=1 Procs=8
Nodename=n1 NodeAddr=node1 Sockets=2 CoresPerSocket=4 ThreadsPerCore=1 Procs=8 State=IDLE
Nodename=n2 NodeAddr=node2 Sockets=2 CoresPerSocket=4 ThreadsPerCore=1 Procs=8 State=IDLE
Nodename=n3 NodeAddr=node3 Sockets=2 CoresPerSocket=4 ThreadsPerCore=2 Procs=16 State=IDLE
PartitionName=regnodes Nodes=n0,n1,n2 OverSubscribe=YES Default=YES State=UP
PartitionName=hypernode Nodes=n3 State=UP

```

EXAMPLE 1: ALLOCATION OF WHOLE NODES

Allocate a minimum of two whole nodes to a job.

`slurm.conf` options:

```
SelectType=select/linear
```

Command line:

```
srun --nodes=2 ...
```

Comments:

The SelectType=select/linear configuration option specifies allocation in units of whole nodes. The --nodes=2 srun option causes Slurm to allocate at least 2 nodes to the job.

```
$ scontrol update job=101 ...
$ scontrol update job=101_1 ...
```

EXAMPLE 2: SIMPLE ALLOCATION OF CORES AS CONSUMABLE RESOURCES

A job requires 6 CPUs (2 tasks and 3 CPUs per task with no overcommitment). Allocate the 6 CPUs as consumable resources from a single node in the default partition.

slurm.conf options:

```
SelectType=select/cons_tres
SelectTypeParameters=CR_Core
```

Comments:

The SelectType configuration options define cores as consumable resources. The --nodes=1-1 srun option restricts the job to a single node. The following table shows a possible pattern of allocation for this job.

Nodename	n0	n1	n2
Number of Allocated CPUs	6	0	0
Number of Tasks	2	0	0

EXAMPLE 3: CONSUMABLE RESOURCES WITH BALANCED ALLOCATION ACROSS NODES

A job requires 9 CPUs (3 tasks and 3 CPUs per task with no overcommitment). Allocate 3 CPUs from each of the 3 nodes in the default partition.

slurm.conf options:

```
SelectType=select/cons_tres
SelectTypeParameters=CR_CorePartitionName=hypernode Nodes=n3 State=UP
```

Comments:

The options specify the following conditions for the job: 3 tasks, 3 unique CPUs per task, using exactly 3 nodes. To satisfy these conditions, Slurm must allocate 3 CPUs from each node. The following table shows the allocation for this job.

Nodename	n0	n1	n2
Number of Allocated CPUs	3	3	3
Number of Tasks	1	1	1

EXAMPLE 4: CONSUMABLE RESOURCES WITH MINIMIZATION OF RESOURCE FRAGMENTATION

A job requires 12 CPUs (12 tasks and 1 CPU per task with no overcommitment). Allocate CPUs using the minimum number of nodes and the minimum number of sockets required for the job in order to minimize fragmentation of allocated/unallocated CPUs in the cluster.

slurm.conf options:

```
SelectType=select/cons_tres  
SelectTypeParameters=CR_Core,CR_CORE_DEFAULT_DIST_BLOCK.
```

Comments:

The default allocation method across nodes is block. This minimizes the number of nodes used for the job. The configuration option CR_CORE_DEFAULT_DIST_BLOCK sets the default allocation method within a node to block. This minimizes the number of sockets used for the job within a node. The combination of these two methods causes Slurm to allocate the 12 CPUs using the minimum required number of nodes (2 nodes) and sockets (3 sockets). The following table shows a possible pattern of allocation for this job.

Nodename	n0	n1	n2			
Socket id	0	1	0	1	0	1
Number of Allocated CPUs	4	4	4	0	0	0
Number of Tasks	8	4	0			

EXAMPLE 5: CONSUMABLE RESOURCES WITH CYCLIC DISTRIBUTION OF TASKS TO NODES

A job requires 12 CPUs (6 tasks and 2 CPUs per task with no overcommitment). Allocate 6 CPUs each from 2 nodes in the default partition. Distribute tasks to nodes cyclically.

slurm.conf options:

```
SelectType=select/cons_tres  
SelectTypeParameters=CR_Core
```

Command line:

```
srun --nodes=2-2 --ntasks-per-node=3 --distribution=cyclic  
--ntasks=6 --cpus-per-task=2 ...
```

Comments:

The options specify the following conditions for the job: 6 tasks, 2 unique CPUs per task, using exactly 2 nodes, and with 3 tasks per node. To satisfy these conditions, Slurm must allocate 6 CPUs from each of the 2 nodes. The --distribution=cyclic option causes the tasks to be distributed to the nodes in a round-robin fashion. The following table shows a possible pattern of allocation and distribution for this job.

Nodename	n0	n1	n2
Number of Allocated CPUs	6	6	0
Number of Tasks	3	3	0
Distribution of Tasks to Nodes, by Task id	0 2 4	1 3 5	-

EXAMPLE 6: CONSUMABLE RESOURCES WITH DEFAULT ALLOCATION AND PLANE DISTRIBUTION OF TASKS TO NODES

A job requires 16 CPUs (8 tasks and 2 CPUs per task with no overcommitment). Use all 3 nodes in the default partition. Distribute tasks to each node in blocks of two in a round-robin fashion.

slurm.conf options:

```
SelectType=select/cons_tres  
SelectTypeParameters=CR_Core
```

Command Line:

```
srun --nodes=3-3 --distribution=plane=2 --ntasks=8 --cpus-per-task=2 ...
```

Comments:

The options specify the following conditions for the job: 8 tasks, 2 unique CPUs per task, using all 3 nodes in the partition. To satisfy these conditions using the default allocation method across nodes (block), Slurm allocates 8 CPUs from the first node, 6 CPUs from the second node and 2 CPUs from the third node. The --distribution=plane=2 option causes Slurm to distribute tasks in blocks of two to each of the nodes in a round-robin fashion, subject to the number of CPUs allocated on each node. So, for example, only 1 task is distributed to the third node because only 2 CPUs were allocated on that node and each task requires 2 CPUs. The following table shows a possible pattern of allocation and distribution for this job.

Nodename	n0	n1	n2
Number of Allocated CPUs	8	6	2
Number of Tasks	4	3	1
Distribution of Tasks to Nodes, by Task id	0 1	2 3	4
	5 6	7	

EXAMPLE 7: CONSUMABLE RESOURCES WITH OVERCOMMITMENT OF CPUS TO TASKS

A job has 20 tasks. Run the job in a single node.

slurm.conf options:

```
SelectType=select/cons_tres  
SelectTypeParameters=CR_Core
```

Command Line:

```
srun --nodes=1-1 --ntasks=20 --overcommit ...
```

Comments:

The --overcommit option allows the job to run in only one node by overcommitting CPUs to tasks. The following table shows a possible pattern of allocation and distribution for this job.

Nodename	n0	n1	n2
Number of Allocated CPUs	8	0	0
Number of Tasks	20	0	0
Distribution of Tasks to Nodes, by Task id	0 - 19	-	-

EXAMPLE 8: CONSUMABLE RESOURCES WITH RESOURCE SHARING BETWEEN JOBS

2 jobs each require 6 CPUs (6 tasks per job with no overcommitment). Run both jobs simultaneously in a single node.

slurm.conf options:

```
SelectType=select/cons_tres
SelectTypeParameters=CR_Core
```

Command Line:

```
srun --nodes=1-1 --nodelist=n0 --ntasks=6 --oversubscribe ...
srun --nodes=1-1 --nodelist=n0 --ntasks=6 --oversubscribe ...
```

Comments:

The CR_CPU configuration option enables the allocation of only one thread per core. The --hint=nomultithread srun option causes Slurm to allocate only one thread from each core to this job. The following table shows a possible pattern of allocation for this job.

Nodename	n3															
Socket id	0								1							
Core id	0	1	2	3	0	1	2	3	0	1	2	3	4	5	6	7
CPU id	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Number of Allocated CPUs	4								4							
Allocated CPU ids	0 2 4 6								8 10 12 14							

EXAMPLE 9: CONSUMABLE RESOURCES ON MULTITHREADED NODE, ALLOCATING ONLY ONE THREAD PER CORE

A job requires 8 CPUs (8 tasks with no overcommitment). Run the job on node n3, allocating only one thread per core.

slurm.conf options:

```
SelectType=select/cons_res
SelectTypeParameters=CR_CPU
```

Command line:

```
srun --partition=hypernode --ntasks=8 --hint=nomultithread ...
```

Comments:

The CR_CPU configuration option enables the allocation of only one thread per core. The --hint=nomultithread srun option causes Slurm to allocate only one thread from each core to this job. The following table shows a possible pattern of allocation for this job.

Nodename	n3											
Socket id	0							1				
Core id	0	1	2	3	4	5	6	7	0	1	2	3
CPU id	0	1	2	3	4	5	6	7	8	9	10	11
Number of Allocated CPUs	4							4				
Allocated CPU ids	0 2 4 6							8 10 12 14				

EXAMPLE 10: CONSUMABLE RESOURCES WITH TASK AFFINITY AND CORE BINDING

A job requires 6 CPUs (6 tasks with no overcommitment). Run the job in a single node in the default partition. Apply core binding to each task.

slurm.conf options:

```
SelectType=select/cons_tres
SelectTypeParameters=CR_Core
TaskPlugin=task/affinity
TaskPluginParam=sched
```

Command line:

```
srun --nodes=1-1 --ntasks=6 --cpu-bind=cores ...
```

Comments:

Using the default allocation method within nodes (cyclic), Slurm allocates 3 CPUs on each socket of 1 node. Using the default distribution method within nodes (cyclic), Slurm distributes and binds each task to an allocated core in a round-robin fashion across the sockets. The following table shows a possible pattern of allocation, distribution and binding for this job. For example, task id 2 is bound to CPU id 1.

Nodename		n0							
Socket id		0			1				
Number of Allocated CPUs		3			3				
Allocated CPU ids		0 1 2			4 5 6				
Binding of Tasks to CPUs	CPU id	0	1	2	3	4	5	6	7
	Task id	0	2	4	-	1	3	5	-

EXAMPLE 11: CONSUMABLE RESOURCES WITH TASK AFFINITY AND SOCKET BINDING, CASE 1

A job requires 6 CPUs (6 tasks with no overcommitment). Run the job in a single node in the default partition. Apply socket binding to each task.

slurm.conf options:

```
SelectType=select/cons_tres
SelectTypeParameters=CR_Core
TaskPlugin=task/affinity
TaskPluginParam=sched
```

Command Line:

```
srun --nodes=1-1 --ntasks=6 --cpu-bind=sockets ...
```

Comments:

Using the default allocation method within nodes (cyclic), Slurm allocates 3 CPUs on each socket of 1 node. Using the default distribution method within nodes (cyclic), Slurm distributes and binds each task to all of the allocated CPUs in one socket in a round-robin fashion across the sockets. The following table shows a possible pattern of allocation, distribution and binding for this job. For example, task ids 1, 3 and 5 are all bound to CPU ids 4, 5 and 6.

Nodename		n0							
Socket id		0				1			
Number of Allocated CPUs		3				3			
Allocated CPU ids		0 1 2				4 5 6			
Binding of Tasks to CPUs	CPU id	0	1	2	3	4	5	6	7
	Task ids	0 2 4	-	-	1 3 5	-	-	-	-

EXAMPLE 12: CONSUMABLE RESOURCES WITH TASK AFFINITY AND SOCKET BINDING, CASE 2

A job requires 6 CPUs (2 tasks with 3 cpus per task and no overcommitment). Run the job in a single node in the default partition. Allocate cores using the block allocation method. Distribute cores using the block distribution method. Apply socket binding to each task.

slurm.conf options:

```
SelectType=select/cons_tres
SelectTypeParameters=CR_Core,CR_CORE_DEFAULT_DIST_BLOCK
TaskPlugin=task/affinity
TaskPluginParam=sched
```

Command line:

```
srun --nodes=1-1 --ntasks=2 --cpus-per-task=3 --cpu-bind=sockets
--distribution=block:block ...
```

Comments:

Using the block allocation method, Slurm allocates 4 CPUs on one socket and 2 CPUs on the other socket of one node. Using the block distribution method within nodes, Slurm distributes 3 CPUs to each task. Applying socket binding, Slurm binds each task to all allocated CPUs in all sockets in which the task has a distributed CPU. The following table shows a possible pattern of allocation, distribution and binding for this job. In this example, using the block allocation method CPU ids 0-3 are allocated on socket id 0 and CPU ids 4-5 are allocated on socket id 1. Using the block distribution method, CPU ids 0-2 were distributed to task id 0, and CPU ids 3-5 were distributed to

task id 1. Applying socket binding, task id 0 is therefore bound to the allocated CPUs on socket 0, and task id 1 is bound to the allocated CPUs on both sockets.

Nodename		n0						
Socket id		0			1			
Number of Allocated CPUs		4			2			
Allocated CPU ids		0 1 2 3			4 5			
Binding of Tasks to CPUs	CPU id	0	1	2	3	4	5	6
	Task ids	0 1		1		-		

EXAMPLE 13: CONSUMABLE RESOURCES WITH TASK AFFINITY AND SOCKET BINDING, CASE 3

A job requires 6 CPUs (2 tasks with 3 cpus per task and no overcommitment). Run the job in a single node in the default partition. Allocate cores using the block allocation method. Distribute cores using the cyclic distribution method. Apply socket binding to each task.

slurm.conf options:

```
SelectType=select/cons_tres
SelectTypeParameters=CR_Core,CR_CORE_DEFAULT_DIST_BLOCK
TaskPlugin=task/affinity
TaskPluginParam=sched
```

Command Line:

```
srun --nodes=1-1 --ntasks=2 --cpus-per-task=3 --cpu-bind=sockets
--distribution=block:cyclic ...
```

Comments:

Using the block allocation method, Slurm allocates 4 CPUs on one socket and 2 CPUs on the other socket of one node. Using the cyclic distribution method within nodes, Slurm distributes 3 CPUs to each task. Applying socket binding, Slurm binds each task to all allocated CPUs in all sockets in which the task has a distributed CPU. The

following table shows a possible pattern of allocation, distribution and binding for this job. In this example, using the block allocation method CPU ids 0-3 are allocated on socket id 0 and CPU ids 4-5 are allocated on socket id 1. Using the cyclic distribution method, CPU ids 0, 1 and 4 were distributed to task id 0, and CPU ids 2, 3 and 5 were distributed to task id 1. Applying socket binding, both tasks are therefore bound to the allocated CPUs on both sockets.

Nodename		n0							
Socket id		0				1			
Number of Allocated CPUs		4				2			
Allocated CPU ids		0 1 2 3				4 5			
Binding of Tasks to CPUs	CPU id	0	1	2	3	4	5	6	7
	Task ids	0 1		0 1	-				

EXAMPLE 14: CONSUMABLE RESOURCES WITH TASK AFFINITY AND CUSTOMIZED ALLOCATION AND DISTRIBUTION

A job requires 18 CPUs (18 tasks with no overcommitment). Run the job in the default partition. Allocate 6 CPUs on each node using block allocation within nodes. Use cyclic distribution of tasks to nodes and block distribution of tasks for CPU binding.

slurm.conf options:

```
SelectType=select/cons_tres
SelectTypeParameters=CR_Core,CR_CORE_DEFAULT_DIST_BLOCK
TaskPlugin=task/affinity
TaskPluginParam=sched
```

Command line:

```
srun --nodes=3-3 --ntasks=18 --ntasks-per-node=6
--distribution=cyclic:block --cpu-bind=cores ...
```

Comments:

This example shows the use of task affinity with customized allocation of CPUs and distribution of tasks across nodes and within nodes for binding. The srun options specify the following conditions for the job: 18 tasks, 1 unique CPU per task, using all 3 nodes in the partition, with 6 tasks per node. The CR_CORE_DEFAULT_DIST_BLOCK configuration option specifies block allocation within nodes. To satisfy these conditions, Slurm allocates 6 CPUs on each node, with 4 CPUs allocated on one socket and 2 CPUs on the other socket. The --distribution=cyclic:block option specifies cyclic distribution of tasks to nodes and block distribution of tasks to CPUs within nodes for binding. The following table shows a possible pattern of allocation, distribution and binding for this job. For example, task id 10 is bound to CPU id 3 on node n1.

Nodename		n0				n1				n2											
Socket id		0		1		0		1		0		1									
Number of Allocated CPUs		4		2		4		2		4		2									
Allocated CPU ids		0 1 2 3 4 5						0 1 2 3 4 5													
Number of Tasks		6						6													
Distribution of Tasks to Nodes, by		0						1													
Task id		3						4													
		6						7													
		9						10													
		12						13													
		15						16													
		17						18													
Binding of Tasks to CPUs	CPU id	0	1	2	3	4	5	6	7	0	1	2	3								
	Task id	0	3	6	9	12	15	-	-	1	4	7	10								
		13	16	-	-	2	5	8	11	14	17	-	-								

EXAMPLE 15: CONSUMABLE RESOURCES WITH TASK AFFINITY TO OPTIMIZE THE PERFORMANCE OF A MULTI-TASK, MULTI-THREAD JOB

A job requires 9 CPUs (3 tasks and 3 CPUs per task with no overcommitment). Run the job in the default partition, managing the CPUs to optimize the performance of the job.

slurm.conf options:

```
SelectType=select/cons_tres
SelectTypeParameters=CR_Core,CR_CORE_DEFAULT_DIST_BLOCK
TaskPlugin=task/affinity
```

```
TaskPluginParam=sched
```

Command line:

```
srun --ntasks=3 --cpus-per-task=3 --ntasks-per-node=1 --cpu-bind=cores ...
```

Comments:

To optimize the performance of this job, the user wishes to allocate 3 CPUs from each of 3 sockets and bind each task to the 3 CPUs in a single socket. The SelectTypeParameters configuration option specifies a consumable resource type of cores and block allocation within nodes. The TaskPlugin and TaskPluginParam configuration options enable task affinity. The srun options specify the following conditions for the job: 3 tasks, with 3 unique CPUs per task, with 1 task per node. To satisfy these conditions, Slurm allocates 3 CPUs from one socket in each of the 3 nodes in the default partition. The --cpu-bind=cores option causes Slurm to bind each task to the 3 allocated CPUs on the node to which it is distributed. The following table shows a possible pattern of allocation, distribution and binding for this job. For example, task id 2 is bound to CPU ids 0, 1 and 2 on socket id 0 of node n2.

Nodename		n0				n1				n2									
Socket id		0		1		0		1		0		1							
Number of Allocated CPUs		3		0		3		0		3		0							
Allocated CPU ids		0 1 2				0 1 2				0 1 2									
Number of Tasks		1				1				1									
Distribution of Tasks to Nodes, by Task id		0				1				2									
Binding of Tasks to CPUs	CPU id	0	1	2	3	4	5	6	7	0	1	2	3						
	Task id	0	-	-	-	-	-	-	-	2	-	-	-						

EXAMPLE 16: CONSUMABLE RESOURCES WITH TASK CGROUP

A job requires 6 CPUs (6 tasks with no overcommitment). Run the job in a single node in the default partition.

slurm.conf options:

```
SelectType=select/cons_tres
SelectTypeParameters=CR_Core
TaskPlugin=task/cgroup
```

cgroup.conf options

```
ConstrainCores=yes
```

Command line:

```
srun --nodes=1-1 --ntasks=6 ...
```

Comments:

The task/cgroup plugin currently supports only the block method for allocating cores within nodes. Slurm distributes tasks to the cores but without cpu binding, each task has access to all the allocated CPUs. The following table shows a possible pattern of allocation, distribution and binding for this job.

Nodename		n0							
Socket id		0				1			
Number of Allocated CPUs		4				2			
Allocated CPU ids		0 1 2 3				4 5			
Binding of Tasks to CPUs	CPU id	0	1	2	3	4	5	6	7
	Task id	0	1	2	3	4	5	-	-

SlurmctldParameters

The Slurm controller, when started, can load with specific switches which determine certain functionality. A newer feature provides for the capability of limiting the number of RPC calls that can be handles by the controller. With clients making repeated queries via sinfo, squeue, and other client commands through the linux `watch` command to the controller, it may overwhelm the schedule and decrease scheduling performance. To that end, being able to limit the number of requests may improve scheduler performance.

These switches are added to the `SlurmdParameters` setting in `slurm.conf`.

`rl_enable`: Enable per-user RPC rate-limiting support. Client-commands will be told to back off and sleep for a second once the limit has been reached. This is implemented as a "token bucket", which permits a certain degree of "bursty" RPC load from an individual user before holding them to a steady-state RPC load established by the refill period and rate.

`rl_bucket_size`: Size of the token bucket. This permits a certain amount of RPC burst from a user before the steady-state rate limit takes effect. The default value is 30.

`rl_refill_period`: How frequently, in seconds, in which additional tokens are added to each user bucket. The default value is 1.

`rl_refill_rate`: How many tokens to add to the bucket on each period. The default value is 2.

`rl_table_size`: Number of entries in the user hash-table. Recommended value should be at least twice the number of active user accounts on the system. The default value is 8192.

scontrol reconfigure

This command instructs all Slurm daemons to re-read the configuration file. This command does not restart the daemons. This mechanism can be used to modify configuration parameters set in slurm.conf. The Slurm controller (slurmctld) forwards the request to all other daemons (slurmd daemon on each compute node).

Running jobs continue execution. Most configuration parameters can be changed by just running this command; however, there are parameters that require a restart of the relevant Slurm daemons. Parameters requiring a restart will be noted in the slurm.conf(5) man page. The slurmctld daemon and all slurmd daemons must also be restarted if nodes are added to or removed from the cluster.

In the latest version of the slurm controller (V23.02), the command scontrol reconfigure and sending a SIGHUP to the slurmctld behave the same. If you were using SIGHUP as a 'lighter' scontrol reconfigure to rotate logs please update your scripts to use SIGUSR2 instead.

Scheduling Exercises

In this set of labs, you will learn how to configure the scheduler for optimal performance.

Exercise 1: Modify the Backfill Scheduler Policy by changing the default depth:

1. As root, edit the /etc/slurm/slurm.conf file:

Add the following to the file:

```
SchedulerParameters=bf_max_job_test=1
```

2. Reload the controller configuration:

```
scontrol reconfigure
```

3. Open another terminal window

4. As fred, run the following in it:

```
watch "squeue ; sinfo -Nel -pdebug"
```

5. In another terminal window as fred, submit a sleep job as follows :

```
sbatch -N1 --mem=1000 --exclusive -t60 --wrap="sleep 3600"
```

You should see the job in the other window:

JOBID	PARTITION	NAME	USER	ST	TIME	NODES	NODELIST (REASON)
1	debug	wrap	fred	R	0:10	1	node00
Wed Nov 27 23:34:29 2019							
NODELIST	NODES	PARTITION	STATE	CPUS	S:C:T	MEMORY	TMP_DISK WEIGHT AVAIL_FE REASON
node00	1	debug*	allocated	4	1:4:1	16011	0 1 (null) none
node01	1	debug*	idle	4	1:4:1	16011	0 1 (null) none
node02	1	debug*	idle	4	1:4:1	16011	0 1 (null) none
node03	1	debug*	idle	4	1:4:1	16011	0 1 (null) none
node04	1	debug*	idle	4	1:4:1	16011	0 1 (null) none
node05	1	debug*	idle	4	1:4:1	16011	0 1 (null) none
node06	1	debug*	idle	4	1:4:1	16011	0 1 (null) none
node07	1	debug*	idle	4	1:4:1	16011	0 1 (null) none
node08	1	debug*	idle	4	1:4:1	16011	0 1 (null) none
node09	1	debug*	idle	4	1:4:1	16011	0 1 (null) none

So the long job is running.

Notice that the job is running on node00, leaving node[01-09] available for workload.

6. Submit another job as fred as follows:

```
sbatch -N10 --mem=1000 --exclusive -t60 --wrap="sleep 3600"
```

NOTE: This job will NOT run because there are only 9 nodes available and it requires 10. So it should go queued for lack of Resources.

The output should now look like this:

JOBID	PARTITION	NAME	USER	ST	TIME	NODES	NODELIST (REASON)			
2	debug	wrap	fred	PD	0:00	10	(Resources)			
1	debug	wrap	fred	R	1:59	1	node00			
Wed Nov 27 23:36:18 2019										
NODELIST NODES PARTITION STATE CPUS S:C:T MEMORY TMP_DISK WEIGHT AVAIL_FE										
REASON										
node00	1	debug*	allocated	4	1:4:1	16011	0	1 (null) none		
node01	1	debug*	planned	4	1:4:1	16011	0	1 (null) none		
node02	1	debug*	planned	4	1:4:1	16011	0	1 (null) none		
node03	1	debug*	planned	4	1:4:1	16011	0	1 (null) none		
node04	1	debug*	planned	4	1:4:1	16011	0	1 (null) none		
node05	1	debug*	planned	4	1:4:1	16011	0	1 (null) none		
node06	1	debug*	planned	4	1:4:1	16011	0	1 (null) none		
node07	1	debug*	planned	4	1:4:1	16011	0	1 (null) none		
node08	1	debug*	planned	4	1:4:1	16011	0	1 (null) none		
node09	1	debug*	idle	4	1:4:1	16011	0	1 (null) none		

With JobID 1 running, and JobID 2 Pending (Status=PD) based on available resources.

NOTE: Since bf_max_job_test=1 is set, the backfill scheduler will stop at JobID4 and not evaluate any other jobs for backfill.

NOTE: The node state of “planned” means the node is planned by the backfill scheduler for a higher priority job.

7. Submit 3 short job **as fred** as follows (they will NOT run on nodes 01-3, even though there are available resources):

```
sbatch -N1 --mem=1000 --exclusive -t10 --wrap="sleep 600"
sbatch -N1 --mem=1000 --exclusive -t10 --wrap="sleep 600"
sbatch -N1 --mem=1000 --exclusive -t10 --wrap="sleep 600"
```

You will see this in the other window:

JOBID	PARTITION	NAME	USER	ST	TIME	NODES	NODELIST (REASON)			
2	debug	wrap	fred	PD	0:00	10	(Resources)			
3	debug	wrap	fred	PD	0:00	1	(Priority)			
4	debug	wrap	fred	PD	0:00	1	(Priority)			
5	debug	wrap	fred	PD	0:00	1	(Priority)			
1	debug	wrap	fred	R	3:48	1	node00			
Wed Nov 27 23:38:07 2019										
NODELIST NODES PARTITION STATE CPUS S:C:T MEMORY TMP_DISK WEIGHT AVAIL_FE REASON										
node00	1	debug*	allocated	4	1:4:1	16011	0	1 (null) none		
node01	1	debug*	planned	4	1:4:1	16011	0	1 (null) none		
node02	1	debug*	planned	4	1:4:1	16011	0	1 (null) none		
node03	1	debug*	planned	4	1:4:1	16011	0	1 (null) none		
node04	1	debug*	planned	4	1:4:1	16011	0	1 (null) none		
node05	1	debug*	planned	4	1:4:1	16011	0	1 (null) none		
node06	1	debug*	planned	4	1:4:1	16011	0	1 (null) none		
node07	1	debug*	planned	4	1:4:1	16011	0	1 (null) none		
node08	1	debug*	planned	4	1:4:1	16011	0	1 (null) none		
node09	1	debug*	planned	4	1:4:1	16011	0	1 (null) none		

Explanation: Since we are forcing the scheduler to only test 1 of all the jobs in the queue (in priority order), only 1 job can be considered for backfill: In my case, job 1.

8. One way to know this is to issue squeue --start:

```
squeue --start -o "%i %.9P %.8u %.2t %.6D %R%S"
```

JOBID	PARTITION	USER	ST	NODES	NODELIST (REASON)	START_TIME
3	debug	fred	PD	1	(Priority)	N/A
4	debug	fred	PD	1	(Priority)	N/A
5	debug	fred	PD	1	(Priority)	N/A
2	debug	fred	PD	10	(Resources)	2019-11-28T00:34:19

NOTE: The only reason JobID 2 has an estimated time start is because it was the only job evaluated for backfill. If you change the value to >1, then the scheduler will evaluate more idle jobs for backfill.

ALSO NOTE: If you change the bf_max_job_test=4, then all the jobs will be evaluated for backfill and start running

- As root, edit the `/etc/slurm/slurm.conf` file and add the following to the file:

`SchedulerParameters=bf_max_job_test=4`

- Cause the scheduler to re-read its configuration file:

`scontrol reconfigure`

- After a scheduling cycle, you will now see all jobs running, because by default the scheduler evaluates past the top jobs in the queue.

You should see this in the other window:

JOBID	PARTITION	NAME	USER	ST	TIME	NODES	NODELIST (REASON)
3	debug	wrap	fred	PD	0:00	10	(Resources)
2	debug	wrap	fred	R	7:38	1	node00
4	debug	wrap	fred	R	0:32	1	node01
5	debug	wrap	fred	R	0:32	1	node02
6	debug	wrap	fred	R	0:32	1	node03
Wed Nov 27 23:41:57 2019							
NODELIST	NODES	PARTITION	STATE	CPUS	S:C:T	MEMORY	TMP_DISK WEIGHT AVAIL_FE REASON
node00	1	debug*	allocated	4	1:4:1	16011	0 1 (null) none
node01	1	debug*	allocated	4	1:4:1	16011	0 1 (null) none
node02	1	debug*	allocated	4	1:4:1	16011	0 1 (null) none
node03	1	debug*	allocated	4	1:4:1	16011	0 1 (null) none
node04	1	debug*	planned	4	1:4:1	16011	0 1 (null) none
node05	1	debug*	planned	4	1:4:1	16011	0 1 (null) none
node06	1	debug*	planned	4	1:4:1	16011	0 1 (null) none
node07	1	debug*	planned	4	1:4:1	16011	0 1 (null) none
node08	1	debug*	planned	4	1:4:1	16011	0 1 (null) none
node09	1	debug*	planned	4	1:4:1	16011	0 1 (null) none

Cleanup

- Reset the Docker Slurm cluster environment back to its original, pristine state by exiting the login Docker container back to the Ubuntu AWS prompt. And as `ubuntu`:

`cd ~/docker-scale-out`

```
make clean && make
```

Exercise 2: Submitting jobs to GPUs

In this lab, you will configure simulated GPUs and submit jobs to them. You will need to modify Nodes *node08* and *node09* so they will belong to a new partition with the appropriate entries. Also you will need to remove them from the *debug* partition

1. As root, edit and **replace** the contents of `/etc/slurm/nodes.conf` with the following:

```
NodeName=DEFAULT CPUs=4 Boards=1 SocketsPerBoard=1 CoresPerSocket=4 ThreadsPerCore=1 RealMemory=15979
NodeName=node00 NodeAddr=2001:db8:1:1::5:10
NodeName=node01 NodeAddr=2001:db8:1:1::5:11
NodeName=node02 NodeAddr=2001:db8:1:1::5:12
NodeName=node03 NodeAddr=2001:db8:1:1::5:13
NodeName=node04 NodeAddr=2001:db8:1:1::5:14
NodeName=node05 NodeAddr=2001:db8:1:1::5:15
NodeName=node06 NodeAddr=2001:db8:1:1::5:16
NodeName=node07 NodeAddr=2001:db8:1:1::5:17
NodeName=node08 NodeAddr=2001:db8:1:1::5:18 Gres=gpu:v100:2
NodeName=node09 NodeAddr=2001:db8:1:1::5:19 Gres=gpu:k80:1
```

NOTE: The NodeName=DEFAULT... should be on ONE LINE

NOTE: VERIFY THE CORRECT AMOUNT OF MEMORY FOR YOUR VM

2. **READ CAREFULLY->** Edit `/etc/slurm/slurm.conf` and add new partition called "gpu", and add "node08" and "node09" to it with billing weight values for the GPUs (THE FOLLOWING ALL GOES ON ONE LINE):

```
PartitionName=gpu Nodes=node[08-09] MaxTime=INFINITE State=UP
TRESBillingWeights="CPU=1.0,Mem=0.25G,GRES/gpu:v100=4.0,GRES/gpu:k80=2.0"
```

3. **While still editing the file, remove "node08" and "node09" from the debug partition by modifying the debug partition entry...**

```
PartitionName=debug Nodes=node[00-07] Default=YES MaxTime=INFINITE State=UP
```

4. ...confirm that the entry for "GresTypes" identifier is in the config...

```
GresTypes=gpu
```

5. ...replace the "AccountingStorageTRES" Plugin definition for the GPUs with the following...

```
AccountingStorageTRES=GRES/gpu,GRES/gpu:k80,GRES/gpu:v100
```

6. ...confirm the "SelectType" plugin to be...

```
SelectType=select/cons_tres
SelectTypeParameters=CR CORE Memory
```

7. **Save and exit the slurm.conf file.**

8. **Edit and replace the entries in `/etc/slurm/gres.conf` with the following:**

```
Name=gpu Type=v100 File=/proc/self/fd/[0-1] Cores=0-1  
Name=gpu Type=k80 File=/proc/self/fd/2 Cores=1
```

9. Save and exit the gres.conf file.

10. Restart the controller:

```
/lab_scripts/restart.sh
```

11. Confirm the GPUs have been added by executing the following:

```
scontrol show node node[08-09] | grep -i cfgtres | grep gpu
```

You should see:

```
CfgTRES=cpu=4,mem=15979M,billing=15,gres/gpu=2,gres/gpu:v100=2  
CfgTRES=cpu=4,mem=15979M,billing=9,gres/gpu=1,gres/gpu:k80=1
```

12. Now that you have 2 GPU nodes, try submitting a job to them **as fred**:

```
srun -N1 --mem=1000 -pgpu --gres=gpu:v100:2 hostname
```

You should see:

```
node08
```

13. Also submit this **as fred**:

```
srun -N1 --mem=1000 -pgpu --gres=gpu:k80:1 hostname
```

You should see:

```
node09
```

14. View how Slurm charged for the CPUs and GPUs:

```
sacct -X --format=jobid%5,jobname%9,partition,account%8,alloccpus,allocctres%50
```

You should see:

JobID	JobName	Partition	Account	AllocCPUS	AllocTRES
1	hostname	gpu	bedrock	1	billing=12,cpu=1,gres/gpu:v100=2,gres/gpu=2,mem=1+
2	hostname	gpu	bedrock	1	billing=6,cpu=1,gres/gpu:k80=1,gres/gpu=1,mem=159+

Notice the billing rate difference between different types of GPUs (12 for v100 jobs, and 6 for k80 jobs):

NOTE: If these were real GPUs, then you could do something like the following to report their status:

```
srun -N1 --mem=1000 --gres=gpu:v100:2 nvidia-smi -q  
srun -N1 --mem=1000 --gres=gpu:k80:1 nvidia-smi -q
```

Cleanup

1. Reset the Docker Slurm cluster environment back to its original, pristine state by exiting the login Docker container back to the Ubuntu AWS prompt. And **as ubuntu**:

```
cd ~/docker-scale-out  
make clean && make
```

Slurm Job Prioritization Discussion

There are several references to priority in the Slurm world: Node priority, Job priority, and others. In this section, we will discuss the concept of priority around the jobs.

By default, Slurm assigns job priority on a First In, First Out (FIFO) basis. FIFO scheduling was the option of choice when Slurm is controlled by an external scheduler. But even if you are looking for simple FIFO, you can still configure Slurm that way.

As with many subsystems in Slurm, the order of running jobs can be based on a module. The priority modules available to the scheduler are the following:

- **Priority/basic** - This provides rudimentary FIFO scheduling (first-in, first-out)
- **Priority/multifactor** - This sets priority based upon job age, queue/partition, size, QOS and fairshare

The PriorityType parameter in the *slurm.conf* file selects the priority plugin. The default value for this variable is "priority/basic" which enables simple FIFO scheduling.

The Multi-factor Job Priority plugin provides a very versatile facility for ordering the queue of jobs waiting to be scheduled. To enable the plugin, enter it in the *slurm.conf*:

```
PriorityType=priority/multifactor
```

Job Prioritization Factors

There are six factors in the Multi-factor Job Priority plugin that influence job priority:

- **Age** The length of time a job has been waiting in the queue, eligible to be scheduled
- **Fair-share** The difference between the portion of the computing resource that has been promised and the amount of resources that has been consumed
- **Job size** The number of nodes or CPUs a job is allocated
- **Partition** A factor associated with each node partition
- **QOS** A factor associated with each Quality Of Service
- **TRES** Each TRES Type has its own factor for a job which represents the number of requested/allocated TRES Type in a given partition.

Slurm uses a plugin method for calculating job priority. When the plugin is active, it can assign priority to jobs based on several configurable factors.

The job priority number given to a job is represented by a 32-bit integer. The factors that make up the priority are floating point numbers between 0 and 1.0.

Additionally, a weight can be assigned to each of the above factors. This provides the ability to enact a policy that blends a combination of any of the above factors in any portion desired. For example, a site could configure fair-share to be the dominant factor (say 70%), set the job size and the age factors to each contribute 15%, and set the partition and QOS influences to zero.

The job priority value is calculated as follows:

```

Job_priority =
(PriorityWeightAge)      * (age_factor)      +
(PriorityWeightFairshare) * (fair-share_factor) +
(PriorityWeightJobSize)   * (job_size_factor)  +
(PriorityWeightPartition) * (partition_factor) +
(PriorityWeightQOS)       * (QOS_factor)
+
SUM(TRES_weight_cpu      * TRES_factor_cpu    +
    TRES_weight_<type> * TRES_factor_<type>  +
    ...)
```

All of the factors in this formula are floating point numbers that range from 0.0 to 1.0. The weights are unsigned, 32 bit integers. The job's priority is an integer that ranges between 0 and 4294967295. The larger the number, the higher the job will be positioned in the queue, and therefore, the sooner the job will be scheduled. A job's priority, and hence its order in the queue, can vary over time. For example, the longer a job sits in the queue, the higher its priority will grow when the age_weight is non-zero.

One thing to remember on Slurm priority, is that it's normalized.

The weight values should be high enough to get a good set of significant digits since all the factors are floating point numbers from 0.0 to 1.0. For example, one job could have a fair-share factor of .59534 and another job could have a fair-share factor of .50002. If the fair-share weight is only set to 10, both jobs would have the same fair-share priority. Therefore, set the weights high enough to avoid this scenario, starting around 1000 or so for those factors you want to make predominant.

Age Factor

The age factor represents the length of time a job has been sitting in the queue and eligible to run. In general, the longer a job waits in the queue, the larger its age factor grows. However, the age factor for a dependent job will not change while it waits for the job it depends on to complete. Also, the age factor will not change when scheduling is withheld for a job whose node or time limits exceed the cluster's current limits.

At some configurable length of time (PriorityMaxAge), the age factor will max out to 1.0.

The PriorityMaxAge is the job age at which AgeFactor reaches 1.0 (The maximum). The default for this policy is 7 days. Meaning, after 7 days, all the jobs would get the same age-based priority

Job Size Factor

The job size factor correlates to the number of nodes or CPUs the job has requested. This factor can be configured to favor larger jobs or smaller jobs based on the state of the PriorityFavorSmall boolean in the slurm.conf file. When PriorityFavorSmall is NO, the larger the job, the greater its job size factor will be. A job that requests all the nodes on the machine will get a job size factor of 1.0. When the PriorityFavorSmall Boolean is YES, the single node job will receive the 1.0 job size factor.

The PriorityFlags value of SMALL_RELATIVE_TO_TIME alters this behavior as follows. The job size in CPUs is divided by the time limit in minutes. The result is divided by the total number of CPUs in the system. Thus a full-system job with a time limit of one will receive a job size factor of 1.0, while a tiny job with a large time limit will receive a job size factor close to 0.0.

Partition Factor

Each node partition can be assigned an integer priority. The larger the number, the greater the job priority will be for jobs that request to run in this partition. This priority value is then normalized to the highest priority of all the partitions to become the partition factor.

Quality of Service (QOS) Factor

Each QOS can be assigned an integer priority. The larger the number, the greater the job priority will be for jobs that request this QOS. This priority value is then normalized to the highest priority of all the QOS's to become the QOS factor.

```
Partition normal
Job 1 QOS high
Job 2 QOS high
Job 3 QOS normal
Job 4 QOS normal
Job 5 QOS low
```

Giving a QOS a higher priority is what prioritization was originally designed for. In fact, Quality of Service is how organizations can adhere to standards by charging a premium for higher-level priority.

TRES Factors

Each TRES Type has its own priority factor for a job which represents the amount of TRES Type requested/allocated in a given partition. For global TRES Types, such as Licenses and Burst Buffers, the factor represents the number of TRES Type requested/allocated in the whole system. The more a given TRES Type is requested/allocated on a job, the greater the job priority will be for that job.

Fair-share Factor

NOTE: Computing the fair-share factor requires the installation and operation of the Slurm Accounting Database to provide the assigned shares and the consumed, computing resources described below.

The fair-share component to a job's priority influences the order in which a user's queued jobs are scheduled to run based on the portion of the computing resources they have been allocated and the resources their jobs have already consumed. The fair-share factor does not involve a fixed allotment, whereby a user's access to a machine is cut off once that allotment is reached.

Instead, the fair-share factor serves to prioritize queued jobs such that those jobs charging accounts that are under-serviced are scheduled first, while jobs charging accounts that are over-serviced are scheduled when the machine would otherwise go idle.

Slurm's fair-share factor is a floating point number between 0.0 and 1.0 that reflects the shares of a computing resource that a user has been allocated and the amount of computing resources the user's jobs have consumed. The higher the value, the higher is the placement in the queue of jobs waiting to be scheduled.

By default, the computing resource is the computing cycles delivered by a machine in the units of allocated_cpus*seconds. Other resources can be taken into account by configuring a partition's TRESBillingWeights option. The TRESBillingWeights option allows you to account for consumed resources other than just CPUs by assigning different billing weights to different Trackable Resources (TRES) such as CPUs, nodes, memory, licenses and generic resources (GRES). For example, when billing only for CPUs, if a job requests 1CPU

and 64GB of memory on a 16CPU, 64GB node the job will only be billed for 1CPU when it really used the whole node.

By default, when TRESBillingWeights is configured, a job is billed for each individual TRES used. The billable TRES is calculated as the sum of all TRES types multiplied by their corresponding billing weight.

For example, the following jobs on a partition configured with TRESBillingWeights=CPU=1.0,Mem=0.25G and 16CPU, 64GB nodes would be billed as:

CPUs	Mem GB
Job1: (1 *1.0) + (60*0.25) = (1 + 15) = 16	
Job2: (16*1.0) + (1 *0.25) = (16+.25) = 16.25	
Job3: (16*1.0) + (60*0.25) = (16+ 15) = 31	

Another method of calculating the billable TRES is by taking the MAX of the individual TRES' on a node (e.g. cpus, mem, gres) plus the SUM of the global TRES' (e.g. licenses). For example the above job's billable TRES would be calculated as:

CPUs	Mem GB
Job1: MAX((1 *1.0), (60*0.25)) = 15	
Job2: MAX((15*1.0), (1 *0.25)) = 15	
Job3: MAX((16*1.0), (64*0.25)) = 16	

This method is turned on by defining the MAX_TRES priority flags in the slurm.conf.

The `sprio` Utility

The `sprio` command provides a summary of the six factors that comprise each job's scheduling priority. While `squeue` has format options (%p and %Q) that display a job's composite priority, `sprio` can be used to display a breakdown of the priority components for each job. In addition, the `sprio -w` option displays the weights (PriorityWeightAge, PriorityWeightFairshare, etc.) for each factor as it is currently configured.

Configuration of Priority

The following slurm.conf (SLURM_CONFIG_FILE) parameters are used to configure the Multi-factor Job Priority Plugin. See slurm.conf(5) man page for more details.

- **PriorityType** Set this value to "priority/multifactor" to enable the Multi-factor Job Priority Plugin. The default value for this variable is "priority/basic" which enables simple FIFO scheduling.
- **PriorityDecayHalfLife** This determines the contribution of historical usage on the composite usage value. The larger the number, the longer past usage affects fair-share. If set to 0 no decay will be applied. This is helpful if you want to enforce hard time limits per association. If set to 0 PriorityUsageResetPeriod must be set to some interval. The unit is a time string (i.e. min, hr:min:00, days-hr:min:00, or days-hr). The default value is 7-0 (7 days).

- **PriorityCalcPeriod** The period of time in minutes in which the half-life decay will be re-calculated. The default value is 5 (minutes).
- **PriorityUsageResetPeriod** At this interval the usage of associations will be reset to 0. This is used if you want to enforce hard limits of time usage per association. If PriorityDecayHalfLife is set to be 0 no decay will happen and this is the only way to reset the usage accumulated by running jobs. By default this is turned off and it is advised to use the PriorityDecayHalfLife option to avoid not having anything running on your cluster, but if your schema is set up to only allow certain amounts of time on your system this is the way to do it. Applicable only if PriorityType=priority/multifactor. The unit is a time string (i.e. NONE, NOW, DAILY, WEEKLY). The default is NONE.
 - **NONE:** Never clear historic usage. The default value.
 - **NOW:** Clear the historic usage now. Executed at startup and reconfiguration time.
 - **DAILY:** Cleared every day at midnight.
 - **WEEKLY:** Cleared every week on Sunday at time 00:00.
 - **MONTHLY:** Cleared on the first day of each month at time 00:00.
 - **QUARTERLY:** Cleared on the first day of each quarter at time 00:00.
 - **YEARLY:** Cleared on the first day of each year at time 00:00.
- **PriorityFavorSmall** A boolean that sets the polarity of the job size factor. The default setting is NO which results in larger node sizes having a larger job size factor. Setting this parameter to YES means that the smaller the job, the greater the job size factor will be.
- **PriorityMaxAge** Specifies the queue wait time at which the age factor maxes out. The unit is a time string (i.e. min, hr:min:00, days-hr:min:00, or days-hr). The default value is 7-0 (7 days).
- **PriorityWeightAge** An unsigned integer that scales the contribution of the age factor.
- **PriorityWeightFairshare** An unsigned integer that scales the contribution of the fair-share factor.
- **PriorityWeightJobSize** An unsigned integer that scales the contribution of the job size factor.
- **PriorityWeightPartition** An unsigned integer that scales the contribution of the partition factor.
- **PriorityWeightQOS** An unsigned integer that scales the contribution of the quality of service factor.
- **PriorityWeightTRES** A list of TRES Types and weights that scales the contribution of each TRES Type's factor.

NOTE: As stated above, the six priority factors range from 0.0 to 1.0. As such, the PriorityWeight terms may need to be set to a high enough value (say, 1000) to resolve very tiny differences in priority factors. This is especially true with the fair-share factor, where two jobs may differ in priority by as little as .001. (or even less!)

Job Priority Exercises

In this set of labs, you will learn how to assign priority to a job.

Exercise 1: Assign priority by QOS

In normal cluster usage, priority is granted at the QOS level. In essence, if the cluster admin wants to provide a "Higher level of service" to a principal investor, he can do so by prioritizing the cluster workload around a Quality of Service (QOS)

The default setup for this training environment looks like this:

Cluster	Account	User	QOS	Def QOS
cluster	root		normal	
cluster	root	root	normal	
cluster	bedrock		normal	
cluster	bedrock	arnold	normal	
cluster	bedrock	bambam	normal	
cluster	bedrock	barney	normal	
cluster	bedrock	betty	normal	
cluster	bedrock	chip	normal	
cluster	bedrock	dino	normal	
cluster	bedrock	edna	normal	
cluster	bedrock	fred	normal	
cluster	bedrock	gazoo	normal	
cluster	bedrock	pebbles	normal	
cluster	bedrock	root	normal	
cluster	bedrock	wilma	normal	

Configure the slurm.conf

1. As root, edit /etc/slurm/slurm.conf and add the following to the end of the file:

```
PriorityWeightQOS=1000000
```

And change this:

```
#PriorityType=priority/multifactor
```

To this (i.e. remove the '#' hash symbol from the start of the line):

```
PriorityType=priority/multifactor
```

2. Save and exit the file.
3. Read in the changes to the controller:
`/lab_scripts/restart.sh`

4. Confirm the QOS weight by using the sprio command:

```
sprio -w
```

Should see something like this:

JOBID	PARTITION	PRIORITY	SITE	QOS
Weights			1	1000000

Add accounts and assign priority to them

In this part of the exercise, you will create the following QOSs, assign a priority to them, and place a limit on the number of jobs that can be sent to them as follows:

QOS	Priority	Max jobs submitted or pending	Users that can use the QOS
high	10000	4	Fred, Barney
medium	5000	20	Fred, Barney, Wilma, Betty
low	100	unlimited	Everybody

1. As root user, create the QOSs:

```
sacctmgr -i add qos high
sacctmgr -i add qos medium
sacctmgr -i add qos low
```

2. Show the qos's:

```
sacctmgr show qos format=Name,Priority
```

You will see:

Name	Priority
normal	0
high	0
medium	0
low	0

3. Set the priorities on the QOSs:

```
sacctmgr -i modify qos high set priority=10000
sacctmgr -i modify qos medium set priority=5000
sacctmgr -i modify qos low set priority=100
```

4. Assign all users, cluster and accounts to the low QOS, and make the low QOS the default for all users, cluster and accounts:

```
sacctmgr -i modify account bedrock set qos=low
sacctmgr -i modify account bedrock set defaultqos=low
```

5. View the created QOSs and their priority values using sacctmgr:

```
sacctmgr show qos format=name,priority
```

Should show:

Name	Priority
normal	0
high	10000
medium	5000
low	100

Assign users to be able to use the QOSs:

```
sacctmgr -i modify user fred,barney set qos=high,medium,low  
sacctmgr -i modify user wilma,betty set qos=medium,low
```

6. Confirm the assignments with the following command:

```
sacctmgr list assoc format=Cluster,Account,User,QOS,defaultqos
```

It should show:

Cluster	Account	User	QOS	Def QOS
cluster	root		normal	
cluster	root	root	normal	
cluster	bedrock		low	low
cluster	bedrock	arnold	low	low
cluster	bedrock	bambam	low	low
cluster	bedrock	barney	high,low,medium	low
cluster	bedrock	betty	low,medium	low
cluster	bedrock	chip	low	low
cluster	bedrock	dino	low	low
cluster	bedrock	edna	low	low
cluster	bedrock	fred	high,low,medium	low
cluster	bedrock	gazoo	low	low
cluster	bedrock	pebbles	low	low
cluster	bedrock	root	low	low

7. It is also possible to place limits on the QOS other than priority. Set the maximum number of jobs pending or running state at any time on the QOS by using the following syntax:

```
sacctmgr -i modify qos high set MaxSubmitJobs=4  
sacctmgr -i modify qos medium set MaxSubmitJobs=20  
sacctmgr -i modify qos low set MaxSubmitJobs=-1
```

8. Show the changes:

```
sacctmgr show qos format=name,priority,MaxSubmit
```

Should show:

Name	Priority	MaxSubmit
normal	0	
high	10000	4
medium	5000	20
low	100	

Submit workload to see the priority

Now it's time to submit workload

1. As **fred**, submit 10, 5-minute jobs to the low qos (enter exactly as shown):

```
for x in {1..10} ; do
  sbatch --mem=1000 -N1 -t5 --qos=low --exclusive --wrap='sleep 300'
done
```

2. As **Barney**, try and submit 5 jobs to the high qos (also enter exactly as shown):

```
for x in {1..5} ; do
  sbatch --mem=1000 -N1 -t5 --qos=high --exclusive --wrap='sleep 300'
done
```

The error will be:

```
sbatch: error: QOSMaxSubmitJobPerUserLimit
sbatch: error: Batch job submission failed: Job violates accounting/QOS
policy (job submit limit, user's size and/or time limits)
```

3. As **bambam**, submit the following (also enter exactly as shown):

```
for x in {1..5} ; do
  sbatch -N1 --mem=1000 -t5 --exclusive --wrap='sleep 300'
done
```

Now there should be many jobs in queue, and they will have different priorities.

4. Verify by running squeue like this:

```
squeue ; sprio -o "%15i %9r %.8u %.10Y %n"
```

You should see:

JOBID	PARTITION	NAME	USER	ST	TIME	NODES	NODELIST (REASON)
11	debug	wrap	barney	PD	0:00	1	(Resources)
12	debug	wrap	barney	PD	0:00	1	(Priority)
13	debug	wrap	barney	PD	0:00	1	(Priority)
14	debug	wrap	barney	PD	0:00	1	(Priority)
15	debug	wrap	bambam	PD	0:00	1	(Priority)
16	debug	wrap	bambam	PD	0:00	1	(Priority)
17	debug	wrap	bambam	PD	0:00	1	(Priority)
18	debug	wrap	bambam	PD	0:00	1	(Priority)
19	debug	wrap	bambam	PD	0:00	1	(Priority)
1	debug	wrap	fred	R	1:35	1	node00
2	debug	wrap	fred	R	1:35	1	node01
3	debug	wrap	fred	R	1:35	1	node02
4	debug	wrap	fred	R	1:35	1	node03
5	debug	wrap	fred	R	1:35	1	node04
6	debug	wrap	fred	R	1:35	1	node05
7	debug	wrap	fred	R	1:35	1	node06
8	debug	wrap	fred	R	1:35	1	node07
9	debug	wrap	fred	R	1:35	1	node08
10	debug	wrap	fred	R	1:35	1	node09

JOBID	PARTITION	USER	PRIORITY	QOSNAME
11	debug	barney	1000000	high
12	debug	barney	1000000	high
13	debug	barney	1000000	high
14	debug	barney	1000000	high
15	debug	bambam	10000	low
16	debug	bambam	10000	low
17	debug	bambam	10000	low
18	debug	bambam	10000	low
19	debug	bambam	10000	low

NOTE: The jobs that are pending (PD) will have a priority listed in the `sprio` command output. And, since they were submitted to a different QOS, they have different priorities. So in the `squeue` output, the barney jobs are higher in the prioritized list, because Barney's job have a higher priority (1000000) because of the QOS submitted to (high)

Cleanup

1. Reset the Docker Slurm cluster environment back to its original, pristine state by exiting the login Docker container back to the Ubuntu AWS prompt. And **as ubuntu**:

```
cd ~/docker-scale-out  
make clean && make
```

Fairshare Discussion

Slurm fairshare is one of the factors that make up a job's priority.

Computing the fair-share factor requires the installation and operation of the Slurm Accounting Database to provide the assigned shares and the consumed, computing resources described below.

The fair-share component to a job's priority influences the order in which a user's queued jobs are scheduled to run based on the portion of the computing resources they have been allocated and the resources their jobs have already consumed. The fair-share factor does not involve a fixed allotment, whereby a user's access to a machine is cut off once that allotment is reached.

Instead, the fair-share factor serves to prioritize queued jobs such that those jobs charging accounts that are under-serviced are scheduled first, while jobs charging accounts that are over-serviced are scheduled when the machine would otherwise go idle.

Slurm's fair-share factor is a floating point number between 0.0 and 1.0 that reflects the shares of a computing resource that a user has been allocated and the amount of computing resources the user's jobs have consumed. The higher the value, the higher is the placement in the queue of jobs waiting to be scheduled.

By default, the computing resource is the computing cycles delivered by a machine in the units of allocated_cpus*seconds. Other resources can be taken into account by configuring a partition's TRESBillingWeights option. The TRESBillingWeights option allows you to account for consumed resources other than just CPUs by assigning different billing weights to different Trackable Resources (TRES) such as CPUs, nodes, memory, licenses and generic resources (GRES). For example, when billing only for CPUs, if a job requests 1CPU and 64GB of memory on a 16CPU, 64GB node the job will only be billed for 1CPU when it really used the whole node.

By default, when TRESBillingWeights is configured, a job is billed for each individual TRES used. The billable TRES is calculated as the sum of all TRES types multiplied by their corresponding billing weight.

For example, the following jobs on a partition configured with TRESBillingWeights=CPU=1.0,Mem=0.25G and 16CPU, 64GB nodes would be billed as:

CPUs	Mem	GB
Job1:	(1 *1.0)	+ (60*0.25) = (1 + 15) = 16
Job2:	(16*1.0)	+ (1 *0.25) = (16+.25) = 16.25
Job3:	(16*1.0)	+ (60*0.25) = (16+ 15) = 31

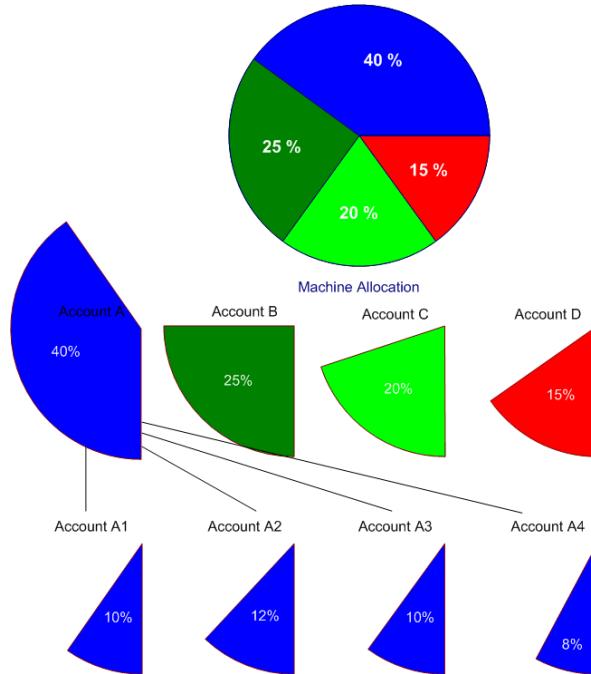
Another method of calculating the billable TRES is by taking the MAX of the individual TRES' on a node (e.g. cpus, mem, gres) plus the SUM of the global TRES' (e.g. licenses). For example the above job's billable TRES would be calculated as:

CPUs	Mem	GB
Job1: MAX((1 *1.0), (60*0.25)) = 15		
Job2: MAX((15*1.0), (1 *0.25)) = 15		
Job3: MAX((16*1.0), (64*0.25)) = 16		

This method is turned on by defining the MAX_TRES priority flags in the slurm.conf.

NORMALIZED SHARES

The fair-share hierarchy represents the portions of the computing resource that have been allocated to multiple projects. These allocations are assigned to an account. There can be multiple levels of allocations made as allocations of a given account are further divided to sub-accounts:



The chart above shows the resources of the machine allocated to four accounts, A, B, C and D. Furthermore, account A's shares are further allocated to sub accounts, A1 through A4. Users are granted permission (through `sacctmgr`) to submit jobs against specific accounts. If there are 10 users given equal shares in Account A3, they will each be allocated 1% of the machine.

A user's normalized shares is simply

$$S = (S_{\text{user}} / S_{\text{siblings}}) * (S_{\text{account}} / S_{\text{sibling-accounts}}) * (S_{\text{parent}} / S_{\text{parent-siblings}}) * \dots$$

Where:

S is the user's normalized share, between zero and one

S_{user} are the number of shares of the account allocated to the user

S_{siblings} are the total number of shares allocated to all users permitted to charge the account (including S_{user})

S_{account} are the number of shares of the parent account allocated to the account

$S_{\text{sibling-accounts}}$ are the total number of shares allocated to all sub-accounts of the parent account

S_{parent} are the number of shares of the grandparent account allocated to the parent

$S_{parent-siblings}$ are the total number of shares allocated to all sub-accounts of the grandparent account

NORMALIZED USAGE

The processor*seconds allocated to every job are tracked in real-time. If one only considered usage over a fixed time period, then calculating a user's normalized usage would be a simple quotient:

$$U_N = U_{user} / U_{total}$$

Where:

U_N is normalized usage, between zero and one

U_{user} is the processor*seconds consumed by all of a user's jobs in a given account for over a fixed time period

U_{total} is the total number of processor*seconds utilized across the cluster during that same time period

However, significant real-world usage quantities span multiple time periods. Rather than treating usage over a number of weeks or months with equal importance, Slurm's fair-share priority calculation places more importance on the most recent resource usage and less importance on usage from the distant past.

The Slurm usage metric is based off a half-life formula that favors the most recent usage statistics. Usage statistics from the past decrease in importance based on a single decay factor, D:

$$U_H = U_{current_period} + (D * U_{last_period}) + (D * D * U_{period-2}) + \dots$$

Where:

U_H is the historical usage subject to the half-life decay

$U_{current_period}$ is the usage charged over the current measurement period

U_{last_period} is the usage charged over the last measurement period

$U_{period-2}$ is the usage charged over the second last measurement period

D is a decay factor between zero and one that delivers the half-life decay based off the PriorityDecayHalfLife setting in the slurm.conf file. Without accruing additional usage, a user's UH usage will decay to half its original value after a time period of PriorityDecayHalfLife seconds.

In practice, the PriorityDecayHalfLife could be a matter of seconds or days as appropriate for each site. The decay is recalculated every PriorityCalcPeriod minutes, or 5 minutes by default. The decay factor, D, is assigned the value that will achieve the half-life decay rate specified by the PriorityDecayHalfLife parameter.

The total number of processor*seconds utilized can be similarly aggregated with the same decay factor:

$$R_H = R_{\text{current_period}} + (D * R_{\text{last_period}}) + (D * D * R_{\text{period-2}}) + \dots$$

Where:

R_H is the total historical usage subject to the same half-life decay as the usage formula.

$R_{\text{current_period}}$ is the total usage charged over the current measurement period

$R_{\text{last_period}}$ is the total usage charged over the last measurement period

$R_{\text{period-2}}$ is the total usage charged over the second last measurement period

D is the decay factor between zero and one

A user's normalized usage that spans multiple time periods then becomes:

$$U = U_H / R_H$$

SIMPLIFIED FAIR-SHARE FORMULA

The simplified formula for calculating the fair-share factor for usage that spans multiple time periods and subject to a half-life decay is:

$$F = 2^{**}(-U/S/d)$$

Where:

F is the fair-share factor

S is the normalized shares

U is the normalized usage factoring in half-life decay

d is the FairShareDampeningFactor (a configuration parameter, default value of 1)

The fair-share factor will therefore range from zero to one, where one represents the highest priority for a job. A fair-share factor of 0.5 indicates that the user's jobs have used exactly the portion of the machine that they have been allocated. A fair-share factor of above 0.5 indicates that the user's jobs have consumed less than their allocated share while a fair-share factor below 0.5 indicates that the user's jobs have consumed more than their allocated share of the computing resources.

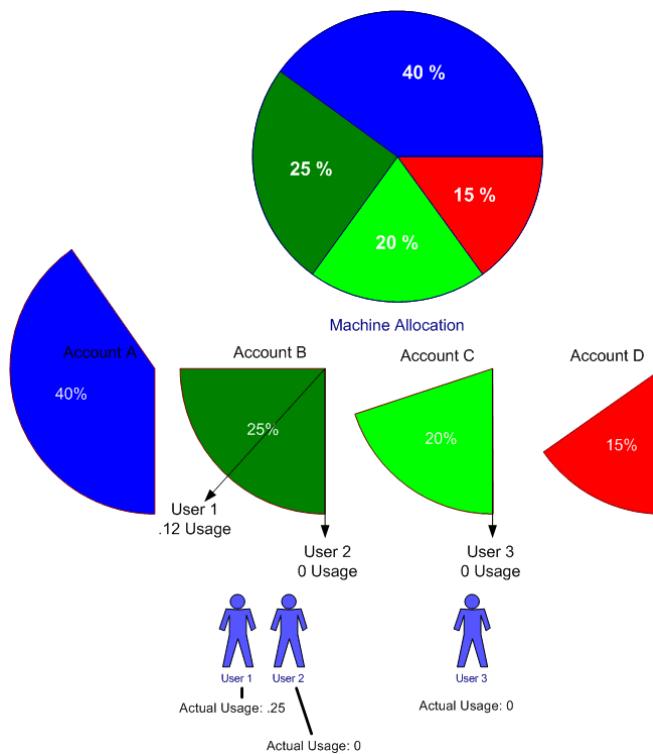
THE FAIR-SHARE FACTOR UNDER AN ACCOUNT HIERARCHY

The method described above presents a system whereby the priority of a user's job is calculated based on the portion of the machine allocated to the user and the historical usage of all the jobs run by that user under a specific account.

Another layer of "fairness" is necessary however, one that factors in the usage of other users drawing from the same account. This allows a job's fair-share factor to be influenced by the computing resources delivered to jobs of other users drawing from the same account.

If there are two members of a given account, and if one of those users has run many jobs under that account, the job priority of a job submitted by the user who has not run any jobs will be negatively affected. This ensures that the combined usage charged to an account matches the portion of the machine that is allocated to that account.

In the example below, when user 3 submits their first job using account C, they will want their job's priority to reflect all the resources delivered to account B. They do not care that user 1 has been using up a significant portion of the cycles allocated to account B and user 2 has yet to run a job out of account B. If user 2 submits a job using account B and user 3 submits a job using account C, user 3 expects their job to be scheduled before the job from user 2.



THE SLURM FAIR-SHARE FORMULA

The Slurm fair-share formula has been designed to provide fair scheduling to users based on the allocation and usage of every account.

The actual formula used is a refinement of the formula presented above:

$$F = 2^{**} (-U_E / S)$$

The difference is that the usage term is effective usage, which is defined as:

$$U_E = U_{Achild} + ((U_{Eparent} - U_{Achild}) * S_{child}/S_{all_siblings})$$

Where:

U_E is the effective usage of the child user or child account

U_{Achild} is the actual usage of the child user or child account

$U_{Eparent}$ is the effective usage of the parent account

S_{child} is the shares allocated to the child user or child account

$S_{all_siblings}$ is the shares allocated to all the children of the parent account

This formula only applies with the second tier of accounts below root. For the tier of accounts just under root, their effective usage equals their actual usage.

Because the formula for effective usage includes a term of the effective usage of the parent, the calculation for each account in the tree must start at the second tier of accounts and proceed downward: to the children accounts, then grandchildren, etc. The effective usage of the users will be the last to be calculated.

Plugging in the effective usage into the fair-share formula above yields a fair-share factor that reflects the aggregated usage charged to each of the accounts in the fair-share hierarchy.

FAIRSHARE=PARENT

It is possible to disable the fairshare at certain levels of the fair share hierarchy by using the FairShare=parent option of `sacctmgr`. For users and accounts with FairShare=parent the normalized shares and effective usage values from the parent in the hierarchy will be used when calculating fairshare priorities.

If all users in an account is configured with FairShare=parent the result is that all the jobs drawing from that account will get the same fairshare priority, based on the accounts total usage. No additional fairness is added based on users individual usage.

EXAMPLE

The following example demonstrates the effective usage calculations and resultant fair-share factors. (See Figure 3 below.)

The machine's computing resources are allocated to accounts A and D with 40 and 60 shares respectively. Account A is further divided into two children accounts, B with 30 shares and C with 10 shares. Account D is further divided into two children accounts, E with 25 shares and F with 35 shares.

Note: the shares at any given tier in the Account hierarchy do not need to total up to 100 shares. This example shows them totaling up to 100 to make the arithmetic easier to follow in your head.

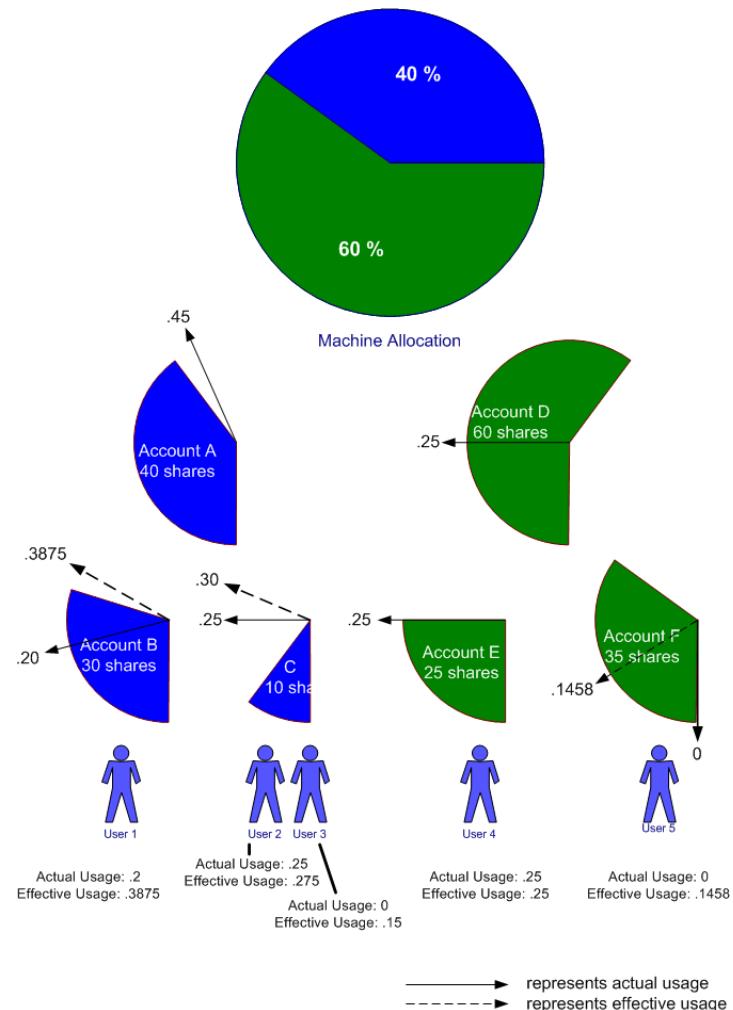
User 1 is granted permission to submit jobs against the B account. Users 2 and 3 are granted one share each in the C account. User 4 is the sole member of the E account and User 5 is the sole member of the F account.

Note: accounts A and D do not have any user members in this example, though users could have been assigned.

The shares assigned to each account make it easy to determine normalized shares of the machine's complete resources. Account A has .4 normalized shares, B has .3 normalized shares, etc. Users who are sole members of an account have the same number of normalized shares as the account. (E.g., User 1 has .3 normalized shares). Users who share accounts have a portion of the normalized shares based on their shares. For example, if user 2 had been allocated 4 shares instead of 1, user 2 would have had .08 normalized shares. With users 2 and 3 each holding 1 share, they each have a normalized share of 0.05.

Users 1, 2, and 4 have run jobs that have consumed the machine's computing resources. User 1's actual usage is 0.2 of the machine; user 2 is 0.25, and user 4 is 0.25.

The actual usage charged to each account is represented by the solid arrows. The actual usage charged to each account is summed as one goes up the tree. Account C's usage is the sum of the usage of Users 2 and 3; account A's actual usage is the sum of its children, accounts B and C.



- User 1 normalized share: 0.3
- User 2 normalized share: 0.05
- User 3 normalized share: 0.05
- User 4 normalized share: 0.25
- User 5 normalized share: 0.35

As stated above, the effective usage is computed from the formula:

$$U_E = U_{Achild} + ((U_{Eparent} - U_{Achild}) * S_{child}/S_{all_siblings})$$

The effective usage for all accounts at the first tier under the root allocation is always equal to the actual usage:

- Account A's effective usage is therefore equal to .45. Account D's effective usage is equal to .25.
- Account B effective usage: $0.2 + ((0.45 - 0.2) * 30 / 40) = 0.3875$
- Account C effective usage: $0.25 + ((0.45 - 0.25) * 10 / 40) = 0.3$
- Account E effective usage: $0.25 + ((0.25 - 0.25) * 25 / 60) = 0.25$
- Account F effective usage: $0.0 + ((0.25 - 0.0) * 35 / 60) = 0.1458$

The effective usage of each user is calculated using the same formula:

- User 1 effective usage: $0.2 + ((0.3875 - 0.2) * 1 / 1) = 0.3875$
- User 2 effective usage: $0.25 + ((0.3 - 0.25) * 1 / 2) = 0.275$
- User 3 effective usage: $0.0 + ((0.3 - 0.0) * 1 / 2) = 0.15$
- User 4 effective usage: $0.25 + ((0.25 - 0.25) * 1 / 1) = 0.25$
- User 5 effective usage: $0.0 + ((0.1458 - 0.0) * 1 / 1) = 0.1458$

Using the Slurm fair-share formula,

$$F = 2^{**}(-U_E/S)$$

the fair-share factor for each user is:

- User 1 fair-share factor: $2^{**}(-.3875 / .3) = 0.408479$
- User 2 fair-share factor: $2^{**}(-.275 / .05) = 0.022097$
- User 3 fair-share factor: $2^{**}(-.15 / .05) = 0.125000$
- User 4 fair-share factor: $2^{**}(-.25 / .25) = 0.500000$
- User 5 fair-share factor: $2^{**}(-.1458 / .35) = 0.749154$

From this example, one can see that users 1,2, and 3 are over-serviced while user 5 is under-serviced. Even though user 3 has yet to submit a job, his/her fair-share factor is negatively influenced by the jobs users 1 and 2 have run.

Based on the fair-share factor alone, if all 5 users were to submit a job charging their respective accounts, user 5's job would be granted the highest scheduling priority.

The `sprio` utility

The `sprio` command provides a summary of the six factors that comprise each job's scheduling priority. While `squeue` has format options (`%p` and `%Q`) that display a job's composite priority, `sprio` can be used to display a breakdown of the priority components for each job. In addition, the `sprio -w` option displays the weights (PriorityWeightAge, PriorityWeightFairshare, etc.) for each factor as it is currently configured.

Configuration

The following slurm.conf (SLURM_CONFIG_FILE) parameters are used to configure the Multi-factor Job Priority Plugin. See slurm.conf(5) man page for more details.

PriorityType

Set this value to "priority/multifactor" to enable the Multi-factor Job Priority Plugin. The default value for this variable is "priority/basic" which enables simple FIFO scheduling.

PriorityDecayHalfLife

This determines the contribution of historical usage on the composite usage value. The larger the number, the longer past usage affects fair-share. If set to 0 no decay will be applied. This is helpful if you want to enforce hard time limits per association. If set to 0 PriorityUsageResetPeriod must be set to some interval. The unit is a time string (i.e. min, hr:min:00, days-hr:min:00, or days-hr). The default value is 7-0 (7 days).

PriorityCalcPeriod

The period of time in minutes in which the half-life decay will be re-calculated. The default value is 5 (minutes).

PriorityUsageResetPeriod

At this interval the usage of associations will be reset to 0. This is used if you want to enforce hard limits of time usage per association. If PriorityDecayHalfLife is set to be 0 no decay will happen and this is the only way to reset the usage accumulated by running jobs. By default this is turned off and it is advised to use the PriorityDecayHalfLife option to avoid not having anything running on your cluster, but if your schema is set up to only allow certain amounts of time on your system this is the way to do it. Applicable only if PriorityType=priority/multifactor. The unit is a time string (i.e. NONE, NOW, DAILY, WEEKLY). The default is NONE.

- **NONE:** Never clear historic usage. The default value.
- **NOW:** Clear the historic usage now. Executed at startup and reconfiguration time.
- **DAILY:** Cleared every day at midnight.
- **WEEKLY:** Cleared every week on Sunday at time 00:00.
- **MONTHLY:** Cleared on the first day of each month at time 00:00.
- **QUARTERLY:** Cleared on the first day of each quarter at time 00:00.
- **YEARLY:** Cleared on the first day of each year at time 00:00.

PriorityFavorSmall

A boolean that sets the polarity of the job size factor. The default setting is NO which results in larger node sizes having a larger job size factor. Setting this parameter to YES means that the smaller the job, the greater the job size factor will be.

PriorityMaxAge

Specifies the queue wait time at which the age factor maxes out. The unit is a time string (i.e. min, hr:min:00, days-hr:min:00, or days-hr). The default value is 7-0 (7 days).

PriorityWeightAge

An unsigned integer that scales the contribution of the age factor.

PriorityWeightFairshare

An unsigned integer that scales the contribution of the fair-share factor.

PriorityWeightJobSize

An unsigned integer that scales the contribution of the job size factor.

PriorityWeightPartition

An unsigned integer that scales the contribution of the partition factor.

PriorityWeightQOS

An unsigned integer that scales the contribution of the quality of service factor.

PriorityWeightTRES

A list of TRES Types and weights that scales the contribution of each TRES Type's factor.

Note: As stated above, the six priority factors range from 0.0 to 1.0. As such, the PriorityWeight terms may need to be set to a high enough value (say, 1000) to resolve very tiny differences in priority factors. This is especially true with the fair-share factor, where two jobs may differ in priority by as little as .001. (or even less!)

Configuration Example

The following are sample slurm.conf file settings for the Multi-factor Job Priority Plugin.

The first example is for running the plugin applying decay over time to reduce usage. Hard limits can be used in this configuration, but will have less effect since usage will decay over time instead of having no decay over time.

```
# Activate the Multi-factor Job Priority Plugin with decay
PriorityType=priority/multifactor

# 2 week half-life
PriorityDecayHalfLife=14-0

# The larger the job, the greater its job size priority.
PriorityFavorSmall=NO

# The job's age factor reaches 1.0 after waiting in the
# queue for 2 weeks.
PriorityMaxAge=14-0

# This next group determines the weighting of each of the
# components of the Multi-factor Job Priority Plugin.
# The default value for each of the following is 1.
PriorityWeightAge=1000
PriorityWeightFairshare=10000
PriorityWeightJobSize=1000
PriorityWeightPartition=1000
PriorityWeightQOS=0 # don't use the qos factor
```

This example is for running the plugin with no decay on usage, thus making a reset of usage necessary.

```
# Activate the Multi-factor Job Priority Plugin with decay
PriorityType=priority/multifactor

# apply no decay
PriorityDecayHalfLife=0
```

```
# reset usage after 1 month
PriorityUsageResetPeriod=MONTHLY

# The larger the job, the greater its job size priority.
PriorityFavorSmall=NO

# The job's age factor reaches 1.0 after waiting in the
# queue for 2 weeks.
PriorityMaxAge=14-0

# This next group determines the weighting of each of the
# components of the Multi-factor Job Priority Plugin.
# The default value for each of the following is 1.
PriorityWeightAge=1000
PriorityWeightFairshare=10000
PriorityWeightJobSize=1000
PriorityWeightPartition=1000
PriorityWeightQOS=0 # don't use the qos factor
```

Fairshare Exercises

In this set of labs, you will learn how to manage fairshare in Slurm.

Exercise 1: Create a fairshare usage paradigm based on accounts

This exercise shows how to create several accounts, assign users to the accounts, then assign shares to them. For the purposes of this lab, the accounts, shares, and respective users assigned to them are:

Account	Shares	Members
Bioinformatics	50	Fred, Wilma
Math	30	Barney, Betty
Stats	15	Pebbles, Bambam
Music	5	Dino, Gazoo

The shares represent how much of the cluster each entity has been granted. When users submit workload, they are charged back for their runs against the Account they belong to.

Configure slurm.conf for fairshare

1. As root, edit the /etc/slurm/slurm.conf, and add, or confirm, the following entries:

```
...
PriorityFlags=FAIR_TREE
PriorityType=priority/multifactor
PriorityWeightAge=1000
PriorityWeightFairshare=100000
PriorityWeightPartition=10000
PriorityWeightQOS=1000000
PriorityCalcPeriod=1
...
```

2. Restart the scheduler:

```
/lab_scripts/restart.sh
```

3. Verify the new plugin is loaded, by entering:

```
scontrol show config | grep -i priority
```

Should return:

```
PriorityParameters      = (null)
PrioritySiteFactorParameters = (null)
PrioritySiteFactorPlugin = (null)
PriorityDecayHalfLife    = 7-00:00:00
PriorityCalcPeriod       = 00:01:00
PriorityFavorSmall       = No
PriorityFlags            =
PriorityMaxAge          = 00:01:00
PriorityUsageResetPeriod = NONE
PriorityType              = priority/multifactor
PriorityWeightAge         = 1000
PriorityWeightAssoc        = 0
PriorityWeightFairShare   = 100000
PriorityWeightJobSize     = 0
PriorityWeightPartition   = 10000
PriorityWeightQOS          = 1000000
PriorityWeightTRES         = (null)
```

Create the accounts with fairshare values

1. Delete root user and bedrock accounts from having any shares:
`sacctmgr -i modify user name=root set fairshare=0
sacctmgr -i modify account name=bedrock set fairshare=0`
2. As the root user, create bioinformatics, math, stats, and music accounts, and assign a fairshare shares to them:
`sacctmgr -i create account name=bioinformatics fairshare=50
sacctmgr -i create account name=math fairshare=30
sacctmgr -i create account name=stats fairshare=15
sacctmgr -i create account name=music fairshare=5`
3. Add users to the newly-created accounts:
`sacctmgr -i create user fred,wilma account=bioinformatics
sacctmgr -i create user barney,betty account=math
sacctmgr -i create user pebbles,bambam account=stats
sacctmgr -i create user dino,gazoo account=music`
4. Confirm the creation of the accounts, the user assignments, and share values (You can adjust the output format of each column by placing a %<VALUE> just after the column name. If you want to left-justify the results in the column, place a “-” in front, like %-<VALUE>):
`sacctmgr list assoc account=bioinformatics,stats,math,music \
format=Account%15,User,Cluster,fairshare`

You should see:

Account	User	Cluster	Share
<hr/>			
bioinformatics		cluster	50
bioinformatics	fred	cluster	1
bioinformatics	wilma	cluster	1
math		cluster	30
math	barney	cluster	1
math	betty	cluster	1
music		cluster	5
music	dino	cluster	1
music	gazoo	cluster	1
stats		cluster	15
stats	bambam	cluster	1
stats	pebbles	cluster	1

- Another way to show the relationships is through the associations. You can do this by running the scontrol command in a loop (enter exactly as shown):

```
for x in bioinformatics math music stats ; do  
  scontrol show assoc_mgr flags=assoc | grep $x  
done
```

You should see (UIDs and PartitionIDs may differ from yours, of course):

```
ClusterName=cluster Account=bioinformatics UserName= Partition= ID=15  
ClusterName=cluster Account=bioinformatics UserName=fred(2017) Partition= ID=19  
ClusterName=cluster Account=bioinformatics UserName=wilma(2020) Partition= ID=20  
ClusterName=cluster Account=math UserName= Partition= ID=16  
ClusterName=cluster Account=math UserName=barney(2012) Partition= ID=21  
ClusterName=cluster Account=math UserName=betty(2013) Partition= ID=22  
ClusterName=cluster Account=music UserName= Partition= ID=18  
ClusterName=cluster Account=music UserName=dino(2015) Partition= ID=25  
ClusterName=cluster Account=music UserName=gazoo(2018) Partition= ID=26  
ClusterName=cluster Account=stats UserName= Partition= ID=17  
ClusterName=cluster Account=stats UserName=bambam(2011) Partition= ID=24  
ClusterName=cluster Account=stats UserName=pebbles(2019) Partition= ID=23
```

Submit some jobs and view the change in job priorities

- As fred, in his home directory, create the following submit script called testping.sh and add the following to it (This script is also available here: /lab_scripts/testping.sh.fred):

```

#!/bin/bash
#SBATCH -N2 -t 20
#SBATCH --mem=1000
#SBATCH -o testping-out%j.txt
#SBATCH -e testping-err%j.txt
#SBATCH --exclusive
#SBATCH -A bioinformatics

ping -c 20 -4 google.com

```

2. If you've copied it, rename it and flag it executable (these commands should work either way, whether you typed it out or copied it):

```

mv testping.sh{.fred,} 2>/dev/null
chmod +x testping.sh

```

3. As pebbles, in her home directory, create the following submit script called `testping.sh` and add the following to it (just copy, paste, and use the previous script and change the string in purple-- or just copy, paste, rename, and use the script available here: `/lab_scripts/testping.sh.pebbles`):

```

#!/bin/bash
#SBATCH -N2 -t 20
#SBATCH --mem=1000
#SBATCH -o testping-out%j.txt
#SBATCH -e testping-err%j.txt
#SBATCH --exclusive
#SBATCH -A stats

ping -c 20 -4 google.com

```

4. If you've copied it, rename it and flag it executable (these commands should work either way, whether you typed it out or copied it):

```

mv testping.sh{.pebbles,} 2>/dev/null
chmod +x testping.sh

```

5. Before submitting any workload, let's evaluate the output from the `sshare` command. Run the `sshare` command as root:

```

sshare --accounts=bioinformatics,math,music,stats

```

You should see:

Account	User	RawShares	NormShares	RawUsage	EffectvUsage	FairShare
bioinformatics		50	0.500000	0	0.000000	
math		30	0.300000	0	0.000000	
music		5	0.050000	0	0.000000	

The accounts have the following shares calculations:

sum of Raw Shares (parent level): $50+30+15+5 = 100$ (doesn't HAVE to total 100, but it's easier for my math)

Raw Shares of bioinformatics: 50

Norm Shares of bioinformatics: $50/100 = 0.500000$ (i.e. ~50% of the compute resources are for bioinformatics).

Raw Shares of math: 30

Norm Shares of math: $30/100 = 0.300000$ (i.e. ~30% of the compute resources are for math).

Raw Shares of music: 5

Norm Shares of music: $5/100 = 0.050000$ (i.e. ~5% of the compute resources are for music).

Raw Shares of stats: 15

Norm Shares of stats: $15/100 = 0.150000$ (i.e. ~15% of the compute resources are for math).

Since no jobs have been submitted yet, the RawUsage is zero for all accounts. Likewise, the Effective Usage is also zero.

6. To see how Slurm is going to be assigning priority weights, use the sprio -w command:

`sprio -w`

Should show:

JOBID	PARTITION	PRIORITY	AGE	FAIRSHARE	PARTITION	QOS
Weights			1000	100000	10000	1000000

In this output, the weights (multipliers) are applied as:

AGE: 1000

FAIRSHARE: 100000

PARTITION: 10000

QOS: 1000000

7. As fred (who belongs to the *bioinformatics* account), submit this job 10 times, in a loop:

`for x in {1..10} ; do sbatch testping.sh ; done`

8. Let them run through completion.

9. As they run, launch sprio to see the job priorities:

`sprio`

Output:

JOBID	PARTITION	PRIORITY	AGE	FAIRSHARE	PARTITION	QOS
6	debug	110000	0	100000	10000	0
7	debug	110000	0	100000	10000	0
8	debug	110000	0	100000	10000	0
9	debug	110000	0	100000	10000	0
10	debug	110000	0	100000	10000	0

NOTE: The FAIRSHARE value is contributing 91% to the overall Priority (110000) for the jobs in the queue. This is because of the slurm.conf parameter: **PriorityWeightFairshare=100000**. The PARTITION value is contributing 9% to the overall Priority because of the slurm.conf parameter:

PriorityWeightPartition=10000

10. While the jobs are running, run the sshare command through watch to see the "Raw Usage" increment:

```
sshare --accounts=root,bioinformatics,math,music,stats
```

You should see:

Account	User	RawShares	NormShares	RawUsage	EffectvUsage	FairShare
root			0.000000	1535	1.000000	
root	root	0	0.000000	0	0.000000	0.600000
bioinformatics		50	0.500000	1535	0.000000	
math		30	0.300000	0	0.000000	
music		5	0.050000	0	0.000000	
stats		15	0.150000	0	0.000000	

NOTE: The RawUsage value increases (the number of cpu-seconds of all the jobs)

11. As **pebbles** (who belongs to the stats account), submit her testping.sh job 10 times, in a loop:
`for x in {1..10} ; do sbatch testping.sh ; done`
12. Let them run for a few minutes.
13. As they run, launch sprio to see the job priorities:

```
sprio
```

Output:

JOBID	PARTITION	PRIORITY	AGE	FAIRSHARE	PARTITION	QOS
16	debug	110000	0	100000	10000	0
17	debug	110000	0	100000	10000	0
18	debug	110000	0	100000	10000	0
19	debug	110000	0	100000	10000	0
20	debug	110000	0	100000	10000	0

The FAIRSHARE value is contributing 91% to the overall Priority (100000/110000) for the jobs in the queue

14. Again **as fred** (who submits to the bioinformatics account, and has already consumed resources), submit this job 10 times:

```
for x in {1..10} ; do sbatch testping.sh ; done
```

Then run sprio again, to see the fairshare adjust the priority:

JOBID	PARTITION	PRIORITY	AGE	FAIRSHARE	PARTITION	QOS
17	debug	85000	0	75000	10000	0
18	debug	85000	0	75000	10000	0
19	debug	85000	0	75000	10000	0
20	debug	85000	0	75000	10000	0
21	debug	75000	0	65000	10000	0
22	debug	75000	0	65000	10000	0
23	debug	75000	0	65000	10000	0
24	debug	75000	0	65000	10000	0
25	debug	75000	0	65000	10000	0
26	debug	75000	0	65000	10000	0
27	debug	75000	0	65000	10000	0
28	debug	75000	0	65000	10000	0
29	debug	75000	0	65000	10000	0
30	debug	75000	0	65000	10000	0

Notice the priority has been reduced because of the Fairshare priority being reduced. In this output, the higher-priority jobs belong to Pebbles.

15. Submit jobs back and forth between **pebbles** and **fred** and you will see how Slurm adjusts the priority based on usage.

16. Over time, you can run the sshare command and see the Effective Usage value changing. As mentioned, this is the percentage used of available cpu time:

```
sshare --accounts=bioinformatics,math,music,stats
```

Should show:

Account	User	RawShares	NormShares	RawUsage	EffectvUsage	FairShare
bioinformatics		50	0.500000	3158	0.667230	
math		30	0.300000	0	0.000000	
music		5	0.050000	0	0.000000	
stats		15	0.150000	1575	0.332770	

In this example, the percentage of usage of overall CPU usage is roughly 67% (EffectvUsage=0.667230) for bioinformatics, and 33% (EffectvUsage=0.332770) for stats account in the given fairshare window.

NOTE: It make take some time for the values to adjust

Cleanup

- Reset the Docker Slurm cluster environment back to its original, pristine state by exiting the login Docker container back to the Ubuntu AWS prompt. And **as ubuntu**:

```
cd ~/docker-scale-out  
make clean && make
```

Accounting Discussion

Slurm can be configured to collect accounting information for every job and job step executed. Accounting records can be written to a simple text file or a database. Information is available about both currently executing jobs and jobs which have already terminated. The `sacct` command can report resource usage for running or terminated jobs including individual tasks, which can be useful to detect load imbalance between the tasks. The `sstat` command can be used to status only currently running jobs. It also can give you valuable information about imbalance between tasks. The `sreport` can be used to generate reports based upon all jobs executed in a particular time interval.

There are three distinct plugin types associated with resource accounting. The Slurm configuration parameters (in `slurm.conf`) associated with these plugins include:

- **AccountingStorageType** controls how detailed job and job step information is recorded. You can store this information in a text file or into SlurmDBD.
- **JobAcctGatherType** is operating system dependent and controls what mechanism is used to collect accounting information. Supported values are `jobacct_gather/linux` (recommended), `jobacct_gather/cgroup` and `jobacct_gather/none` (no information collected).
- **JobCompType** controls how job completion information is recorded. This can be used to record basic job information such as job name, user name, allocated nodes, start time, completion time, exit status, etc. If the preservation of only basic job information is required, this plugin should satisfy your needs with minimal overhead. You can store this information in a text file, or MySQL or MariaDB database.

The use of `sacct` to view information about jobs is dependent upon `AccountingStorageType` being configured to collect and store that information. The use of `sreport` is dependent upon some database being used to store that information.

The use of `sacct` or `sstat` to view information about resource usage within jobs is dependent upon both `JobAcctGatherType` and `AccountingStorageType` being configured to collect and store that information.

Storing the accounting information into text files is very simple. Just configure the appropriate plugin (e.g. `AccountingStorageType=accounting_storage/filetxt` and/or `JobCompType=jobcomp/filetxt`) and then specify the pathname of the file (e.g. `AccountingStorageLoc=/var/log/slurm/accounting` and/or `JobCompLoc=/var/log/slurm/job_completions`). Use the logrotate or similar tool to prevent the log files from getting too large. Send a `SIGUSR2` signal to the `slurmctld` daemon after moving the files, but before compressing them so that new log files will be created.

Storing the data directly into a database from Slurm may seem attractive, but it requires the availability of user name and password data not only for the Slurm control daemon (`slurmctld`), but also for user commands which need to access the data (`sacct`, `sreport`, and `sacctmgr`). Making potentially sensitive information available to all users makes database security more difficult to provide. Sending the data through an intermediate daemon can provide better security and performance (through caching data). SlurmDBD (Slurm Database Daemon) provides such services. SlurmDBD is written in C, multi-threaded, secure and fast. The configuration required to use SlurmDBD will be described below. Storing information directly into a database would be similar.

Note that SlurmDBD relies upon existing Slurm plugins for authentication and Slurm sql for database use, but the other Slurm commands and daemons are not required on the host where SlurmDBD is installed. Install the `slurm` and `slurm-slurmdbd` RPMs on the server where SlurmDBD is to run.

Note if you switch from using the MySQL plugin to use the SlurmDBD plugin you must make sure the cluster has been added to the database. The MySQL plugin doesn't require this, but also will not hurt things if you have it there when using the MySQL plugin. You can verify with:

```
> sacctmgr list cluster
```

If the cluster isn't there add it (where my cluster's name was snowflake):

```
> sacctmgr add cluster snowflake
```

Failure to do so will result in the slurmctld failing to talk to the slurmdbd after the switch. If you plan to upgrade to a new version of Slurm don't switch plugins at the same time or you may get unexpected results. Do one then the other.

If SlurmDBD is configured for use but not responding then slurmctld will utilize an internal cache until SlurmDBD is returned to service. The cached data is written by slurmctld to local storage upon shutdown and recovered at startup. If SlurmDBD is not available when slurmctld starts, a cache of valid bank accounts, user limits, etc. based upon their state when the daemons were last communicating will be used. Note that SlurmDBD must be responding when slurmctld is first started since no cache of this critical data will be available. Job and step accounting records generated by slurmctld will be written to a cache as needed and transferred to SlurmDBD when returned to service.

Infrastructure

With the SlurmDBD, we are able to collect data from multiple clusters in a single location. This does impose some constraints on the user naming and IDs. Accounting is maintained by user name (not user ID), but a given user name should refer to the same person across all of the computers. Authentication relies upon user ID numbers, so those must be uniform across all computers communicating with each SlurmDBD, at least for users requiring authentication. In particular, the configured SlurmUser must have the same name and ID across all clusters. If you plan to have administrators of user accounts, limits, etc. they must also have consistent names and IDs across all clusters. If you plan to restrict access to accounting records (e.g. only permit a user to view records of his jobs), then all users should have consistent names and IDs.

NOTE: Only lowercase usernames are supported.

The best way to ensure security of the data is by authenticating communications to the SlurmDBD and we recommend MUNGE for that purpose. If you have one cluster managed by Slurm and execute the SlurmDBD on that one cluster, the normal MUNGE configuration will suffice. Otherwise MUNGE should then be installed on all nodes of all Slurm managed clusters, plus the machine where SlurmDBD executes. You then have a choice of either having a single MUNGE key for all of these computers or maintaining a unique key for each of the clusters plus a second key for communications between the clusters for better security. MUNGE enhancements are planned to support two keys within a single configuration file, but presently two different daemons must be started with different configurations to support two different keys (create two key files and start the daemons with the --key-file option to locate the proper key plus the --socket option to specify distinct local domain sockets for each). The pathname of local domain socket will be needed in the Slurm and SlurmDBD configuration files (slurm.conf and slurmdbd.conf respectively, more details are provided below).

Whether you use any authentication module or not you will need to have a way for the SlurmDBD to get uid's for users and/or admin. If using MUNGE, it is ideal for your users to have the same id on all your clusters. If this is the case you should have a combination of every clusters /etc/passwd file on the database server to allow the DBD to resolve names for authentication. If using MUNGE and a users name is not in the passwd file the action will fail. If not using MUNGE, you should add anyone you want to be an administrator or operator to the passwd file. If they plan on running sacctmgr or any of the accounting tools they should have the same uid, or they will not authenticate correctly. An LDAP server could also serve as a way to gather this information.

Slurm JobComp Configuration

Presently job completion is not supported with the SlurmDBD, but can be written directly to a database, script or flat file. If you are running with the accounting storage plugin, use of the job completion plugin is probably redundant. If you would like to configure this, some of the more important parameters include:

- **JobCompHost:** Only needed if using a database. The name or address of the host where the database server executes.
- **JobCompPass:** Only needed if using a database. Password for the user connecting to the database. Since the password can not be securely maintained, storing the information directly in a database is not recommended.
- **JobCompPort:** Only needed if using a database. The network port that the database accepts communication on.
- **JobCompType:** Type of jobcomp plugin set to "jobcomp/mysql" or "jobcomp/filetxt".
- **JobCompUser:** Only needed if using a database. User name to connect to the database with.

Slurm Accounting Configuration Before Build

While the SlurmDBD will work with a flat text file for recording job completions and such this configuration will not allow "associations" between a user and account. A database allows such a configuration.

MySQL or MariaDB is the preferred database. To enable this database support one only needs to have the development package for the database they wish to use on the system. Slurm uses the InnoDB storage engine in MySQL to make rollback possible. This must be available on your MySQL installation or rollback will not work.

The slurm configure script uses mysql_config to find out the information it needs about installed libraries and headers. You can specify where your mysql_config script is with the --with-mysql_conf=/path/to/mysql_config option when configuring your slurm build. On a successful configure, output is something like this:

```
checking for mysql_config...
/usr/bin/mysql_config
MySQL test program built properly.
```

NOTE: Before running the slurmdbd for the first time, review the current setting for MySQL's innodb_buffer_pool_size. Consider setting this value large enough to handle the size of the database. This helps when converting large tables over to the new database schema and when purging old records. Setting innodb_lock_wait_timeout and innodb_log_file_size to larger values than the default is also recommended.

ex.

```

mysql> SHOW VARIABLES LIKE 'innodb_buffer_pool_size';
+-----+-----+
| Variable_name      | Value   |
+-----+-----+
| innodb_buffer_pool_size | 134217728 |
+-----+-----+
1 row in set (0.00 sec)

$cat my.cnf
...
[mysqld]
innodb_buffer_pool_size=1024M
innodb_log_file_size=64M
innodb_lock_wait_timeout=900
...

```

Slurm Accounting Configuration After Build

For simplicity sake we are going to reference everything as if you are running with the SlurmDBD. You can communicate with a storage plugin directly, but that offers minimal security.

Several Slurm configuration parameters must be set to support archiving information in SlurmDBD. SlurmDBD has a separate configuration file which is documented in a separate section. Note that you can write accounting information to SlurmDBD while job completion records are written to a text file or not maintained at all. If you don't set the configuration parameters that begin with "AccountingStorage" then accounting information will not be referenced or recorded.

- **AccountingStorageEnforce:** This option contains a comma separated list of options you may want to enforce. The valid options are any comma separated combination of
 - **associations** This will prevent users from running jobs if their association is not in the database. This option will prevent users from accessing invalid accounts.
 - **limits** This will enforce limits set on associations and QoS'. By setting this option, the 'associations' option is automatically set. If a QoS is used the limits will be enforced, but 'qos' described below is still needed if you want to enforce access to the QoS.
 - **nojobs** This will make it so no job information is stored in accounting. By setting this 'nosteps' is also set.
 - **nosteps** This will make it so no step information is stored in accounting. Both nojobs and nosteps could be helpful in an environment where you want to use limits but don't really care about utilization.
 - **qos** This will require all jobs to specify (either overtly or by default) a valid QoS (Quality of Service). QOS values are defined for each association in the database. By setting this option, the 'associations' option is automatically set. If you want QOS limits to be enforced you need to use the 'limits' option.

- **safe** This will ensure a job will only be launched when using an association or QoS that has a GrpTRESMins limit set if the job will be able to run to completion. Without this option set, jobs will be launched as long as their usage hasn't reached the TRES-minutes limit which can lead to jobs being launched but then killed when the limit is reached. By setting this option, both the 'associations' option and the 'limits' option are set automatically.
- **wckeys** This will prevent users from running jobs under a wckey that they don't have access to. By using this option, the 'associations' option is automatically set. The 'TrackWCKey' option is also set to true.

(**NOTE:** The association is a combination of cluster, account, user names and optional partition name.) Without AccountingStorageEnforce being set (the default behavior) jobs will be executed based upon policies configured in Slurm on each cluster.

- **AccountingStorageHost:** The name or address of the host where SlurmDBD executes
- **AccountingStoragePass:** If using SlurmDBD with a second MUNGE daemon, store the pathname of the named socket used by MUNGE to provide enterprise-wide authentication (i.e. /var/run/munge/moab.socket.2). Otherwise the default MUNGE daemon will be used.
- **AccountingStoragePort:** The network port that SlurmDBD accepts communication on
- **AccountingStorageType:** Set to "accounting_storage/slurmdbd".
- **ClusterName:** Set to a unique name for each Slurm-managed cluster so that accounting records from each can be identified.
- **TrackWCKey:** Boolean. If you want to track wckeys (Workload Characterization Key) of users. A Wckey is an orthogonal way to do accounting against possibly unrelated accounts. When a job is run, use the --wckey option to specify a value and accounting records will be collected by this wckey.

SlurmDBD Configuration

SlurmDBD requires its own configuration file called `slurmdbd.conf`. This file should be only on the computer where SlurmDBD executes and should only be readable by the user which executes SlurmDBD (e.g. "slurm"). This file should be protected from unauthorized access since it contains a database login name and password. See `man slurmdbd.conf` for a more complete description of the configuration parameters. Some of the more important parameters include:

- **AuthInfo:** If using SlurmDBD with a second MUNGE daemon, store the pathname of the named socket used by MUNGE to provide enterprise-wide. Otherwise the default MUNGE daemon will be used.
- **AuthType:** Define the authentication method for communications between Slurm components. A value of "auth/munge" is recommended.
- **DbdHost:** The name of the machine where the Slurm Database Daemon is executed. This should be a node name without the full domain name (e.g. "lx0001"). This defaults to localhost but should be supplied to avoid a warning message.

- **DbdPort:** The port number that the Slurm Database Daemon (slurmdbd) listens to for work. The default value is SLURMDBD_PORT as established at system build time. If none is explicitly specified, it will be set to 6819. This value must be equal to the AccountingStoragePort parameter in the slurm.conf file.
- **LogFile:** Fully qualified pathname of a file into which the Slurm Database Daemon's logs are written. The default value is none (performs logging via syslog).
- **PluginDir:** Identifies the places in which to look for Slurm plugins. This is a colon-separated list of directories, like the PATH environment variable. The default value is the prefix given at configure time + "/lib/slurm".
- **SlurmUser:** The name of the user that the slurmcld daemon executes as. This user must exist on the machine executing the Slurm Database Daemon and have the same user ID as the hosts on which slurmcld execute. For security purposes, a user other than "root" is recommended. The default value is "root". This name should also be the same SlurmUser on all clusters reporting to the SlurmDBD.
- **StorageHost:** Define the name of the host the database is running where we are going to store the data. Ideally this should be the host on which SlurmDBD executes. But could be a different machine.
- **StorageLoc:** Specifies the name of the database where accounting records are written. For databases the default database is slurm_acct_db. Note the name can not have a '/' in it or the default will be used.
- **StoragePass:** Define the password used to gain access to the database to store the job accounting data.
- **StoragePort:** Define the port on which the database is listening.
- **StorageType:** Define the accounting storage mechanism is. The only acceptable value at present is "accounting_storage/mysql". The value "accounting_storage/mysql" indicates that accounting records should be written to a MySQL or MariaDB database specified by the StorageLoc parameter. This value must be specified.
- **StorageUser:** Define the name of the user we are going to connect to the database with to store the job accounting data.

MySQL Configuration

While Slurm will create the database tables automatically you will need to make sure the StorageUser is given permissions in the MySQL or MariaDB database to do so. As the mysql user grant privileges to that user using a command such as:

```
GRANT ALL ON StorageLoc.* TO 'StorageUser'@'StorageHost';
```

(The ticks [single quotes] are needed.)

(You need to be root to do this. Also in the info for password usage there is a line that starts with '->'. This a continuation prompt since the previous mysql statement did not end with a ';'. It assumes that you wish to input more info.)

If you want Slurm to create the database itself, and any future databases, you can change your grant line to be *.* instead of StorageLoc.*

Live example:

```
mysql@snowflake:~$ mysql
Welcome to the MySQL monitor. Commands end with ; or \g.
Your MySQL connection id is 538
Server version: 5.0.51a-3ubuntu5.1 (Ubuntu)

Type 'help;' or '\h' for help. Type '\c' to clear the buffer.

mysql> create user 'slurm'@'localhost' identified by 'password';
Query OK, 0 rows affected (0.00 sec)

mysql> grant all on slurm_acct_db.* TO 'slurm'@'localhost';
Query OK, 0 rows affected (0.00 sec)
```

You may also need to do the same with the system name in order for mysql to work correctly:

```
mysql> grant all on slurm_acct_db.* TO 'slurm'@'system0';
Query OK, 0 rows affected (0.00 sec)
where 'system0' is the localhost or database storage host.
```

Or with a password (in the following, the command line wraps and the '->' is a continuation prompt)...

```
mysql> grant all on slurm_acct_db.* TO 'slurm'@'localhost'
-> identified by 'some_pass' with grant option;
Query OK, 0 rows affected (0.00 sec)
```

The same is true in the case, you made to do the same with the system name:

```
mysql> grant all on slurm_acct_db.* TO 'slurm'@'system0'
-> identified by 'some_pass' with grant option;
where 'system0' is the localhost or database storage host.
```

Verify you have InnoDB support

```
mysql> SHOW ENGINES;
+-----+-----+-----+-----+-----+
| Engine | Support | Comment          | Transactions | XA    | Savepoints |
+-----+-----+-----+-----+-----+
| ...    |         |                 |             |       |           |
| InnoDB | DEFAULT | Supports transactions, row-level locking, and foreign keys | YES        | YES   | YES       |
| ...    |         |                 |             |       |           |
+-----+-----+-----+-----+-----+
```

Then create the database:

```
mysql> create database slurm_acct_db;
```

This will grant user 'slurm' access to do what it needs to do on the local host or the storage host system. This must be done before the SlurmDBD will work properly. After you grant permission to the user 'slurm' in mysql then you can start SlurmDBD and the other Slurm daemons. You start SlurmDBD by typing its pathname:

```
> /usr/sbin/slurmdbd'
```

or

```
> /etc/init.d/slurmdbd start
```

You can verify that SlurmDBD is running by typing...

```
> /etc/init.d/slurmdbd status
```

If the SlurmDBD is not running you can use the -v option when you start SlurmDBD to get more detailed information. Starting the SlurmDBD in daemon mode with the '-D' option can also help in debugging so you don't have to go to the log to find the problem

Tools

There are a few tools available to work with accounting data, **sacct**, **sacctmgr**, and **sreport**. These tools all get or set data through the SlurmDBD daemon.

- **sacct** Used to generate accounting reports for both running and completed jobs
- **sacctmgr** Used to manage associations in the database: add or remove clusters, add or remove users, etc.
- **sreport** Used to generate various reports on usage collected over a given time period

See the man pages for each command for more information.

Database Configuration

Accounting records are maintained based upon what we refer to as an Association, which consists of four elements: cluster, account, user names and an optional partition name. Use the **sacctmgr** command to create and manage these records.

NOTE: There is an order to set up accounting associations. You must define clusters before you add accounts and you must add accounts before you can add users.

For example, to add a cluster named "snowflake" to the database execute this line:

```
> sacctmgr add cluster snowflake
```

Add accounts "none" and "test" to cluster "snowflake" with an execute line of this sort:

```
> sacctmgr add account none,test Cluster=snowflake Description="none" Organization="none"
```

If you have more clusters you want to add these accounts, to you can either not specify a cluster, which will add the accounts to all clusters in the system, or comma separate the cluster names you want to add to in the cluster option. Note that multiple accounts can be added at the same time by comma separating the names. A description of the account and the organization which it belongs to must be specified. These terms can be used later to generated accounting reports. Accounts may be arranged in a hierarchical fashion, for example accounts chemistry and physics may be children of the account science. The hierarchy may have an arbitrary depth. Just specify the *parent=* option in the add account line to construct the hierarchy. For the example above execute

```
> sacctmgr add account science Description="science accounts" Organization=science  
> sacctmgr add account chemistry,physics parent=science Description="physical sciences" Organization=science
```

Add users to accounts using similar syntax. For example, to permit user da to execute jobs on all clusters with a default account of test execute:

```
> sacctmgr add user brian Account=physics  
> sacctmgr add user da DefaultAccount=test
```

If **AccountingStorageEnforce=associations** is configured in the *slurm.conf* of the cluster snowflake then user da would be allowed to run in account test and any other accounts added in the future. Any attempt to use other accounts will result in the job being aborted. Account test will be the default if he doesn't specify one in the job submission command.

Partition names can also be added to an "add user" command with the Partition='partitionname' option to specify an association specific to a slurm partition.

sacctmgr dump

One of the great things about using a database for maintaining access control and account definitions, is the ability to dump and restore the configuration.

To dump the current database configuration, use the following syntax:

```
> sacctmgr dump <CLUSTERID> file=<FILENAME>
```

The output looks something like this:

```
[root@login ~]# sacctmgr dump cluster file=cluster.cfg  
sacctmgr: Cluster - 'cluster':Fairshare=1:QOS='normal'  
sacctmgr: Parent - 'root'  
sacctmgr: User - 'root':DefaultAccount='bedrock':AdminLevel='Administrator':Fairshare=1  
sacctmgr: Account - 'bedrock':Description='none':Organization='none':Fairshare=1  
sacctmgr: Account - 'managers':Description='managers':Organization='none':Fairshare=1  
sacctmgr: Account - 'science':Description='science department':Organization='science':Fairshare=1  
sacctmgr: Parent - 'bedrock'  
sacctmgr: User - 'arnold':DefaultAccount='bedrock':Coordinator='bioinformatics':Fairshare=1  
sacctmgr: User - 'bambam':DefaultAccount='bedrock':Coordinator='chemistry':Fairshare=1  
sacctmgr: User - 'barney':DefaultAccount='bedrock':Fairshare=1  
sacctmgr: User - 'betty':DefaultAccount='bedrock':Coordinator='math':Fairshare=1  
sacctmgr: User - 'chip':DefaultAccount='bedrock':Fairshare=1
```

```

sacctmgr: User - 'dino':DefaultAccount='bedrock':Fairshare=1
sacctmgr: User - 'edna':DefaultAccount='bedrock':Fairshare=1
sacctmgr: User - 'fred':DefaultAccount='bedrock':Fairshare=1
sacctmgr: User - 'gazoo':DefaultAccount='bedrock':Fairshare=1
sacctmgr: User - 'pebbles':DefaultAccount='bedrock':Fairshare=1
sacctmgr: User - 'root':DefaultAccount='bedrock':AdminLevel='Administrator':Fairshare=1
sacctmgr: User - 'wilma':DefaultAccount='bedrock':Fairshare=1
sacctmgr: Parent - 'managers'
sacctmgr: User - 'barney':DefaultAccount='bedrock':Fairshare=1
sacctmgr: User - 'fred':DefaultAccount='bedrock':Fairshare=1
sacctmgr: Parent - 'science'
sacctmgr: Account - 'bioinformatics':Description='bioinformatics'
department':Organization='science':Fairshare=1
sacctmgr: Account - 'chemistry':Description='chemistry department':Organization='science':Fairshare=1
sacctmgr: Account - 'math':Description='math department':Organization='science':Fairshare=1
sacctmgr: Parent - 'bioinformatics'
sacctmgr: User - 'arnold':DefaultAccount='bedrock':Coordinator='bioinformatics':Fairshare=1
sacctmgr: User - 'edna':DefaultAccount='bedrock':Fairshare=1
sacctmgr: Parent - 'chemistry'
sacctmgr: User - 'bam bam':DefaultAccount='bedrock':Coordinator='chemistry':Fairshare=1
sacctmgr: User - 'pebbles':DefaultAccount='bedrock':Fairshare=1
sacctmgr: Parent - 'math'
sacctmgr: User - 'betty':DefaultAccount='bedrock':Coordinator='math':Fairshare=1
sacctmgr: User - 'wilma':DefaultAccount='bedrock':Fairshare=1

```

Once you have the output file, it's possible to make changes to it, then upload it back into the database:

```
> sacctmgr load <FILENAME>
```

Example

NOTE: There is an order to set up accounting associations. You must define clusters before you add accounts and you must add accounts before you can add users.

```

> sacctmgr create cluster tux
> sacctmgr create account name=science fairshare=50
> sacctmgr create account name=chemistry parent=science fairshare=30
> sacctmgr create account name=physics parent=science fairshare=20
> sacctmgr create user name=adam cluster=tux account=physics fairshare=10
> sacctmgr delete user name=adam cluster=tux account=physics
> sacctmgr delete account name=physics cluster=tux
> sacctmgr modify user where name=adam cluster=tux account=physics set maxjobs=2

```

(Example commands continue on next page.)

```

> maxwall=30:00
> sacctmgr add user brian account=chemistry
> sacctmgr list associations cluster=tux format=Account,Cluster,User,Fairshare tree withd
> sacctmgr list transactions Action="Add Users" Start=11/03-10:30:00 format=Where,Time
> sacctmgr dump cluster=tux file=tux_data_file
> sacctmgr load tux_data_file

```

A user's account can not be changed directly. A new association needs to be created for the user with the new account. Then the association with the old account can be deleted.

When modifying an object placing the key words 'set' and the optional 'where' is critical to perform correctly below are examples to produce correct results. As a rule of thumb anything you put in front of the set will be used as a quantifier. If you want to put a quantifier after the key word 'set' you should use the key word 'where'.

wrong -> `sacctmgr modify user name=adam set fairshare=10 cluster=tux`

This will produce an error as the above line reads modify user adam set fairshare=10 and cluster=tux.

right-> `sacctmgr modify user name=adam cluster=tux set fairshare=10`

right-> `sacctmgr modify user name=adam set fairshare=10 where cluster=tux`

When changing QoS for something, only use the '=' operator when wanting to explicitly set the QoS to something.

In most cases you will want to use the '+=' or '-=' operator to either add to or remove from the existing QoS already in place.

If a user already has QoS of normal,standby for a parent or it was explicitly set you should use qos+=expedite to add this to the list in this fashion.

If you are looking to only add the QoS expedite to only a certain account and or cluster you can do that by specifying them in the `sacctmgr` line.

```
> sacctmgr modify user name=adam set qos+=expedite  
> sacctmgr modify user name=adam acct=this cluster=tux set
```

Let's give an example how to add QOS to user accounts.

List all available QOSs in the cluster.

```
> sacctmgr show qos format=name  
      Name  
-----  
    normal  
  expedite
```

List all the associations in the cluster.

```
> sacctmgr show assoc format=cluster,account,qos  
Cluster   Account     QOS  
-----  -----  -----  
zebra      root       normal  
zebra      root       normal  
zebra        g        normal  
zebra       g1        normal
```

Add the QOS expedite to account G1 and display the result. Using the operator += the QOS will be added together with the existing QOS to this account.

```
> sacctmgr show assoc format=cluster,account,qos
Cluster    Account    QOS
-----
zebra      root       normal
zebra      root       normal
zebra      g          normal
zebra      g1         normal
```

```
> sacctmgr modify account name=g1 set qos+=expedite
> sacctmgr show assoc format=cluster,account,qos
Cluster    Account    QOS
-----
zebra      root       normal
zebra      root       normal
zebra      g          normal
zebra      g1         expedite,normal
```

Now set the QOS expedite as the only QOS for the account G and display the result. Using the operator = that expedite is the only usable QOS by account G

```
> sacctmgr modify account name=G set qos=expedite
> sacctmgr show assoc format=cluster,account,user,qos
Cluster    Account    QOS
-----
zebra      root       normal
zebra      root       normal
zebra      g          expedite
zebra      g1         expedite,normal
```

If a new account is added under the account G it will inherit the QOS expedite and it will not have access to QOS normal.

```
> sacctmgr add account banana parent=G
> sacctmgr show assoc format=cluster,account,qos
Cluster    Account    QOS
-----
zebra      root       normal
zebra      root       normal
zebra      g          expedite
zebra      banana     expedite
zebra      g1         expedite,normal
```

An example of listing trackable resources

> sacctmgr show tres		
Type	Name	ID
cpu		1
mem		2
energy		3
node		4
billing		5
gres	gpu:tesla	1001
license	vcs	1002
bb	cray	1003

Database Maintenance

A database with very many job records (maybe of the order of a million) can cause issues when upgrading Slurm and the database.

In order to solve this problem, it is recommended to purge job records from the Slurm database. In slurmdbd.conf you may define a number of purge parameters such as:

- **PurgeEventAfter**
- **PurgeJobAfter**
- **PurgeResvAfter**
- **PurgeStepAfter**
- **PurgeUsageAfter**

The values of these parameters depend on the number of jobs in the database, which differs between sites. There does not seem to be any heuristics for determining good values, so some testing will be required.

From the high_throughput page: You might also consider setting the Purge options in your slurmdbd.conf to clear out old Data. A typical configuration might look like this:

```
PurgeEventAfter=12months
PurgeJobAfter=12months
PurgeResvAfter=2months
PurgeStepAfter=2months
PurgeSuspendAfter=1month
PurgeTXNAfter=12months
PurgeUsageAfter=12months
```

The purge operation is done at the start of each time interval, which means on the 1st day of the month in this example.

Monthly, daily or even hourly purge operations would occur when using different time units for the same interval:

```
PurgeStepAfter=2months
PurgeStepAfter=60days
PurgeStepAfter=1440hours
```

Logging of purge events can be configured in slurmdbd.conf using:

```
DebugLevel=verbose  
DebugFlags=DB_ARCHIVE
```

In Slurm V23.02, purge and archive functionality for job environment and job batch script records have been added.

Cluster Options

When either adding or modifying a cluster, these are the options available with `sacctmgr`:

- **Name** Cluster name

Account Options

When either adding or modifying an account, the following `sacctmgr` options are available:

- **Cluster** Only add this account to these clusters. The account is added to all defined clusters by default.
- **Description** Description of the account. (Default is account name)
- **Name** Name of account. Note the name must be unique and can not represent different bank accounts at different points in the account hierarchy
- **Organization** Organization of the account. (Default is parent account unless parent account is root then organization is set to the account name.)
- **Parent** Make this account a child of this other account (already added).

User Options

When either adding or modifying a user, the following `sacctmgr` options are available:

- **Account** Account(s) to add user to
- **AdminLevel** This field is used to allow a user to add accounting privileges to this user.

Valid options are:

- **None**
 - **Operator:** Can add, modify, and remove any database object (user, account, etc), and add other operators
On a SlurmDBD served slurmctld these users can
 - View information that is blocked to regular uses by a PrivateData flag
 - Create/Alter/Delete Reservations
 - **Admin:** These users have the same level of privileges as an operator in the database. They can also alter anything on a served slurmctld as if they were the slurm user or root.
- **Cluster** Only add to accounts on these clusters (default is all clusters)
 - **DefaultAccount** Default account for the user, used when no account is specified when a job is submitted.
(Required on creation)

- **DefaultWCKey** Default wckey for the user, used when no wckey is specified when a job is submitted.
(Only used when tracking wckeys.)
- **Name** User name
- **NewName** Use to rename a user in the accounting database
- **Partition** Name of Slurm partition this association applies to

Limit Enforcement

Various limits and limit enforcement are described in the Resource Limits web page.

To enable any limit enforcement you must at least have AccountingStorageEnforce=limits in your slurm.conf, otherwise, even if you have limits set, they will not be enforced. Other options for AccountingStorageEnforce and the explanation for each are found on the Resource Limits document.

Modifying Entities

When modifying entities, you can specify many different options in SQL-like fashion, using key words like where and set. A typical execute line has the following form:

```
> sacctmgr modify <entity> set <options> where <options>
```

For example:

```
> sacctmgr modify user set default=None where default=test
```

will change all users with a default account of "test" to account "none". Once an entity has been added, modified or removed, the change is sent to the appropriate Slurm daemons and will be available for use instantly.

Removing Entities

Removing entities using an execute line similar to the modify example above, but without the set options. For example, remove all users with a default account "test" using the following execute line:

```
> sacctmgr remove user where default=test
```

will remove all user records where the default account is "test".

```
> sacctmgr remove user brian where account=physics
```

will remove user "brian" from account "physics". If user "brian" has access to other accounts, those user records will remain.

Note: In most cases, removed entities are preserved, but flagged as deleted. If an entity has existed for less than 1 day, the entity will be removed completely. This is meant to clean up after typographic errors.

Accounting Exercises

In this set of labs, you will learn how to configure the MySQL database through SlurmDBD. You will create entries in the accounting database, modify them, and place limits/constraints on the ACLs.

Exercise 1: Use the `sacct` command to view status information of a running job/step:

`sacct` is used to view the database information of jobs.

1. As **fred**, submit a job as follows:

```
sbatch -N1 --wrap="hostname"
```

Should give you a jobID (probably 1, if you reset the learning environment at the end of the last lab exercise)

2. Check the contents of the submission:

```
cat slurm-1.out
```

Should give you node00, the node on which the command "hostname" ran

3. Since the job ran to completion, Slurm should have updated the database. To see the results of the job run, use `sacct`:

```
sacct
```

You should see:

JobID	JobName	Partition	Account	AllocCPUS	State	ExitCode
1	wrap	debug	bedrock	4	COMPLETED	0:0
1.batch	batch		bedrock	4	COMPLETED	0:0
1.extern	extern		bedrock	4	COMPLETED	0:0

4. To show the brief output, use the `-b` switch:

```
sacct -b
```

You should see:

JobID	State	ExitCode
1	COMPLETED	0:0
1.batch	COMPLETED	0:0
1.extern	COMPLETED	0:0

5. If you want to see just the job allocation, not taking into consideration the steps, run this:

```
sacct -x
```

You should see:

JobID	JobName	Partition	Account	AllocCPUS	State	ExitCode
1	wrap	debug	bedrock	4	COMPLETED	0:0

6. If you want to see ALL the fields in the database, you can use the -l (long) switch:

sacct -l

You should see:

JobID	JobIDRaw	JobName	Partition	MaxVMSize	MaxVMSizeNode	MaxVMSizeTask	AveVMSize	
MaxRSS	MaxRSSNode	MaxRSSTask	AveRSS	MaxPages	MaxPagesNode	MaxPagesTask	AvePages	MinCPU
MinCPUNode	MinCPUTask	AveCPU	NTasks	AllocCPUS	Elapsed	State	ExitCode	AveCPUFreq
ReqCPUFreqMin	ReqCPUFreqMax	ReqCPUFreqGov		ReqMem	ConsumedEnergy	MaxDiskRead	MaxDiskReadNode	
MaxDiskReadTask	AveDiskRead	MaxDiskWrite	MaxDiskWriteNode	MaxDiskWriteTask	AveDiskWrite			
AllocGRES	ReqGRES	ReqTRES	AllocTRES	TRESUsageInAve	TRESUsageInMax	TRESUsageInMaxNode		
TRESUsageInMaxTask	TRESUsageInMin	TRESUsageInMinNode	TRESUsageInMinTask	TRESUsageInTot				
TRESUsageOutMax	TRESUsageOutMaxNode	TRESUsageOutMaxTask	TRESUsageOutAve	TRESUsageOutTot				
1	1	wrap	debug					
1	00:00:01	COMPLETED	0:0					
0								
billing=1+ billing=2+								
1.batch	2.batch	batch		146060K	node00		0	146060K
0 node00	0	0	0	node00	0	0	00:00:00	
node00	0	00:00:00	1	2	00:00:01	COMPLETED	0:0	2.30M
0	0	0	0n	0	0	0	node00	
0	0	0	node00		0	0		
cpu=2,mem+ cpu=00:00:00,+ cpu=00:00:00,+ cpu=node00,energy+ cpu=0,fs/disk=0,m+ cpu=00:00:00,+								
cpu=node00,energy+ cpu=0,fs/disk=0,m+ cpu=00:00:00,+ energy=0,fs/di+ energy=node00,fs/d+								
fs/disk=0 energy=0,fs/di+ energy=0,fs/di+								
1.extern	2.extern	extern		145712K	node00		0	145712K
0 node00	0	0	0	node00	0	0	00:00:00	
node00	0	00:00:00	1	2	00:00:01	COMPLETED	0:0	2.30M
0	0	0	0n	0	0	0	node00	
0	0	0	node00		0	0		
billing=2+ cpu=00:00:00,+ cpu=00:00:00,+ cpu=node00,energy+ cpu=0,fs/disk=0,m+ cpu=00:00:00,+								
cpu=node00,energy+ cpu=0,fs/disk=0,m+ cpu=00:00:00,+ energy=0,fs/di+ energy=node00,fs/d+								
fs/disk=0 energy=0,fs/di+ energy=0,fs/di+								

7. To see the same output in a nicer way, you can use the “Parsable” switch:

sacct -l -p

You should see:

```

JobID|JobIDRaw|JobName|Partition|MaxVMSize|MaxVMSizeNode|MaxVMSizeTask|AveVMSize|MaxRSS|MaxRSSNode|MaxRSSTask|AveRSS|MaxPages|MaxPagesNode|MaxPagesTask|AvePages|MinCPU|MinCPUNode|MinCPUTask|AveCPU|NTAsks|AllocCPUS|Elapsed|State|ExitCode|AveCPUFreq|ReqCPUFreqMin|ReqCPUFreqMax|ReqCPUFreqGov|ReqMem|ConsumedEnergy|MaxDiskRead|MaxDiskReadNode|MaxDiskReadTask|AveDiskRead|MaxDiskWrite|MaxDiskWriteNode|MaxDiskWriteTask|AveDiskWrite|AllocGRES|ReqGRES|ReqTRES|AllocTRES|TRESUsageInAve|TRESUsageInMax|TRESUsageInMaxNode|TRESUsageInMaxTask|TRESUsageInMin|TRESUsageInMinNode|TRESUsageInMinTask|TRESUsageInTotal|TRESUsageOutMax|TRESUsageOutMaxNode|TRESUsageOutMaxTask|TRESUsageOutAve|TRESUsageOutTotal|
1|1|wrap|debug|||||||||4|00:00:01|COMPLETED|0:0||Unknown|Unknown|Unknown|On|0|||||||billing=1,cpu=1,node=1|billing=4,cpu=4,node=1|||||||
1.batch|2.batch|batch||342840K|node00|0|342840K|0|node00|0|0|0|node00|0|0|0:00:00|node00|0|0:00:00|
|1|4|00:00:01|COMPLETED|0:0|2.30M|0|0|0|0n|0|0|node00|0|0|0|node00|0|0|1||cpu=4,mem=0,node=1|cpu=0:00:00,energy=0,fs/disk=0,mem=0,pages=0,vmem=342840K|cpu=0:00:00,energy=0,fs/disk=0,mem=0,pages=0,vmem=342840K|cpu=node00,energy=node00,fs/disk=node00,mem=node00,pages=node00,vmem=node00|cpu=0,fs/disk=0,mem=0,pages=0,vmem=0|cpu=0:00:00,energy=0,fs/disk=0,mem=0,pages=0,vmem=342840K|cpu=node00,energy=node00,fs/disk=node00,mem=node00,pages=node00,vmem=node00|cpu=0,fs/disk=0,mem=0,pages=0,vmem=0|cpu=0:00:00,energy=0,fs/disk=0,mem=0,pages=0,vmem=342840K|cpu=node00,energy=node00,fs/disk=node00,mem=node00,pages=node00,vmem=node00|cpu=0,fs/disk=0,mem=0,pages=0,vmem=0|cpu=0:00:00,energy=0,fs/disk=0,mem=0,pages=0,vmem=4372K|cpu=node00,energy=node00,fs/disk=node00,mem=node00,pages=node00,vmem=node00|cpu=0,fs/disk=0,mem=0,pages=0,vmem=0|cpu=0:00:00,energy=0,fs/disk=2076,mem=0,pages=0,vmem=4372K|cpu=0:00:00,energy=0,fs/disk=2076,mem=0,pages=0,vmem=4372K|cpu=node00,energy=node00,fs/disk=node00,mem=node00,pages=node00,vmem=node00|cpu=0,fs/disk=0,mem=0,pages=0,vmem=0|cpu=0:00:00,energy=0,fs/disk=2076,mem=0,pages=0,vmem=4372K|cpu=0:00:00,energy=0,fs/disk=2076,mem=0,pages=0,vmem=4372K|cpu=node00,energy=node00,fs/disk=node00,mem=node00,pages=node00,vmem=node00|cpu=0,fs/disk=0,mem=0,pages=0,vmem=0|cpu=0:00:00,energy=0,fs/disk=2076,mem=0,pages=0,vmem=4372K|cpu=0:00:00,energy=0,fs/disk=2076,mem=0,pages=0,vmem=4372K|cpu=node00,fs/disk=0|energy=0,fs/disk=0|energy=0,fs/disk=0|
1.extern|2.extern|extern||4372K|node00|0|4372K|0|node00|0|0|0|node00|0|0|0:00:00|node00|0|0:00:00|
|1|4|00:00:01|COMPLETED|0:0|2.30M|0|0|0|0n|0|0|0.00M|node00|0|0.00M|0|node00|0|0|1||cpu=4,cpu=4,nod
e=1|cpu=0:00:00,energy=0,fs/disk=2076,mem=0,pages=0,vmem=4372K|cpu=0:00:00,energy=0,fs/disk=2076,mem=0,pages=0,vmem=4372K|cpu=node00,energy=node00,fs/disk=node00,mem=node00,pages=node00,vmem=node00|cpu=0,fs/disk=0,mem=0,pages=0,vmem=0|cpu=0:00:00,energy=0,fs/disk=2076,mem=0,pages=0,vmem=4372K|cpu=0:00:00,energy=0,fs/disk=2076,mem=0,pages=0,vmem=4372K|cpu=node00,energy=node00,fs/disk=node00,mem=node00,pages=node00,vmem=node00|cpu=0,fs/disk=0,mem=0,pages=0,vmem=0|cpu=0:00:00,energy=0,fs/disk=2076,mem=0,pages=0,vmem=4372K|cpu=0:00:00,energy=0,fs/disk=2076,mem=0,pages=0,vmem=4372K|cpu=node00,fs/disk=0|energy=0,fs/disk=0|energy=0,fs/disk=0|

```

8. Or, you can combine the -X, -l, and -p to get:

sacct -X -l -p

You should see:

```

JobID|JobIDRaw|JobName|Partition|MaxVMSize|MaxVMSizeNode|MaxVMSizeTask|AveVMSize|MaxRSS|MaxRSSNode|MaxRSSTask|AveRSS|MaxPages|MaxPagesNode|MaxPagesTask|AvePages|MinCPU|MinCPUNode|MinCPUTask|AveCPU|NTAsks|AllocCPUS|Elapsed|State|ExitCode|AveCPUFreq|ReqCPUFreqMin|ReqCPUFreqMax|ReqCPUFreqGov|ReqMem|ConsumedEnergy|MaxDiskRead|MaxDiskReadNode|MaxDiskReadTask|AveDiskRead|MaxDiskWrite|MaxDiskWriteNode|MaxDiskWriteTask|AveDiskWrite|AllocGRES|ReqGRES|ReqTRES|AllocTRES|TRESUsageInAve|TRESUsageInMax|TRESUsageInMaxNode|TRESUsageInMaxTask|TRESUsageInMin|TRESUsageInMinNode|TRESUsageInMinTask|TRESUsageInTotal|TRESUsageOutMax|TRESUsageOutMaxNode|TRESUsageOutMaxTask|TRESUsageOutAve|TRESUsageOutTotal|
1|1|wrap|debug|||||||||4|00:00:01|COMPLETED|0:0||Unknown|Unknown|Unknown|On|0|||||||billing=1,cpu=1,node=1|billing=4,cpu=4,node=1|||||||

```

9. There are some values that are NOT included in the -l switch. Using --format option, you can narrow down which fields you want to see, for example, on which nodes did my job run:

sacct --format JobID,Account,NNodes,NodeList

Should show:

JobID	Account	NNodes	NodeList
1	bedrock	1	node00
1.batch	bedrock	1	node00
1.extern	bedrock	1	node00

- Another thing you can do is format the width of each field. To shrink the columns to exclude so much whitespace between them, you can force the column width with the %value. Use:
`sacct --format JobID%9,Account%7,NNodes%6,Nodelist%8`

It should show:

JobID	Account	NNodes	NodeList
1	bedrock	1	node00
1.batch	bedrock	1	node00
1.extern	bedrock	1	node00

REMEMBER: The `sacct` command is driven by rights. If you run it as a normal user, without specifying any switches, you will only see that users' information. If you run it as Slurm admin or root, you can specify which user by using `-u <username>` switch. To see ALL users, use `-a` (which you can also call as a normal user, unless `PrivateData` has been enabled).

Exercise 2: Use the `sacctmgr` command to manipulate the database information:

The `sacctmgr` command is used to manage associations in the database: add or remove clusters, add or remove users, etc. In this exercise, you will create a QoS, assign a default job priority to it, and associate users to the QoS as a default so they can submit jobs to it.

- Enable priority/multifactor plugin so priority can be accounted for. Edit the `/etc/slurm/slurm.conf` and change:

#PriorityType=priority/multifactor

To:

PriorityType=priority/multifactor

(I.e. remove the hash symbol from the beginning of the line.)

- As root user, create a new QoS called "High":

```
sacctmgr -i add qos high
```

- Allow fred and barney to submit to the "High" QoS:

```
sacctmgr -i modify user fred,barney set qos==high
```

- Assign a high priority to the high QoS:

```
sacctmgr -i modify qos high set priority=10000
```

5. Assign a priority weight to the QoS:

```
echo "PriorityWeightQOS=10000" >> /etc/slurm/slurm.conf
```

6. Restart the controller:

```
/lab_scripts/restart.sh
```

7. View the QoS configuration:

```
sacctmgr show qos format=Name,Priority
```

Should show:

Name	Priority
normal	0
high	10000

8. Submit 2 jobs **as wilma** taking up the entire cluster:

```
sbatch --mem=1000 -N10 -n20 -t1000 --wrap="sleep 1000"
```

```
sbatch --mem=1000 -N10 -n20 -t1000 --wrap="sleep 1000"
```

9. Now **as fred**, submit a similar job, but send it to the "High" QoS:

```
sbatch -qhigh --mem=1000 -N10 -n20 -t1000 --wrap="sleep 1000"
```

10. To see the jobs in the queue, with their priority, enter the following:

```
squeue -o "%.18i %.9P %.8j %.8u %.8T %.10M %.91 %.6D %R %Q"
```

You should see:

JOBID	PARTITION	NAME	USER	STATE	TIME	TIME_LIMI	NODES	NODELIST (REASON)	PRIORITY
4	debug	wrap	fred	PENDING	0:00	16:40:00	10	(Resources)	10000
3	debug	wrap	wilma	PENDING	0:00	16:40:00	10	(Priority)	1
2	debug	wrap	wilma	RUNNING	1:47	16:40:00	10	node[00-09]	1

11. Using sprio, show the Pending job's priority (sprio by default prints all pending jobs' priority):

```
sprio
```

Should show:

JOBID	PARTITION	PRIORITY	SITE	QOS
3	debug	1	0	0
4	debug	100000	0	100000

Cleanup

1. Reset the Docker Slurm cluster environment back to its original, pristine state by exiting the login Docker container back to the Ubuntu AWS prompt. And **as ubuntu**:

```
cd ~/docker-scale-out
make clean && make
```

Advanced Resource Reservations Discussion

Slurm has the ability to reserve resources for jobs being executed by select users and/or select bank accounts. A resource reservation identifies the resources in that reservation and a time period during which the reservation is available. The resources which can be reserved include cores, nodes, licenses and/or burst buffers. Note that resource reservations are not compatible with Slurm's gang scheduler plugin since the termination time of running jobs cannot be accurately predicted.

Note that reserved burst buffers and licenses are treated somewhat differently than reserved cores or nodes. When cores or nodes are reserved, then jobs using that reservation can use only those resources and no other jobs can use those resources. Reserved burst buffers and licenses can only be used by jobs associated with that reservation, but licenses not explicitly reserved are available to any job. This eliminates the need to explicitly put licenses into every advanced reservation created.

Reservations can be created, updated, or destroyed only by user root or the configured SlurmUser using the scontrol command. The scontrol and sview commands can be used to view reservations. The man pages for the various commands contain details

Reservation Creation

One common mode of operation for a reservation would be to reserve an entire computer at a particular time for a system down time. The example below shows the creation of a full-system reservation at 16:00 hours on 6 February and lasting for 120 minutes. The "maint" flag is used to identify the reservation for accounting purposes as system maintenance. The "ignore_jobs" flag is used to indicate that we can ignore currently running jobs when creating this reservation. By default, only resources which are not expected to have a running job at the start time can be reserved (the time limit of all running jobs will have been reached). In this case we can manually cancel the running jobs as needed to perform system maintenance. As the reservation time approaches, only jobs that can complete by the reservation time will be initiated.

```
$ scontrol create reservation starttime=2019-02-06T16:00:00 \
    duration=120 user=root flags=maint,ignore_jobs nodes=ALL
Reservation created: root_3

$ scontrol show reservation
ReservationName=root_3 StartTime=2009-02-06T16:00:00
    EndTime=2009-02-06T18:00:00 Duration=120
    Nodes=ALL NodeCnt=20
    Features=(null) PartitionName=(null)
    Flags=MAINT,SPEC_NODES,IGNORE_JOBS Licenses=(null)
    BurstBuffers=(null)
    Users=root Accounts=(null)
```

A variation of this would be to configure license to represent system resources, such as a global file system. The system resource may not require an actual license for use, but Slurm licenses can be used to prevent jobs needed the resource from being started when that resource is unavailable. One could create a reservation for all of those licenses in order to perform maintenance on that resource. In the example below, we create a reservation for 1000 licenses with the name of "lustre". If there are a total of 1000 lustre licenses configured in this cluster, this reservation will prevent any job specifying the need for needed a lustre license from being scheduled on this cluster during this reservation.

```

$ scontrol create reservation starttime=2009-04-06T16:00:00 \
    duration=120 user=root flags=license_only \
    licenses=lustre:1000
Reservation created: root_4

$ scontrol show reservation
ReservationName=root_4 StartTime=2009-04-06T16:00:00
    EndTime=2009-04-06T18:00:00 Duration=120
    Nodes= NodeCnt=0
    Features=(null) PartitionName=(null)
    Flags=LICENSE_ONLY Licenses=lustre*1000
    BurstBuffers=(null)
    Users=root Accounts=(null)

```

Another mode of operation would be to reserve specific nodes for an indefinite period in order to study problems on those nodes. This could also be accomplished using a Slurm partition specifically for this purpose, but that would fail to capture the maintenance nature of their use.

```

$ scontrol create reservation user=root starttime=now \
    duration=infinite flags=maint nodes=sun000
Reservation created: root_5

$ scontrol show res
ReservationName=root_5 StartTime=2009-02-04T16:22:57
    EndTime=2009-02-04T16:21:57 Duration=4294967295
    Nodes=sun000 NodeCnt=1
    Features=(null) PartitionName=(null)
    Flags=MAINT,SPEC_NODES Licenses=(null)
    BurstBuffers=(null)
    Users=root Accounts=(null)

```

The next example is to reserve ten nodes in the default Slurm partition starting at noon and with a duration of 60 minutes occurring daily. The reservation will be available only to users "alan" and "brenda"

```

$ scontrol create reservation user=alan,brenda \
    starttime=noon duration=60 flags=daily nodecnt=10
Reservation created: alan_6

$ scontrol show res
ReservationName=alan_6 StartTime=2009-02-05T12:00:00
    EndTime=2009-02-05T13:00:00 Duration=60
    Nodes=sun[000-003,007,010-013,017] NodeCnt=10
    Features=(null) PartitionName=pdebug
    Flags=DAILY Licenses=(null) BurstBuffers=(null)
    Users=alan,brenda Accounts=(null)

```

The next example is to reserve 100GB of burst buffer space starting at noon today and with a duration of 60 minutes. The reservation will be available only to users "alan" and "brenda"

```

$ scontrol create reservation user=alan,brenda \
    starttime=noon duration=60 flags=daily nodecnt=10
Reservation created: alan_6

$ scontrol show res

```

```
ReservationName=alan_6 StartTime=2009-02-05T12:00:00
EndTime=2009-02-05T13:00:00 Duration=60
Nodes=sun[000-003,007,010-013,017] NodeCnt=10
Features=(null) PartitionName=pdebug
Flags=DAILY Licenses=(null) BurstBuffers=(null)
Users=alan,brenda Accounts=(null)
```

Reservations can be optimized with respect to system topology if the reservation request includes information about the sizes of jobs to be created. This is especially important for BlueGene systems due to restrictive rules about the topology of created blocks (due to hardware constraints and/or Slurm's configuration). To take advantage of this optimization, specify the sizes of jobs to be concurrently executed. The example below creates a reservation containing 4096 c-nodes on a BlueGene system so that two 2048 c-node jobs can execute simultaneously.

```
$ scontrol create reservation user=alan,brenda \
  starttime=noon duration=60 nodecnt=2k,2k
Reservation created: alan_8

$ scontrol show res
ReservationName=alan_8 StartTime=2011-12-05T12:00:00
EndTime=2011-12-05T13:00:00 Duration=60
Nodes=bgp[000x011,210x311] NodeCnt=4096
Features=(null) PartitionName=pdebug
Flags= Licenses=(null) BurstBuffers=(null)
Users=alan,brenda Accounts=(null)
```

Note that specific nodes to be associated with the reservation are identified immediately after creation of the reservation. This permits users to stage files to the nodes in preparation for use during the reservation. Note that the reservation creation request can also identify the partition from which to select the nodes or `_one_` feature that every selected node must contain.

On a smaller system, one might want to reserve cores rather than whole nodes. This capability permits the administrator to identify the core count to be reserved on each node as shown in the examples below.

NOTE: Core reservations are not available on Cray/ALPS systems, nor when the system is configured to use the select/linear or select/serial plugins.

```
# Create a two core reservation for user alan
$ scontrol create reservation StartTime=now Duration=60 \
  NodeCnt=1 CoreCnt=2 User=alan

# Create a reservation for user brenda with two cores on
# node tux8 and 4 cores on node tux9
$ scontrol create reservation StartTime=now Duration=60 \
  Nodes=tux8,tux9 CoreCnt=2,4 User=brend
```

Reservations can not only be created for the use of specific accounts and users, but specific accounts and/or users can be prevented from using them. In the following example, a reservation is created for account "foo", but user "alan" is prevented from using the reservation even when using the account "foo".

```
$ scontrol create reservation account=foo \
  user=-alan partition=pdebug \
  starttime=noon duration=60 nodecnt=2k,2k
Reservation created: alan_9

$ scontrol show res
ReservationName=alan_9 StartTime=2011-12-05T13:00:00
  EndTime=2011-12-05T14:00:00 Duration=60
  Nodes=bgp[000x011,210x311] NodeCnt=4096
  Features=(null) PartitionName=pdebug
  Flags= Licenses=(null) BurstBuffers=(null)
  Users=-alan Accounts=foo
```

Reservation Use

The reservation create response includes the reservation's name. This name is automatically generated by Slurm based upon the first user or account name and a numeric suffix. In order to use the reservation, the job submit request must explicitly specify that reservation name. The job must be contained completely within the named reservation. The job will be canceled after the reservation reaches its EndTime. If letting the job continue execution after the reservation EndTime, a configuration option ResvOverRun in slurm.conf can be set to control how long the job can continue execution

```
$ sbatch --reservation=alan_6 -N4 my.script
sbatch: Submitted batch job 65540
```

Note that use of a reservation does not alter a job's priority, but it does act as an enhancement to the job's priority. Any job with a reservation is considered for scheduling to resources before any other job in the same Slurm partition (queue) not associated with a reservation.

Reservation Modification

Reservations can be modified by user root as desired. For example their duration could be altered or the users granted access changed as shown below:

```
$ scontrol update ReservationName=root_3 duration=150 users=admin
Reservation updated.

$ scontrol show ReservationName=root_3
ReservationName=root_3 StartTime=2009-02-06T16:00:00
  EndTime=2009-02-06T18:30:00 Duration=150
  Nodes=ALL NodeCnt=20 Features=(null)
  PartitionName=(null) Flags=MAINT,SPEC_NODES
  Licenses=(null) BurstBuffers=(null)
  Users=admin Accounts=(null)
```

Reservation Deletion

Reservations are automatically purged after their end time. They may also be manually deleted as shown below. Note that a reservation can not be deleted while there are jobs running in it.

```
$ scontrol delete ReservationName=alan_6
```

NOTE: By default, when a reservation ends the reservation request will be removed from any pending jobs submitted to the reservation and will be put into a held state. Use the NO_HOLD_JOBS_AFTER_END reservation flag to let jobs run outside of the reservation after the reservation is gone.

Overlapping Reservations

By default, reservations must not overlap. They must either include different nodes or operate at different times. If specific nodes are not specified when a reservation is created, Slurm will automatically select nodes to avoid overlap and ensure that the selected nodes are available when the reservation begins.

There is very limited support for overlapping reservations with two specific modes of operation available. For ease of system maintenance, you can create a reservation with the "maint" flag that overlaps existing reservations. This permits an administrator to easily create a maintenance reservation for an entire cluster without needing to remove or reschedule pre-existing reservations. Users requesting access to one of these pre-existing reservations will be prevented from using resources that are also in this maintenance reservation. For example, users alan and brenda might have a reservation for some nodes daily from noon until 1PM. If there is a maintenance reservation for all nodes starting at 12:30PM, the only jobs they may start in their reservation would have to be completed by 12:30PM, when the maintenance reservation begins.

The second exception operates in the same manner as a maintenance reservation except that it is not logged in the accounting system as nodes reserved for maintenance. It requires the use of the "overlap" flag when creating the second reservation. This might be used to ensure availability of resources for a specific user within a group having a reservation. Using the previous example of alan and brenda having a 10 node reservation for 60 minutes, we might want to reserve 4 nodes of that for brenda during the first 30 minutes of the time period. In this case, the creation of one overlapping reservation (for a total of two reservations) may be simpler than creating three separate reservations, partly since the use of any reservation requires the job specification of the reservation name.

1. A six node reservation for both alan and brenda that lasts the full 60 minutes
2. A four node reservation for brenda for the first 30 minutes
3. A four node reservation for both alan and brenda that lasts for the final 30 minutes

If the "maint" or "overlap" flag is used when creating reservations, one could create a reservation within a reservation within a third reservation. Note a reservation having a "maint" or "overlap" flag will not have resources removed from it by a subsequent reservation also having a "maint" or "overlap" flag, so nesting of reservations only works to a depth of two.

Reservations Floating Through Time

Slurm can be used to create an advanced reservation with a start time that remains a fixed period of time in the future. These reservation are not intended to run jobs, but to prevent long running jobs from being initiated on specific nodes. That node might be placed in a DRAINING state to prevent any new jobs from being started there. Alternately, an advanced reservation might be placed on the node to prevent jobs exceeding some specific time limit from being started. Attempts by users to make use of a reservation with a floating start time will be rejected. When ready to perform the maintenance, place the node in DRAINING state and delete the previously created advanced reservation.

Create the reservation by using the flag value of TIME_FLOAT and a start time that is relative to the current time (use the keyword now). The reservation duration should generally be a value which is large relative to typical job

run times in order to not adversely impact backfill scheduling decisions. Alternately the reservation can have a specific end time, in which case the reservation's start time will increase through time until the reservation's end time is reached. When the current time passes the reservation end time then the reservation will be purged. In the example below, node tux8 is prevented from starting any jobs exceeding a 60 minute time limit. The duration of this reservation is 100 (minutes).

```
$ scontrol create reservation user=operator nodes=tux8 \
  starttime=now+60minutes duration=100 flags=time_float
```

Reservations that Replace Allocated Resources

Slurm can create an advanced reservation for which nodes which are allocated to jobs are automatically replaced with new idle nodes. The effect of this is to always maintain a constant size pool of resources. This is accomplished by using a "replace" flag as shown in the example below. This option is not supported for reservations of individual cores which span more than one node rather than full nodes (e.g. a 1 core reservation on node "tux1" will be moved if node "tux1" goes down, but a reservation containing 2 cores on node "tux1" and 3 cores on "tux2" will not be moved node "tux1" goes down).

```
$ scontrol create reservation starttime=now duration=60 \
  users=foo nodecnt=2 flags=replace
Reservation created: foo_82

$ scontrol show res
ReservationName=foo_82 StartTime=2014-11-20T16:21:11
  EndTime=2014-11-20T17:21:11 Duration=01:00:00
  Nodes=tux[0-1] NodeCnt=2 CoreCnt=12 Features=(null)
  PartitionName=debug Flags=REPLACE
  Users=jette Accounts=(null) Licenses=(null) State=ACTIVE

$ sbatch -n4 --reservation=foo_82 tmp
Submitted batch job 97

$ scontrol show res
ReservationName=foo_82 StartTime=2014-11-20T16:21:11
  EndTime=2014-11-20T17:21:11 Duration=01:00:00
  Nodes=tux[1-2] NodeCnt=2 CoreCnt=12 Features=(null)
  PartitionName=debug Flags=REPLACE
  Users=jette Accounts=(null) Licenses=(null) State=ACTIVE

$ sbatch -n4 --reservation=foo_82 tmp
Submitted batch job 98

$ scontrol show res
ReservationName=foo_82 StartTime=2014-11-20T16:21:11
  EndTime=2014-11-20T17:21:11 Duration=01:00:00
  Nodes=tux[2-3] NodeCnt=2 CoreCnt=12 Features=(null)
  PartitionName=debug Flags=REPLACE
  Users=jette Accounts=(null) Licenses=(null) State=ACTIVE

$ squeue
JOBID PARTITION  NAME   USER ST    TIME   NODES NODELIST(REASON)
  97      debug     tmp   foo  R  0:09        1  tux0
  98      debug     tmp   foo  R  0:07        1  tux1
```

Reservation Purging After Last Job

A reservation may be automatically purged after the last associated job completes. This is accomplished by using a "purge_comp" flag. Once the reservation has been created, it must be populated within 5 minutes of its start time or it will be purged before any jobs have been run.

Reservation Accounting

Jobs executed within a reservation are accounted for using the appropriate user and bank account. If resources within a reservation are not used, those resources will be accounted for as being used by all users or bank accounts associated with the reservation on an equal basis (e.g. if two users are eligible to use a reservation and neither does, each user will be reported to have used half of the reserved resources).

Prolog and Epilog

Slurm supports both a reservation prolog and epilog. They may be configured using the ResvProlog and ResvEpilog configuration parameters in the slurm.conf file. These scripts can be used to cancel jobs, modify partition configuration, etc.

Reservations Lab Exercises

In this set of labs, you will learn how to configure Slurm for reservations.

Exercise 1: Create an administrative reservation across the entire cluster

1. As root, create an administrative (for maintenance) reservation on the entire system that starts now and lasts for 120 minutes (the following is a single command that wraps in this document):

```
scontrol create reservation partition=debug starttime=now duration=120  
user=root flags=maint nodes=ALL
```

You should see:

```
Reservation created: root_1
```

2. View the reservation:

```
scontrol show reservations
```

You should see:

```
ReservationName=root_1 StartTime=2020-04-02T16:43:43 EndTime=2020-04-02T18:43:43  
Duration=02:00:00  
Nodes=node[00-09] NodeCnt=10 CoreCnt=40 Features=(null) PartitionName=debug  
Flags=MAINT,SPEC_NODES,ALL_NODES  
TRES=cpu=40  
Users=root Accounts=(null) Licenses=(null) State=ACTIVE BurstBuffer=(null) Watts=n/a  
MaxStartDelay=(null)
```

3. Or, alternatively, use the sinfo command to see it:

```
sinfo -T -l
```

You should see:

Thu Apr 02 16:44:45 2020	RESV_NAME	STATE	START_TIME	END_TIME	DURATION	NODELIST
	root_1	ACTIVE	2020-04-02T16:43:43	2020-04-02T18:43:43	02:00:00	node[00-09]

4. As the fred user submit 5, 5-minute jobs:

```
srun sleep 300 &  
srun sleep 300 &  
srun sleep 300 &  
srun sleep 300 &  
srun sleep 300 &
```

5. View the jobs, notice that the resources are not currently available: (ReqNodeNotAvail is the value that defines the availability of the resources)

```
squeue
```

You should see:

JOBID	PARTITION	NAME	USER	ST	TIME	NODES	NODELIST (REASON)
1	debug	sleep	fred	PD	0:00	1	(ReqNodeNotAvail, Reserved for maintenance)
2	debug	sleep	fred	PD	0:00	1	(ReqNodeNotAvail, Reserved for maintenance)
3	debug	sleep	fred	PD	0:00	1	(ReqNodeNotAvail, Reserved for maintenance)
4	debug	sleep	fred	PD	0:00	1	(ReqNodeNotAvail, Reserved for maintenance)
5	debug	sleep	fred	PD	0:00	1	(ReqNodeNotAvail, Reserved for maintenance)

6. As root, modify the reservation to free up node00 to run the queued up jobs:

```
scontrol update ReservationName=root_1 Nodes=node[01-09]
```

7. View the updated reservation:

```
scontrol show reservations
```

You should see:

```
ReservationName=root_1 StartTime=2020-04-02T16:43:43 EndTime=2020-04-02T18:43:43 Duration=02:00:00
Nodes=node[01-09] NodeCnt=9 CoreCnt=36 Features=(null) PartitionName=debug Flags=MAINT,SPEC_NODES
TRES=cpu=36
Users=root Accounts=(null) Licenses=(null) State=ACTIVE BurstBuffer=(null) Watts=n/a
MaxStartDelay=(null)
```

Notice the Nodes=node[01-09] now.

8. Or, use the sinfo command to see it:

```
sinfo -T -l
```

You should see:

Wed Nov 27 22:51:21 2019	RESV_NAME	STATE	START_TIME	END_TIME	DURATION	NODELIST
	root_1	ACTIVE	2020-04-02T16:43:43	2020-04-02T18:43:43	02:00:00	node[01-09]

9. View the jobs in the queue now:

```
squeue
```

You should see:

JOBID	PARTITION	NAME	USER	ST	TIME	NODES	NODELIST (REASON)
2	debug	sleep	fred	PD	0:00	1	(Resources)
3	debug	sleep	fred	PD	0:00	1	(ReqNodeNotAvail, Reserved for maintenance)
4	debug	sleep	fred	PD	0:00	1	(ReqNodeNotAvail, Reserved for maintenance)
5	debug	sleep	fred	PD	0:00	1	(ReqNodeNotAvail, Reserved for maintenance)
1	debug	sleep	fred	R	0:05	1	node00

Note that the jobs are running and scheduled to run on node00 since it now free from reservations

10. As root user, cancel the jobs and remove the reservation:

```
scancel -u fred
scontrol delete ReservationName=root_1
```

Exercise 2: Create a reservation on a specific node for a specific user

In this exercise, you create a reservation on a specific node for a specific user, much like a principal investor (PI) would want to take advantage of a node he has purchased for research

1. As the root user, create a reservation on node09:

```
scontrol create reservation starttime=now+1hour duration=120 user=fred nodes=node09
```

You should see:

```
Reservation created: fred_2
```

2. Show the reservation:

```
scontrol show reservations
```

You should see:

```
ReservationName=fred_2 StartTime=2020-04-02T18:20:04 EndTime=2020-04-02T20:20:04 Duration=02:00:00  
Nodes=node09 NodeCnt=1 CoreCnt=4 Features=(null) PartitionName=debug Flags=SPEC_NODES  
TRES(cpu=4  
Users=fred Accounts=(null) Licenses=(null) State=INACTIVE BurstBuffer=(null) Watts=n/a  
MaxStartDelay=(null)
```

3. Or, use the sinfo command to see it:

```
sinfo -T -l
```

You should see:

Thu Apr 02 17:21:06 2020	RESV_NAME	STATE	START_TIME	END_TIME	DURATION	NODELIST
	fred_2	INACTIVE	2020-04-02T18:20:04	2020-04-02T20:20:04	02:00:00	node09

NOTE: The State is INACTIVE because the reservation hasn't started yet

4. As fred, submit a job to that reservation:

```
su - fred  
srun --reservation=fred_2 sleep 300 &
```

5. Show the queued job:

```
squeue
```

You should see:

JOBID	PARTITION	NAME	USER	ST	TIME	NODES	NODELIST (REASON)
6	debug	sleep	fred	PD	0:00	1	(Reservation)

Cleanup

1. Reset the Docker Slurm cluster environment back to its original, pristine state by exiting the login Docker container back to the Ubuntu AWS prompt. And **as ubuntu**:

```
cd ~/docker-scale-out  
make clean && make
```

Managing Access to Compute Nodes Discussion

The most common configuration in HPC with regard to user connectivity between controller nodes and the compute resources takes advantage of the SSH protocol. Once the SSH keys have been generated and distributed among the cluster, it makes it very simple for Slurm to push the job scripts to the appropriate resources for execution.

pam_slurm_adopt Module

The purpose of this module is to prevent users from sshing into nodes that they do not have a running job on, and to track the ssh connection and any other spawned processes for accounting and to ensure complete job cleanup when the job is completed. This module does this by determining the job which originated the ssh connection. The user's connection is "adopted" into the "external" step of the job.

Installation from Source

In your Slurm build directory, navigate to `slurm/contribs/pam_slurm_adopt/` and run the following as root:

```
make && make install
```

This will place `pam_slurm_adopt.a`, `pam_slurm_adopt.la`, and `pam_slurm_adopt.so` in `/lib/security/` (on Debian systems) or `/lib64/security/` (on RedHat/SuSE systems)

Installation by RPM

The included `slurm.spec` will build a `slurm-pam_slurm` RPM which will install `pam_slurm_adopt`. Refer to the Quick Start Administrator Guide for instructions on managing an RPM-based install.

PAM Configuration

Add the following line to the appropriate file in `/etc/pam.d`, such as `system-auth` or `sshd` (you may use either the "required" or "sufficient" PAM control flag):

```
account required pam_slurm_adopt.so
```

The order of plugins is very important. `pam_slurm_adopt.so` should be the last PAM module in the account stack. Included files such as `common-account` should normally be included before `pam_slurm_adopt`. You might have the following account stack in `sshd`:

```
account required pam_nologin.so
account include password-auth
...
account required pam_slurm_adopt.so
```

`pam_slurm_adopt` must be used with the task/cgroup plugin. The `pam_systemd` module will conflict with `pam_slurm_adopt`, so you need to disable it in all files that are included in `sshd` or `system-auth` (e.g. `password-auth`, `common-session`, etc.). You should also stop and mask `systemd-logind`. You must also make sure a different PAM module isn't short-circuiting the account stack before it gets to `pam_slurm_adopt.so`. From the example above, the following two lines have been commented out in the included `password-auth` file:

```
#account sufficient pam_localuser.so  
#-session optional pam_systemd.so
```

Note: This may involve editing a file that is auto-generated. Do not run the config script that generates the file or your changes will be erased.

If you always want to allow access for an administrative group (e.g., wheel), stack the pam_access module after pam_slurm_adopt. A success with pam_slurm_adopt is sufficient to allow access, but the pam_access module can allow others, such as administrative staff, access even without jobs on that node:

```
account sufficient pam_slurm_adopt.so  
account required pam_access.so
```

Then edit the pam_access configuration file (/etc/security/access.conf):

```
+:wheel:ALL  
-:ALL:ALL
```

An alternative to pam_access is to place pam_listfile.so before pam_slurm_adopt.so. For example:

```
account sufficient pam_listfile.so item=user sense=allow onerr=fail  
file=/path/to/allowed_users_file  
account required pam_slurm_adopt.so
```

List the usernames of the allowed users in allowed_users_file.

When access is denied, the user will receive a relevant error message

pam_slurm_adopt Module Options

This module is configurable. Add these options to the end of the pam_slurm_adopt line in the appropriate file in /etc/pam.d/ (e.g., sshd or system-auth):

```
account sufficient pam_slurm_adopt.so optionname=optionvalue
```

This module has the following options:

- **action_no_jobs:** The action to perform if the user has no jobs on the node. Configurable values are:
 - ignore: Do nothing. Fall through to the next pam module.
 - deny (default): Deny the connection.
- **action_unknown:** The action to perform when the user has multiple jobs on the node and the RPC does not locate the source job. If the RPC mechanism works properly in your environment, this option will likely be relevant only when connecting from a login node. Configurable values are:
 - newest (default): Pick the newest job on the node. The "newest" job is chosen based on the mtime of the job's step_extern cgroup; asking Slurm would require an RPC to the controller. Thus, the memory cgroup must be in use so that the code can check mtimes of cgroup

- directories. The user can ssh in but may be adopted into a job that exits earlier than the job they intended to check on. The ssh connection will at least be subject to appropriate limits and the user can be informed of better ways to accomplish their objectives if this becomes a problem.
- allow: Let the connection through without adoption.
 - deny: Deny the connection.
- **action_adopt_failure:** The action to perform if the process is unable to be adopted into any job for whatever reason. If the process cannot be adopted into the job identified by the callerid RPC, it will fall through to the action_unknown code and try to adopt there. A failure at that point or if there is only one job will result in this action being taken. Configurable values are:
 - allow (default): Let the connection through without adoption. WARNING: This value is insecure and is recommended for testing purposes only. We recommend using "deny."
 - deny: Deny the connection.
 - **action_generic_failure:** The action to perform if there are certain failures such as the inability to talk to the local slurmd or if the kernel doesn't offer the correct facilities. Configurable values are:
 - ignore (default): Do nothing. Fall through to the next pam module. WARNING: This value is insecure and is recommended for testing purposes only. We recommend using "deny."
 - allow: Let the connection through without adoption.
 - deny: Deny the connection.
 - **disable_x11:** Turn off Slurm built-in X11 forwarding support. Configurable values are:
 - 0 (default): If the job the connection is adopted into has Slurm's X11 forwarding enabled, the DISPLAY variable will be overwritten with the X11 tunnel endpoint details.
 - 1: Do not check for Slurm's X11 forwarding support, and do not alter the DISPLAY variable.
 - **log_level:** See SlurmdDebug in slurm.conf for available options. The default log_level is info.
 - **nodename:** If the NodeName defined in slurm.conf is different than this node's hostname (as reported by hostname -s), then this must be set to the NodeName in slurm.conf that this host operates as.

Slurm Configuration

PrologFlags=contain must be set in the slurm.conf. This sets up the "extern" step into which ssh-launched processes will be adopted. You must also enable the task/cgroup plugin in slurm.conf. See the Slurm cgroups guide.

NOTE: If you are doing X11 forwarding, PrologFlags=x11 implies PrologFlags=contain and you don't have to define BOTH x11 and contain

IMPORTANT

PrologFlags=contain must be in place before using this module. The module bases its checks on local steps that have already been launched. If the user has no steps on the node, such as the extern step, the module will assume that the user has no jobs allocated to the node. Depending on your configuration of the PAM module, you might accidentally deny all user ssh attempts without PrologFlags=contain.

The UsePAM option in slurm.conf is not related to pam_slurm_adopt.

NSS_Slurm

nss_slurm is an optional NSS plugin that can permit passwd and group resolution for a job on the compute node to be serviced through the local slurmstepd process, rather than through some alternate network-based service such as LDAP, SSSD, or NSLCD.

When enabled on the cluster, for each job, the job's user will have their full struct passwd info — username, uid, primary gid, gecos info, home directory, and shell — securely sent as part of each step launch, and cached within the slurmstepd process. This info will then be provided to any process launched by that step through the getpwuid()/getpwnam()/getpwent() system calls.

For group information — from the getgrgid()/getgrnam()/getgrent() system calls —, an abbreviated view of struct group will be provided. Within a given process, the response will include only those groups that the user belongs to, but with only the user themselves listed as a member. The full list of group members is not provided.

Exercise 1: pam_slurm_adopt

In this set of labs, you will configure the pam_slurm_adopt module to restrict users from ssh'ing to a compute node when they do not have jobs running on them.

Configure PAM:

1. In the Login container as root user, edit `/etc/pam.d/system-auth` and add the following to the end:

```
account required pam_slurm_adopt.so action_no_jobs=deny
```

Note: Remove any other entries to pam_slurm_adopt.so

2. Edit `/etc/pam.d/sshd`. Change (or add if not present):

```
account sufficient pam_slurm_adopt.so
```

To:

```
account required pam_slurm_adopt.so action_no_jobs=deny
```

3. Distribute the PAM config files to the compute nodes:

```
pdcp /etc/pam.d/system-auth /etc/pam.d  
pdcp /etc/pam.d/sshd /etc/pam.d
```

Configure slurm.conf:

In order for pam_slurm_adopt.so to be in effect, you must have PrologFlags=contain in the slurm.conf. However, the PrologFlags=x11 implies "contain" so if you have x11 you don't need "contain".

Watch it

1. ssh to a compute node as the fred user:

```
ssh node00
```

You should see the PAM denial:

```
[fred@login ~]$ ssh node00  
Access denied by pam_slurm_adopt: you have no active jobs on this node  
  
Connection closed by 2001:db8:1:1::5:0 port 22  
[fred@login ~]$
```

2. Submit a job as fred:

```
sbatch -wnode00 -t 1000 --wrap="sleep 1000"
```

3. With the job running on node00, ssh to the node:

```
ssh node00
```

It should work

4. Exit the job:

```
exit
```

5. Cancel the job (if you did this example as **fred**):

```
scancel -u fred
```

Interactive session with cgroup enforcement

With Cgroups enabled, it is possible to now demonstrate a pam_slurm_adopt interactive session and the enforcement of the allocated resources. In this exercise, you will start a pam_slurm_adopt controlled interactive session and see how attempting to use more cores than allocated will result in failure.

1. As **betty**, start a single node, 2-core interactive session with srun:

```
salloc -n2 -N1 --mem=1000
```

2. Copy the Monte Carlo simulation MPI pi.c program out of **/lab_scripts** to betty's home directory:

```
cp /lab_scripts/pi.c ~/
```

3. Compile the MPI pi program:

```
/usr/local/openmpi/4.1.4/bin/mpicc pi.c
```

Note: This will generate a file called a.out

4. Run the "Calculate pi" program across 2 cores:

```
/usr/local/openmpi/4.1.4/bin/mpirun --np 2 a.out
```

Note: In a moment you should see:

```
[betty@node00 ~]$ /usr/local/openmpi/4.1.4//bin/mpirun --np 2 a.out
np= 2;      Time=14.752216s;      PI=3.141685960
[betty@node00 ~]$
```

5. Now try running the "Calculate pi" program across 4 cores:

```
/usr/local/openmpi/4.1.4/bin/mpirun --np 4 a.out
```

Note: Almost immediately, you should see:

```
-----  
There are not enough slots available in the system to satisfy the 4  
slots that were requested by the application:
```

```
a.out
```

```
Either request fewer slots for your application, or make more slots  
available for use.
```

```
A "slot" is the Open MPI term for an allocatable unit where we can  
launch a process. The number of slots available are defined by the  
environment in which Open MPI processes are run:
```

1. Hostfile, via "slots=N" clauses (N defaults to number of processor cores if not provided)
2. The --host command line parameter, via a ":N" suffix on the hostname (N defaults to 1 if not provided)
3. Resource manager (e.g., SLURM, PBS/Torque, LSF, etc.)
4. If none of a hostfile, the --host command line parameter, or an RM is present, Open MPI defaults to the number of processor cores

```
In all the above cases, if you want Open MPI to default to the number of hardware threads instead of the number of processor cores, use the --use-hwthread-cpus option.
```

```
Alternatively, you can use the --oversubscribe option to ignore the number of available slots when deciding the number of processes to launch.
```

```
-----  
[betty@node00 ~]
```

6. So cgroup restricted the --np 4 request because the allocation was only granted 2 cores.

Cleanup

1. Reset the Docker Slurm cluster environment back to its original, pristine state by exiting the login Docker container back to the Ubuntu AWS prompt. And as ubuntu:

```
cd ~/docker-scale-out  
make clean && make
```

Exercise 2: Configure nss_slurm

nss_slurm provides the capability of not having to have users exist on compute nodes. Rather, jobs are proxied via the user nobody.

Note: do this exercise on the mgmtnode container.

1. Compile the required binaries and libraries. In the **mgmtnode** container, **as the root user**, change directory to the Slurm source code directory:

```
ssh mgmtnode
```

```
cd /usr/local/src/slurm/contribs/nss_slurm
```

2. Compile the code:

```
make && make install
```

3. Edit the `/etc/slurm/slurm.conf` and confirm the following setting:

```
LaunchParameters=enable_nss_slurm
```

Comment out the following:

```
#AccountingStorageEnforce=limits,qos,safe
```

4. Restart the scheduler:

```
/lab_scripts/restart.sh
```

5. Add your user account to the **mgmtnode** container:

```
useradd -m shawn
```

6. On the mgmtnode container, change to your user account and submit a job:

```
su - shawn
```

```
sbatch --wrap "sleep 100"
```

NOTE: The job should run fine, even though the user doesn't exist on the compute nodes. Even if you include filesystem components to your job scripts, the files will be owned by the UID of the user you just added on the mgmtnode.

7. In addition, the job runs as user "nobody" on the mgmtnode, as you can see in squeue ON NODE00:

```
squeue
```

Wed Sep 29 12:18:54 2021								
JOBID	PARTITION	NAME	USER	STATE	TIME	TIME_LIMI	NODES	NODELIST (REASON)
2	debug	wrap	nobody	RUNNING	0:58	1:00	1	node00

Cleanup

1. Reset the Docker Slurm cluster environment back to its original, pristine state by exiting the login Docker container back to the Ubuntu AWS prompt. And as ubuntu:

```
cd ~/docker-scale-out  
make clean && make
```

sctrontab Discussion

`scrontab` is used to set, edit, and remove a user's Slurm-managed crontab. This file can define a number of recurring batch jobs to run on a scheduled interval.

Lines must be either comments starting with '#', whitespace, or valid crontab entries.

Lines starting with #SCRON allow options to be defined for the single following crontab entry. Options are always reset in between each crontab entry. Options include most of those available to the `sbatch` command; details are available in the `sbatch(1)` man page.

Note that jobs are not guaranteed to execute at the preferred time. Jobs will run no more frequently than requested, but are managed by setting the BeginTime field to the next valid iteration, and are then subject to queuing delays. The exact behavior will depend on the local site configuration. Because of this method of implementation, the next job in the series won't be submitted until after the previous job has completed. For example, if you have a monitoring job that is scheduled to run every minute on a busy system, if the job scheduled to start at 08:20:00 isn't able to start until 08:20:30 and it runs for 35 seconds then the job at 08:21:00 will be skipped and the next job will be scheduled for 08:22:00.

`scrontab` uses the same syntax for date and time specifiers as cron. Each line has five fields that have the following meanings:

field	allowed values
---	-----
minute	0-59
hour	0-23
day of month	1-31
month	1-12 (or name)
day of week	0-7 (0 and 7 are Sunday, or use name)

A field can contain an asterisk (*) which means that it's valid for each of the allowed values for the given time period. Ranges are allowed where a range is two numbers with a hyphen between them. The second number must be greater than the first. Lists are allowed, with commas separating the numbers or ranges being separated. Step values can be specified by entering a slash (/), followed by the step value, causing the job to run at the specified interval appropriate for that field.

Options

- e Edit the crontab. If a crontab does not exist already, a default example (without any defined entries) will be provided in the editor.
- l List the crontab. (Prints directly to stdout.)
- r Remove the crontab. Any currently running crontab-defined jobs will continue to run but will no longer recur. All other crontab-defined jobs will be cancelled.
- u <user> Edit or view a different user's crontab. Listing is permitted for Operators and Admins. Editing/removal is only permitted for root and the SlurmUser account.

Sctrontab Options

scrontab allows you to use shortcuts to specify some common time intervals for the specified script to run.

These include the following:

- **@yearly | @annually** Job will become eligible at 00:00 Jan 01 each year.
- **@monthly** Job will become eligible at 00:00 on the first day of each month.
- **@weekly** Job will become eligible at 00:00 Sunday of each week.
- **@daily | @midnight** Job will become eligible at 00:00 each day.
- **@hourly** Job will become eligible at the first minute of each hour.

Scrontab Examples

scrontab is only available if the ScronParameters=enable option has been enabled in the `slurm.conf`.

Examples

To create a job that would run at the beginning of each hour, using the 'high' partition, 'sub1' account and have a walltime of 1 minute, you would add the following to `scrontab`:

```
#SCRON -p high
#SCRON -A sub1
#SCRON -t 1:00
@hourly /home/fred/date.printer.job
```

To have a job run every Wednesday, every other hour during the work day, each of the first five minutes of the hour and again at the thirty minute mark, you would add the following to `scrontab`.

```
1-5,30 8-17/2 * * wed /home/fred/example.job
```

Exercise 1: scrontab

In this set of labs, you will configure Slurm to run recurring jobs through `scrontab`, roughly equivalent to running other applications through crontab.

Configure scrontab:

1. As root, edit `/etc/slurm/slurm.conf` and add the following to the end:

```
ScronParameters=enable
```

2. Restart the scheduler:

```
/lab_scripts/restart.sh
```

Configure a batch job for fred:

Either an admin can create an `scrontab` entry for a user, or a user can create one for themselves.

1. As fred, create a job script to run through scrontab. Create the file called `myjob.sh` in fred's home directory, and add the following to it:

```
#!/bin/bash
date
srun ping -c 1 -4 google.com
echo $?
```

2. Make the script executable:

```
chmod +x ~/myjob.sh
```

Configure an scrontab entry for a fred:

Either an admin can create an `scrontab` entry for a user, or a user can create one for themselves.

1. As fred, create a crontab entry for user fred:

```
scrontab
```

2. Enter the following at the end of the file:

```
#SCRON -p debug
#SCRON -A bedrock
#SCRON --mem=1000
#SCRON -t 1
#SCRON -n 1
#SCRON -e /home/fred/myjob_%j.err
#SCRON -o /home/fred/myjob_%j.out
#SCRON --open-mode=append

* * * * * /home/fred/myjob.sh
```

3. Look at the queue:

```
squeue -la
```

It should show:

```
Wed Sep 29 12:34:51 2021
JOBID PARTITION      NAME      USER      STATE      TIME TIME_LIMI  NODES NODELIST(REASON)
      1    debug /home/fr      fred      PENDING      0:00      1:00      1 (BeginTime)
```

Note: The job will track with the crontab event. So as long as the controller is not restarted, the jobID will remain the same. However, the job steps will increment every minute, each time the job is started.

4. Check the sacct output to verify the success (or failure) of the job:

```
sacct -u fred
```

Should show:

```
...
1.0          ping            bedrock        4  COMPLETED      0:0
```

5. As fred, check the job output:

```
cat /home/fred/myjob_1.out
```

Should show:

```
[fred@login ~]$ cat myjob_1.out
Thu Oct 27 11:36:08 PDT 2022
PING google.com (142.250.189.14) 56(84) bytes of data.
64 bytes from lax31s16-in-f14.1e100.net (142.250.189.14): icmp_seq=1 ttl=127 time=14.8 ms

--- google.com ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 14.770/14.770/14.770/0.000 ms
0
.
.
.
```

Cleanup

1. Reset the Docker Slurm cluster environment back to its original, pristine state by exiting the login Docker container back to the Ubuntu AWS prompt. And as ubuntu:

```
cd ~/docker-scale-out
make clean && make
```

Job Submit Plugins Discussion

Job submit plugins are called from the slurmctld daemon when a job is submitted (or an allocation request is made). Each job submit and job modify call must pass through the job submit plugin. The job submit plugins are "Stackable." Meaning multiple job submit plugins can be used to provide complementary functions. All of the job submit or modify arguments are passed to the functions.

Some examples of current job submit plugins are:

- **throttle** - Limits rate at which users can submit jobs
(SchedulerParameters=jobs_per_user_per_hour=#)
- **require_timelimit** - Reject jobs without an explicit time limit
- **pbs** - Sets PBS environment variables and supports "before" job dependency
- **cray** - Sets Cray-specific generic resources
- **lua** - Executes LUA script

The job submit plugins are normally implemented as shared libraries (.so) implementing two required functions: job_submit and job_modify and two optional: init and fini.

In Slurm V23.02, a change has been made to the controller. The slurmctld will fatal() when reconfiguring the job_submit plugin fails. The best way to catch this is by either looking at the log file, or starting the controller in foreground mode: slurmctld -Dvvvv

Prolog/Epilog Discussion

Slurm supports a multitude of prolog and epilog programs. Note that for security reasons, these programs do not have a search path set. Either specify fully qualified path names in the program or set the "PATH" environment variable. The first table below identifies what prologs and epilogs are available for job allocations, when and where they run:

Parameter	Location	Invoked by	User	When executed
Prolog (from slurm.conf)	Compute or front end node	slurmd daemon	SlurmdUser (normally root)	First job or job step initiation on that node (by default); PrologFlags=Alloc will force the script to be executed at job allocation
PrologSlurmctld (from slurm.conf)	Head node (where slurmctld darmon runs)	slurmctld daemon	SlurmctldUser	At job allocation

Epilog (from slurm.conf)	Compute or front end node	slurmd daemon	SlurmdUser (normal user root)	At job termination
EpilogSlurmctld (from slurm.conf)	Head node (where slurmctld daemon runs)	slurmctld daemon	SlurmctldUser	At job termination

This second table below identifies what prologs and epilogs are available for job step allocations, when and where they run

Parameter	Location	Invoked by	User	When executed
SrunProlog (from slurm.conf) or srun --prolog	srun invocation node	srun command	User invoking srun command	Prior to launching job step
TaskProlog (from slurm.conf)	compute node	slurmstepd daemon	User invoking srun command	Prior to launching job step
srun --task-prolog	computer node	slurmstepd daemon	User invoking srun command	Prior to launching job step
TaskEpilog (from slurm.conf)	computer node	slurmstepd daemon	User invoking srun command	Completion job step
srun --task-epilog	computer node	slurmstepd daemon	User invoking srun command	Completion job step
SrunEpilog (from slurm.conf) or srun --epilog	srun invocation node	srun command	User invoking srun command	Completion job step

By default the Prolog script is only run on any individual node when it first sees a job step from a new allocation; it does not run the Prolog immediately when an allocation is granted. If no job steps from an allocation are run on a node, it will never run the Prolog for that allocation. This Prolog behaviour can be changed by the PrologFlags parameter. The Epilog, on the other hand, always runs on every node of an allocation when the allocation is released.

The task prolog is executed with the same environment as the user tasks to be initiated. The standard output of that program is read and processed as follows:

export name=value sets an environment variable for the user task

unset name clears an environment variable from the user task

print ... writes to the task's standard output.

Unless otherwise specified, these environment variables are available to all of the programs.

- **BASIL_RESERVATION_ID** Basil reservation ID. Available on Cray systems with ALPS only.
- **SLURM_ARRAY_JOB_ID** If this job is part of a job array, this will be set to the job ID. Otherwise it will not be set. To reference this specific task of a job array, combine SLURM_ARRAY_JOB_ID with SLURM_ARRAY_TASK_ID (e.g. "scontrol update \${SLURM_ARRAY_JOB_ID}_\${SLURM_ARRAY_TASK_ID} ..."); Available in PrologSlurmctld and EpilogSlurmctld only.
- **SLURM_ARRAY_TASK_ID** If this job is part of a job array, this will be set to the task ID. Otherwise it will not be set. To reference this specific task of a job array, combine SLURM_ARRAY_JOB_ID with SLURM_ARRAY_TASK_ID (e.g. "scontrol update \${SLURM_ARRAY_JOB_ID}_\${SLURM_ARRAY_TASK_ID} ..."); Available in PrologSlurmctld and EpilogSlurmctld only.
- **SLURM_ARRAY_TASK_MAX** If this job is part of a job array, this will be set to the maximum task ID. Otherwise it will not be set. Available in PrologSlurmctld and EpilogSlurmctld only.
- **SLURM_ARRAY_TASK_MIN** If this job is part of a job array, this will be set to the minimum task ID. Otherwise it will not be set. Available in PrologSlurmctld and EpilogSlurmctld only.
- **SLURM_ARRAY_TASK_STEP** If this job is part of a job array, this will be set to the step size of task IDs. Otherwise it will not be set. Available in PrologSlurmctld and EpilogSlurmctld only.
- **SLURM_CLUSTER_NAME** Name of the cluster executing the job.
- **SLURM_JOB_GPUS** GPU IDs allocated to the job (if any). Available in the Prolog only.
- **SLURM_JOB_ACCOUNT** Account name used for the job. Available in PrologSlurmctld and EpilogSlurmctld only.
- **SLURM_JOB_CONSTRAINTS** Features required to run the job. Available in Prolog, PrologSlurmctld and EpilogSlurmctld only.
- **SLURM_JOB_DERIVED_EC** The highest exit code of all of the job steps. Available in EpilogSlurmctld only.
- **SLURM_JOB_EXIT_CODE** The exit code of the job script (or `salloc`). The value is the status as returned by the `wait()` system call (See `wait(2)`) Available in EpilogSlurmctld only.
- **SLURM_JOB_EXIT_CODE2** The exit code of the job script (or `salloc`). The value has the format `..`. The first number is the exit code, typically as set by the `exit()` function. The second number of the signal that caused the process to terminate if it was terminated by a signal. Available in EpilogSlurmctld only.
- **SLURM_JOB_GID** Group ID of the job's owner. Available in PrologSlurmctld and EpilogSlurmctld only.
- **SLURM_JOB_GROUP** Group name of the job's owner. Available in PrologSlurmctld and EpilogSlurmctld only.
- **SLURM_JOB_ID** Job ID. CAUTION: If this job is the first task of a job array, then Slurm commands using this job ID will refer to the entire job array rather than this specific task of the job array.
- **SLURM_JOB_NAME** Name of the job. Available in PrologSlurmctld and EpilogSlurmctld only.

- **SLURM_JOB_NODELIST** Nodes assigned to job. A Slurm hostlist expression. "scontrol show hostnames" can be used to convert this to a list of individual host names. Available in PrologSlurmctld and EpilogSlurmctld only.
- **SLURM_JOB_PARTITION** Partition that job runs in. Available in Prolog, PrologSlurmctld and EpilogSlurmctld only.
- **SLURM_JOB_UID** User ID of the job's owner.
- **SLURM_JOB_USER** User name of the job's owner.
- **SLURM_WCKEY** User name of the job's wckey (if any). Available in PrologSlurmctld and EpilogSlurmctld only.

Plugins functions may also be useful to execute logic at various well defined points.

SPANK is another mechanism that may be useful to invoke logic in the user commands, slurmd daemon, and slurmstepd daemon.

FAILURE HANDLING

If the Epilog fails (returns a non-zero exit code), this will result in the node being set to a DOWN state. If the EpilogSlurmctld fails (returns a non-zero exit code), this will only be logged. If the Prolog fails (returns a non-zero exit code), this will result in the node being set to a DRAIN state and the job requeued in a held state (unless nohold_on_prolog_fail is configured in SchedulerParameters). If the PrologSlurmctld fails (returns a non-zero exit code), this will result in the job requeued to executed on another node if possible. Only batch jobs can be requeued. Interactive jobs (`salloc` and `srun`) will be cancelled if the PrologSlurmctld fails.

SPANK

SPANK provides a very generic interface for stackable plug-ins which may be used to dynamically modify the job launch code in Slurm. SPANK plugins may be built without access to Slurm source code. They need only be compiled against Slurm's spank.h header file, added to the SPANK config file plugstack.conf, and they will be loaded at runtime during the next job launch. Thus, the SPANK infrastructure provides administrators and other developers a low cost, low effort ability to dynamically modify the run-time behavior of Slurm job launch.

SPANK Plugins

SPANK plugins are loaded in up to five separate contexts during a Slurm job. Briefly, the five contexts are:

- **local**. In local context, the plugin is loaded by `srun`. (i.e. the "local" part of a parallel job).
- **remote**. In remote context, the plugin is loaded by `slurmstepd`. (i.e. the "remote" part of a parallel job).
- **allocator**. In allocator context, the plugin is loaded in one of the job allocation utilities `sbatch` or `salloc`
- **slurmd**. In `slurmd` context, the plugin is loaded in the `slurmd` daemon itself. Note: Plugins loaded in `slurmd` context persist for the entire time `slurmd` is running, so if configuration is changed or plugins are updated, `slurmd` must be restarted for the changes to take effect.
- **Job_script**. In the `job_script` context, plugins are loaded in the context of the job prolog or epilog. Note: Plugins are loaded in `job_script` context on each run on the job prolog or epilog, in a separate address space from plugins in `slurmd` context. This means there is no state shared between this context and other contexts, or even between one call to `slurm_spank_job_prolog` or `slurm_spank_job_epilog` and subsequent calls.

job_submit_lua plugin Discussion

Lua is scripting language implemented as a C library, which makes it perfect choice for small plugins to bigger C applications. It's both efficient (since it's really a C library, lua file can be read once and functions are normally executed multiple times) and easy to modify.

In Slurm, there is usually a need for customization that will fulfill organization specific requirements – like some preliminary checks done on job submission. In this case Slurm framework offers so called job_submit plugins, which are normally a shared libraries (.so) implementing two required functions job_submit and job_modify and two optional init and fini. While compiling .c into shared library is not a big deal there are situations where one would rather use script version, for those Slurm provides job_submit_lua plugin that simply calls lua script for real work. In this post I'll cover the issues I met configuring it.

An example here would be to develop a plugin that will prevent submission of jobs without explicit account specification. It is possible that users don't adjust account specification simply relying on the default, which can cause some internal billing issues. Slurm provides a sample lua script in the source tree. You can copy it, and change the job_submit function to very simple:

```
function slurm_job_submit(job_desc, part_list, submit_uid)

    if job_desc.account == nil then
        slurm.log_user("You have to specify account. Usage of
default accounts is forbidden.")
        return slurm.ESLURM_INVALID_ACCOUNT
    end
end
```

Next, add the JobSubmitPlugins=lua line to slurm.conf and restart slurmctld. Any job that is submitted now without a target account will return an error. In this sample, slurm_error.h has ESLURM_INVALID_ACCOUNT already defined so it will return this error:

```
[root@hpc-slurmtest slurm]# srun --pty bash -l
srun: error: You have to specify account. Usage of default accounts is
forbidden.
srun: error: Unable to allocate resources: Invalid account or
account/partition combination specified
```

In the Slurm v23.02 upgrade, job_submit/lua added support for log_user() from slurm_job_modify().

Job Submit Plugins Lab Exercises

In Slurm, there are many ways to configure plugins. And, like epilog and prolog scripts, they can be configured to start at any number of paths along a jobs' lifecycle. In this set of labs, you will try different job submit plugins

Exercise 1: Enable the throttle plugin

In this exercise, you will enable one of the supplied plugins called throttle. This plugin will evaluate the job submission frequency to see if there has been a threshold reached in the number of jobs submitted in a given timeframe. If so, it will send an error directly to STDOUT and fail the job submission.

1. As root, edit /etc/slurm/slurm.conf and add the following:

```
JobSubmitPlugins=throttle  
SchedulerParameters=jobs_per_user_per_hour=5
```

2. As root, re-read the configuration file:

```
scontrol reconfigure
```

3. Confirm the plugin is loaded by entering:

```
scontrol show config | grep -i plugin
```

You should see:

```
CoreSpecPlugin      = core_spec/none  
JobSubmitPlugins    = throttle  
MCSPlugin          = mcs/none  
NodeFeaturesPlugins = (null)  
PluginDir          = /usr/local/lib/slurm  
PowerPlugin         =  
RoutePlugin         = route/default  
TaskPlugin          = task/affinity  
TaskPluginParam     = (null type)  
TopologyPlugin      = topology/none
```

4. As fred, submit 6 jobs in sequence as follows:

```
sbatch -n1 --wrap="sleep 200"
```

You should get the following error:

```
[fred@login ~]$ sbatch -n1 --wrap="sleep 200"
Submitted batch job 2
[fred@login ~]$ sbatch -n1 --wrap="sleep 200"
Submitted batch job 3
[fred@login ~]$ sbatch -n1 --wrap="sleep 200"
Submitted batch job 4
[fred@login ~]$ sbatch -n1 --wrap="sleep 200"
Submitted batch job 5
[fred@login ~]$ sbatch -n1 --wrap="sleep 200"
Submitted batch job 6
[fred@login ~]$ sbatch -n1 --wrap="sleep 200"
sbatch: error: Reached jobs per hour limit

sbatch: error: Batch job submission failed: Job violates accounting/QOS
policy (job submit limit, user's size and/or time limits)
```

5. Check the log file:

```
ssh mgmtnode cat /var/log/slurmctld.log | grep -i "job submit"
```

You should see output similar to this:

```
[2021-10-01T10:04:57.828] _slurm_rpc_submit_batch_job: Job violates
accounting/QOS policy (job submit limit, user's size and/or time
limits)
```

Cleanup

1. As the root user, comment out the following two parameters in the /etc/slurm/slurm.conf file:

```
#JobSubmitPlugins=throttle
#SchedulerParameters=jobs_per_user_per_hour=5
```

2. As root, re-read the configuration file:

```
scontrol reconfigure
```

Exercise 2: Enable the require_timelimit plugin

In this exercise, you will enable one of the supplied plugins called require_timelimit. This plugin will evaluate the job submission to see if there is a time= value on the submission. If not, it will send an error directly to STDOUT and fail the job submission.

1. As root, edit /etc/slurm/slurm.conf and add the following:

```
JobSubmitPlugins=require_timelimit
```

2. As **root**, re-read the configuration file:

```
/lab_scripts/restart.sh
```

3. Confirm the plugin is loaded by entering:

```
scontrol show config|grep -i plugin
```

You should see:

```
CoreSpecPlugin      = core_spec/none
JobSubmitPlugins   = require_timelimit
MCSPlugin          = mcs/none
NodeFeaturesPlugins = (null)
PluginDir          = /usr/local/lib/slurm
PowerPlugin         =
RoutePlugin         = route/default
TaskPlugin          = task/affinity
TaskPluginParam     = (null type)
```

4. As **fred**, submit a job as follows:

```
sbatch -n1 --wrap="sleep 200"
```

You should get the following error:

```
sbatch: error: Batch job submission failed:
Time limit specification required, but not provided
```

5. Try with srun:

```
srun --pty bash
```

You should get the same error:

```
srun: error: Unable to allocate resources: Time limit specification
required, but not provided
```

6. Try with salloc:

```
salloc -n1 bash
```

You should get the same error:

```
salloc: error: Job submit/allocate failed: Time limit specification
required, but not provided
```

7. Now, try the above commands with a time value as an option:

```
sbatch -t20 -n1 --wrap="sleep 200"
```

```
srun -t20 --pty bash
```

```
salloc -t20 -n1 bash
```

They should run!

Cleanup

1. As root, cancel all the jobs.
2. Comment out the following in /etc/slurm/slurm.conf:

#JobSubmitPlugins=require_timelimit
3. As root, re-read the configuration file:
scontrol reconfigure

Exercise 3: Create a job_submit.lua plugin

In this exercise, you will create a job submit plugin that will do 2 things:

- Make sure every job submitted has an account
- Make sure every job submitted has a time value
- Log the errors in the slurmcld log file

If a job is submitted without these 2 switches (-t and/or -A), then they will return an error to STDOUT as well as the log file.

1. As root, create the file /etc/slurm/job_submit.lua and add the following to it:

```
function slurm_job_submit(job_desc, part_list, submit_uid)
    if job_desc.account == nil then
        slurm.log_user("--account option required")
        return slurm.ESLURM_INVALID_ACCOUNT
    end
    if job_desc.time_limit == slurm.NO_VAL then
        slurm.log_user("--time limit option required")
        return slurm.ESLURM_INVALID_TIME_LIMIT
    end

    return slurm.SUCCESS
end
function slurm_job_modify(job_desc, job_rec, part_list, modify_uid)
    return slurm.SUCCESS
end

return slurm.SUCCESS
```

2. As root, edit /etc/slurm/slurm.conf and add the following:

JobSubmitPlugins=lua

3. As **root**, re-read the configuration file:

```
scontrol reconfigure
```

4. Confirm the plugin is loaded by entering:

```
scontrol show config|grep -i plugin
```

You should see:

```
CoreSpecPlugin      = core_spec/none
JobSubmitPlugins   = lua
MCSPlugin          = mcs/none
NodeFeaturesPlugins = (null)
PluginDir          = /usr/local/lib/slurm
PowerPlugin         =
RoutePlugin         = route/default
TaskPlugin          = task/affinity
TaskPluginParam     = (null type)
```

5. As **fred**, submit a barebones job as follows:

```
sbatch -n1 --wrap="sleep 200"
```

You should get the following error:

```
sbatch: error: Batch job submission failed: Invalid account or
account/partition combination specified
```

6. Add an account to the job submit:

```
sbatch -A bedrock -n1 --wrap="sleep 200"
```

You should see:

```
sbatch: error: --time limit option required
sbatch: error: Batch job submission failed: Requested time limit is
invalid (missing or exceeds some limit)
```

7. Now try with time AND account parameters:

```
sbatch -A bedrock -t 20 -n1 --wrap="sleep 200"
```

You should see the job successfully submitted

Cleanup

1. As **root**, cancel all the jobs.

2. Comment out the following in the `/etc/slurm/slurm.conf`:

```
#JobSubmitPlugins=lua
```

3. As **root**, re-read the configuration file:

```
scontrol reconfigure
```

Exercise 4: Job Prolog Exercise

In this exercise, you will create a simple TaskProlog script that will print a batch job's jobID and node list to the job's stdout.

1. As **root**, create a file called `prolog.sh` in `/etc/slurm` and add the following content to it:

```
#!/bin/sh
#
# Sample TaskProlog script that will print a batch job's
# job ID and node list to the job's stdout
#
if [ "$SLURM_STEP_ID" = "0" -a "$SLURM_PROCID" = "0" ]
then
    echo -e "print =====\n"
    echo -e "print SLURM_JOB_ID = $SLURM_JOB_ID\n"
    echo -e "print SLURM_JOB_NODELIST = $SLURM_JOB_NODELIST\n"
    echo -e "print =====\n"
fi
```

2. Make the file executable (a new requirement for Slurm 23.02):

```
chmod +x /etc/slurm/prolog.sh
```

3. Edit the `/etc/slurm/slurm.conf` and add the following line:

```
TaskProlog=/etc/slurm/prolog.sh
```

4. As **root**, re-read the configuration file:

```
scontrol reconfigure
```

5. Confirm the prolog is configured by entering:

```
scontrol show config | grep -i prolog
```

You should see:

```
Prolog          = (null)
PrologEpilogTimeout = 65534
PrologSlurmctld   = (null)
PrologFlags       = Alloc,Contain,X11
ResvProlog        = (null)
SrunProlog        = (null)
TaskProlog        = /etc/slurm/prolog.sh
```

6. As **fred**, submit a job using sbatch:

```
sbatch -N1 --wrap="srun hostname"
```

7. Cat the output file:

```
cat slurm-9.out
```

You should see:

```
=====
SLURM_JOB_ID = 9
SLURM_JOB_NODELIST = node00
=====
node00
```

8. As fred, submit another job requiring more than one node using sbatch:

```
sbatch -N5 --wrap="srun hostname"
```

9. Cat the output file:

```
cat slurm-10.out
```

You should see:

```
[fred@login ~]$ cat slurm-10.out
=====
SLURM_JOB_ID = 10
SLURM_JOB_NODELIST = node[00-04]
=====
node04
node01
node02
node03
node00
```

Cleanup

1. As root, cancel all the jobs.

2. Comment out the following in the /etc/slurm/slurm.conf:

```
#TaskProlog=/etc/slurm/prolog.sh
```

3. As root, re-read the configuration file:

```
scontrol reconfigure
```

Exercise 5: SPANK Example

In this exercise, you will create a SPANK plugin to nice a binary (found in /lab_scripts).

1. As root, in root's home folder (/root), copy the file /lab_scripts/renice.c there:

```
cd /root
```

- ```
cp /lab_scripts/renice.c .
```
2. Build the plugin:  
`gcc -fPIC -std=c99 -shared -o renice.so renice.c`
3. Confirm the symbols are built by typing:  
`nm renice.so`
4. Confirm the shared libraries are available:  
`ldd renice.so`
5. **As root**, add the renice.so entry to end of the plugstack.conf in /etc/slurm/. Make it look like this:
- ```
include /etc/slurm/plugstack.conf.d/*
optional           renice.so          min_prio=-20
```
6. Edit the /etc/slurm/slurm.conf file and modify the "PluginDir=" directive so that it is uncommented and contains the following:
- ```
PluginDir=/usr/local/lib/slurm
```
7. Copy the .so to the Slurm plugin directory (as defined by plugindir in slurm.conf):  
`cp ~/renice.so /usr/local/lib/slurm`
8. Send the renice.so out to the compute nodes:  
`pdcp ~/renice.so /usr/local/lib/slurm`
9. **As the root user**, re-read the configuration file:  
`scontrol reconfigure`
10. **As fred**, submit a job:  
`sbatch -N1 -t300 --wrap="srun --renice=-20 sleep 300"`
11. **As the job is running**, ssh to the node on which the job is running  
`ssh node00`
12. Enter:  
`ps ax -o pid,ni,cmd | grep "sleep\|NI"`

You should see:

```
[root@node00 ~]# ps ax -o pid,ni,cmd | grep "sleep\|NI"
 PID NI CMD
 3210 0 sleep 100000000
 3219 0 srun --renice=-20 sleep 300
 3222 0 srun --renice=-20 sleep 300
 3241 -20 /bin/sleep 300
 3243 0 grep --color=auto sleep\|NI
```

That's it.

## Exercise 6: SPANK and LUA Job Submit Example

In this exercise, you will put together a SPANK plugin created in the previous exercise and a lua job submission script so that depending on which account a job is assigned, a renice value will be applied.

1. Make sure the following setting is in the `/etc/slurm/slurm.conf`:

```
jobsubmitplugins(lua)
```

- 2.

As root user, copy the `/lab_scripts/job_submit_spank.lua` to `/etc/slurm` directory and call it `job_submit.lua`:

```
cp /lab_scripts/job_submit_spank.lua /etc/slurm/job_submit.lua
```

3. Create the 2 project accounts:

```
sacctmgr -i create account projecta,projectb
```

4. Assign users to the accounts:

```
sacctmgr -i create user fred account=projecta
```

```
sacctmgr -i create user barney account=projectb
```

5. Restart the controller:

```
scontrol reconfigure
```

6. Now as fred, submit the following job command:

```
sbatch -A projecta -N1 -n1 -t300 --wrap="srun sleep 300"
```

7. As the job is running, ssh to the node on which the job is running and type the following to confirm the re-assignment of the nice value based on the account submitted to:

```
ps ax -o pid,ni,cmd | grep "sleep\|NI"
```

8. Now as Barney, submit the following job command:

```
sbatch -A projectb -N1 -n1 -t300 --wrap="srun sleep 300"
```

9. As the job is running, ssh to the node on which the job is running and type the following to confirm the re-assignment of the nice value based on the account submitted to:

```
ps ax -o pid,ni,cmd | grep "sleep\|NI"
```

**NOTE:** The reason Fred's job ended up with a nice value of -10 is because the Submit script applies the policy based on the target account of projecta. Similarly, Barney's job is automatically assigned a nice value of 15 because the job submit script forces it due to his projectb destination.

## Cleanup

1. Reset the Docker Slurm cluster environment back to its original, pristine state by exiting the login Docker container back to the Ubuntu AWS prompt. And **as the ubuntu user**:

```
cd ~/docker-scale-out
make clean && make
```

## Prometheus-Slurm-Exporter

The Prometheus collector and exporter for metrics extracted from the Slurm resource scheduling system is provided and supported by the Slurm community. The exporter can gather metrics and push them to a Prometheus server, such as:

### State of the CPUs:

- **Allocated:** CPUs which have been allocated to a job
- **Idle:** CPUs not allocated to a job and thus available for use
- **Other:** CPUs which are unavailable for use at the moment
- **Total:** Total number of CPUs

### State of the Nodes:

- **Allocated:** Nodes which has been allocated to one or more jobs
- **Completing:** All jobs associated with these nodes are in the process of being completed.
- **Down:** Nodes which are unavailable for use
- **Drain:** With this metric, two different states are accounted for:
  - Nodes in drained state (marked unavailable for use per system administrator request)
  - Nodes in draining state (currently executing jobs but which will not be allocated for new ones)
- **Fail:** These nodes are expected to fail soon and are unavailable for use per system administrator request.
- **Error:** Nodes which are currently in an error state and not capable of running any jobs
- **Idle:** Nodes not allocated to any jobs and thus available for use
- **Maint:** Nodes which are currently marked with the maintenance flag
- **Mixed:** Nodes which have some of their CPUs ALLOCATED while others are IDLE
- **Resv:** These nodes are in an advanced reservation and not generally available.

### Status of the Jobs:

- **PENDING:** Jobs awaiting for resource allocation.
- **PENDING\_DEPENDENCY:** Jobs awaiting because of an unexecuted job dependency.
- **RUNNING:** Jobs currently allocated.
- **SUSPENDED:** Job has an allocation but execution has been suspended and CPUs have been released for other jobs.
- **CANCELLED:** Jobs which were explicitly cancelled by the user or system administrator.
- **COMPLETING:** Jobs which are in the process of being completed.
- **COMPLETED:** Jobs have terminated all processes on all nodes with an exit code of zero.
- **CONFIGURING:** Jobs have been allocated resources, but are waiting for them to become ready for use.
- **FAILED:** Jobs terminated with a non-zero exit code or other failure condition.
- **TIMEOUT:** Jobs terminated upon reaching their time limit.
- **PREEMPTED:** Jobs terminated due to preemption.
- **NODE\_FAIL:** Jobs terminated due to failure of one or more allocated nodes.

### State of the Partitions:

- **Running/suspended** Jobs per partitions, divided between Slurm accounts and users.
- **CPUs** total/allocated/idle per partition plus used CPU per user ID.

Jobs Information Per Account and User:

- **Running/Pending/Suspended** jobs per SLURM Account.
- **Running/Pending/Suspended** jobs per SLURM User.

Scheduler Information:

- **Server Thread count:** The number of current active slurmctld threads.
- **Queue size:** The length of the scheduler queue.
- **DBD Agent queue size:** The length of the message queue for SlurmDBD.
- **Last cycle:** Time in microseconds for last scheduling cycle.
- **Mean cycle:** Mean of scheduling cycles since last reset.
- **Cycles per minute:** Counter of scheduling executions per minute.
- **(Backfill) Last cycle:** Time in microseconds of last backfilling cycle.
- **(Backfill) Mean cycle:** Mean of backfilling scheduling cycles in microseconds since last reset.
- **(Backfill) Depth mean:** Mean of processed jobs during backfilling scheduling cycles since last reset.
- **(Backfill) Total Backfilled Jobs (since last slurm start):** Number of jobs started thanks to backfilling since last Slurm start.
- **(Backfill) Total Backfilled Jobs (since last stats cycle start):** Number of jobs started thanks to backfilling since last time stats were reset.
- **(Backfill) Total backfilled heterogeneous Job components:** Number of heterogeneous job components started thanks to backfilling since last Slurm start.

Once configured, the information that is scraped from the exporter into Prometheus can be graphed. One of the common platforms for doing this is Grafana.

## Graphing Prometheus data with Grafana

Grafana is open source visualization and analytics software. It allows you to query, visualize, alert on, and explore your metrics no matter where they are stored. In plain English, it provides you with tools to turn your time-series database (TSDB) data into beautiful graphs and visualizations.

After creating a dashboard, there are many possible things you might do next. It all depends on your needs and your use case.

For example, if you want to view weather data and statistics about your smart home, then you might create a playlist. If you are the administrator for a corporation and are managing Grafana for multiple teams, then you might need to set up provisioning and authentication.

Integrating Slurm accounting records captured via the InfluxDB database with a graphing tool provides administrators a great way to visualize the utilization of scheduled cluster resources.

## Prometheus, Prometheus-slurm-exporter, and Grafana Lab Exercises

The training environment comes with Grafana already installed. This lab will show you how to install Prometheus, the Slurm exporter, and then create a dashboard and panels to monitor Slurm performance in real-time.

### Exercise 1: Install Prometheus

These exercises need to be performed **in the mgmtnode docker container as root user**. You can get there by either ssh'ing from the login container, or do: “**docker-compose exec mgmtnode bash**” from the AWS host.

1. In these labs, we don't need the cloud nodes and their partition. So disable them:

On the **mgmtnode host**: Edit `/etc/slurm/slurm.conf` and comment out the following:

```
#NodeName=cloud[0000-1024] Weight=8 Feature=cloud State=CLOUD
#PartitionName=cloud Nodes=cloud[0000-1024] Default=no
```

Then restart the controller:

```
/lab_scripts/restart.sh
```

2. Start by creating a Prometheus user and group. **As root, on the mgmtnode container**:

```
useradd -m -s /bin/false prometheus
```

3. Verify it was created:

```
id prometheus
```

You should see:

```
[root@mgmtnode ~]# id prometheus
uid=1014(prometheus) gid=1014(prometheus) groups=1014(prometheus)
[root@mgmtnode ~]#
```

4. Create the configuration directories for Prometheus:

```
mkdir /etc/prometheus
mkdir /var/lib/prometheus
```

5. Set the ownership on /var/lib/prometheus:

```
chown prometheus /var/lib/prometheus/
```

6. Download Prometheus (**the following is all one command that wraps around**):

```
wget https://github.com/prometheus/prometheus/releases/download/v2.47.0/\\
prometheus-2.47.0.linux-amd64.tar.gz -P /tmp
```

7. Extract the tarball:

```
cd /tmp
tar -zxpvf prometheus-2.47.0.linux-amd64.tar.gz
```

8. Copy the binaries:

```
cd /tmp/prometheus-2.47.0.linux-amd64
cp prometheus /usr/local/bin
cp promtool /usr/local/bin
```

9. Copy the configuration file for Prometheus from /lab\_scripts:

```
cp /lab_scripts/prometheus.yml /etc/prometheus
```

10. Copy the Systemd service unit file for the Prometheus Server from /lab\_scripts:

```
cp /lab_scripts/prometheus.service /etc/systemd/system/
```

11. Enable and start the service::

```
systemctl daemon-reload
systemctl enable --now prometheus
```

12. Check the status:

```
systemctl status prometheus
netstat -tunlp | grep prometheus
```

13. Confirm the installation by running:

```
curl http://localhost:9090/metrics
```

You should see:

```
TYPE go_gc_duration_seconds summary
go_gc_duration_seconds{quantile="0"} 4.5672e-05
go_gc_duration_seconds{quantile="0.25"} 5.4797e-05
go_gc_duration_seconds{quantile="0.5"} 0.000558858
go_gc_duration_seconds{quantile="0.75"} 0.00255599
go_gc_duration_seconds{quantile="1"} 0.003843951
go_gc_duration_seconds_sum 0.007059268
go_gc_duration_seconds_count 5
HELP go_goroutines Number of goroutines that
currently exist.
TYPE go_goroutines gauge
go_goroutines 39
HELP go_info Information about the Go environment.
...
prometheus_tsdb_wal_truncations_total 0
HELP promhttp_metric_handler_requests_in_flight
Current number of scrapes being served.
TYPE promhttp_metric_handler_requests_in_flight
gauge
promhttp_metric_handler_requests_in_flight 1
HELP promhttp_metric_handler_requests_total Total
number of scrapes by HTTP status code.
TYPE promhttp_metric_handler_requests_total counter
promhttp_metric_handler_requests_total{code="200"} 9
promhttp_metric_handler_requests_total{code="500"} 0
promhttp_metric_handler_requests_total{code="503"} 0
[root@mgmtnode prometheus-2.44.0.linux-amd64]#
```

14. If you are have a terminal emulator open (MacOS or Linux, or Cygwin in Windows), initiate a tunnel to the Prometheus service in the AWS environment:

```
ssh -L 9090:10.11.1.1:9090 -i slurm_training.pem ubuntu@<YOUR AWS IP>
```

15. If you are running PuTTy (in Windows), do the following:

- a. Launch putty.exe
- b. From the left frame, go to **Session > Host Name** (enter the **AWS\_Public\_IP\_Address**)
- c. Go to **SSH > Tunnels** (Check the box labeled "**Local ports accept connections for other hosts**")
- d. Go to **SSH > Tunnels > Source Port** and enter **9090**
- e. Go to **SSH > Tunnels > Destination** and enter **10.11.1.1:9090**
- f. Click **Add**
- g. On the same page enter source port **3000**, and in the Destination box enter **10.11.1.20:3000**
- h. Click **Add**
- i. Go to **SSH > Auth** and browse to and select **slurm\_training.ppk** file
- j. Go to **Session > Open**
- k. Login as user **ubuntu**

16. Open a browser on your own computer and point to:

**http://localhost:9090**

You should see:

The screenshot shows the Prometheus web interface with a graph tab selected. At the top, there are tabs for Prometheus, Alerts, Graph, Status, and Help. Below the tabs, there is a search bar labeled "Expression (press Shift+Enter for newlines)" and an "Execute" button. The main area displays a graph with a single data series named "Moment". The graph has two points: one at approximately 1480000000000 with a value of 1, and another at approximately 1480000000000 with a value of 0. A "no data" message is visible below the graph. On the right side, there is a "Remove Graph" button and an "Add Graph" button.

17. Click on the '**Status**' tab and then click on '**Targets**'

You should see:

The screenshot shows the Prometheus Targets page. At the top, there is a navigation bar with tabs for Prometheus, Alerts, Graph, Status, and Help. Below the navigation bar, there is a search bar with the URL "10.11.1.1:9090/targets?search=". The main content area is titled "Targets" and shows a table with one entry: "prometheus (1/1 up)". The table has columns for Endpoint, State, Labels, Last Scrape, Scrape Duration, and Error. The "Endpoint" column shows "http://localhost:9090/metrics", the "State" column shows "UP", and the "Labels" column shows "Instance='localhost:9090'" and "Job='prometheus'". The "Last Scrape" column shows "643.000ms ag", the "Scrape Duration" column shows "7.588ms", and the "Error" column is empty. There are also buttons for "All scrape pools", "All", "Unhealthy", "Collapse All", "Filter by endpoint or labels", and checkboxes for "Unknown", "Unhealthy", and "Healthy".

Note: I am taking the screenshots on a local VM, not AWS. So the URL will be different in your case

## Exercise 2: Install and Configure node\_exporter

Node exporter is a utility that collects and ships a vast array of Linux system metrics such as CPU, memory usage, filesystems and network statistics. In this section, we are going to install node\_exporter on the Prometheus server (mgmtnode) and on a remote compute nodes. **You will do all this as root on the mgmtnode.**

1. Create the node\_exporter user account:

```
useradd -m -s /bin/false node_exporter
pdsh "useradd -m -s /bin/false node_exporter"
```

**NOTE:** You can ignore the skel and mailbox warnings

2. Download node\_exporter to the mgmtnode (**the following is all one command that wraps around**):

```
wget https://github.com/prometheus/node_exporter/releases/download/v1.6.1/\\
node_exporter-1.6.1.linux-amd64.tar.gz
```

3. Download the node\_exporter to the compute nodes(**the following is all one command that wraps around**):

```
pdsh "wget https://github.com/prometheus/node_exporter/releases/download/v1.6.1/\\
node_exporter-1.6.1.linux-amd64.tar.gz"
```

4. Extract the tarball:

```
tar -zxpvf node_exporter-1.6.1.linux-amd64.tar.gz
pdsh "tar -zxpvf node_exporter-1.6.1.linux-amd64.tar.gz"
```

5. Copy the binaries to the right place on the mgmtnode, and then on the compute nodes:

```
cp node_exporter-1.6.1.linux-amd64/node_exporter /usr/local/bin
pdsh "cp node_exporter-1.6.1.linux-amd64/node_exporter /usr/local/bin"
```

6. Set the correct file permissions on the mgmtnode, and then on the compute nodes:

```
chown node_exporter:node_exporter /usr/local/bin/node_exporter
pdsh "chown node_exporter:node_exporter /usr/local/bin/node_exporter"
```

7. Copy the service file for node\_exporter from /lab\_scripts to the /etc/systemd/system directory:

```
cp /lab_scripts/node_exporter.service /etc/systemd/system/
```

8. Send the service file to the same location on all of the compute nodes:

```
pdcp /etc/systemd/system/node_exporter.service /etc/systemd/system
```

9. Start the service on the mgmtnode and on the compute nodes, and then enable it in those places, as well:

```
systemctl daemon-reload
pdsh systemctl daemon-reload
systemctl enable --now node_exporter
pdsh systemctl enable --now node_exporter
```

10. Copy the updated node\_exporter YML to the Prometheus service:

```
cp /lab_scripts/prometheus.update-1.yml /etc/prometheus/prometheus.yml
```

#### Confirm the overwrite

**NOTE: Make sure all the targets are populated in the /etc/prometheus/prometheus.yml before continuing. You may have to re-copy the prometheus.update-1.yml to make sure**

11. Restart Prometheus:

```
systemctl restart prometheus
```

12. Go back to your browser and point to the Prometheus Targets page:

| Endpoint                      | State | Labels                                        | Last Scrape   | Scrape Duration | Error |
|-------------------------------|-------|-----------------------------------------------|---------------|-----------------|-------|
| http://10.11.5.0:9100/metrics | UP    | instance="10.11.5.0:9100" job="node_exporter" | 7.917s ago    | 48.157ms        |       |
| http://10.11.5.1:9100/metrics | UP    | instance="10.11.5.1:9100" job="node_exporter" | 6.335s ago    | 35.171ms        |       |
| http://10.11.5.3:9100/metrics | UP    | instance="10.11.5.3:9100" job="node_exporter" | 276.000ms ago | 34.373ms        |       |
| http://10.11.5.7:9100/metrics | UP    | instance="10.11.5.7:9100" job="node_exporter" | 14.435s ago   | 37.928ms        |       |
| http://10.11.5.8:9100/metrics | UP    | instance="10.11.5.8:9100" job="node_exporter" | 12.401s ago   | 49.396ms        |       |
| http://10.11.5.2:9100/metrics | UP    | instance="10.11.5.2:9100" job="node_exporter" | 3.502s ago    | 35.114ms        |       |
| http://10.11.5.4:9100/metrics | UP    | instance="10.11.5.4:9100" job="node_exporter" | 9.175s ago    | 41.641ms        |       |
| http://10.11.5.9:9100/metrics | UP    | instance="10.11.5.9:9100" job="node_exporter" | 14.959s ago   | 40.916ms        |       |
| http://10.11.5.6:9100/metrics | UP    | instance="10.11.5.6:9100" job="node_exporter" | 8.42s ago     | 41.690ms        |       |
| http://localhost:9100/metrics | UP    | instance="localhost:9100" job="node_exporter" | 13.52s ago    | 43.238ms        |       |
| http://10.11.5.5:9100/metrics | UP    | instance="10.11.5.5:9100" job="node_exporter" | 4.128s ago    | 34.486ms        |       |

| Endpoint                      | State | Labels                                     | Last Scrape | Scrape Duration | Error |
|-------------------------------|-------|--------------------------------------------|-------------|-----------------|-------|
| http://localhost:9090/metrics | UP    | instance="localhost:9090" job="prometheus" | 2.161s ago  | 3.950ms         |       |

## Exercise 3: Install the prometheus-slurm-exporter

Just as the node-exporter feeds Prometheus server with valuable node data, so will the prometheus-slurm-exporter feed prometheus with valuable Slurm metrics. As with the other exercises, perform this as root on the mgmtnode container.

1. Install the “Go” utility from source:

```
export VERSION=1.15 OS=linux ARCH=amd64
wget https://dl.google.com/go/go$VERSION.$OS-$ARCH.tar.gz
tar -xzvf go$VERSION.$OS-$ARCH.tar.gz
export PATH=$PWD/go/bin:$PATH
```

2. Git clone and make the prometheus-slurm-exporter:

```
git clone https://github.com/vpenso/prometheus-slurm-exporter.git
cd prometheus-slurm-exporter
make
```

3. Start the exporter in the background and make sure it runs:

```
./bin/prometheus-slurm-exporter &
```

See if it’s gathering data:

```
curl http://localhost:8080/metrics
```

You should see:

```
TYPE go_gc_duration_seconds summary
go_gc_duration_seconds{quantile="0"} 0
go_gc_duration_seconds{quantile="0.25"} 0
go_gc_duration_seconds{quantile="0.5"} 0
go_gc_duration_seconds{quantile="0.75"} 0
go_gc_duration_seconds{quantile="1"} 0
go_gc_duration_seconds_sum 0
go_gc_duration_seconds_count 0
HELP go_goroutines Number of goroutines that currently exist.
TYPE go_goroutines gauge
go_goroutines 14
HELP go_info Information about the Go environment.
TYPE go_info gauge
go_info{version="go1.15"} 1
...
lurm_scheduler_dbd_queue_size 0
HELP slurm_scheduler_last_cycle Information provided by the Slurm sdiag
command, scheduler last cycle time in (microseconds)
TYPE slurm_scheduler_last_cycle gauge
slurm_scheduler_last_cycle 113
HELP slurm_scheduler_mean_cycle Information provided by the Slurm sdiag
command, scheduler mean cycle time in (microseconds)
TYPE slurm_scheduler_mean_cycle gauge
slurm_scheduler_mean_cycle 128
HELP slurm_scheduler_queue_size Information provided by the Slurm sdiag
command, length of the scheduler queue
TYPE slurm_scheduler_queue_size gauge
slurm_scheduler_queue_size 0
HELP slurm_scheduler_threads Information provided by the Slurm sdiag
command, number of scheduler threads
TYPE slurm_scheduler_threads gauge
slurm_scheduler_threads 5
[root@mgmtnode prometheus-slurm-exporter]#
```

4. Run some jobs to see if captures the metrics, then run the curl command again
5. Break out of it
6. Copy the binary to the /usr/bin directory:  
`cp ./bin/prometheus-slurm-exporter /usr/bin`
7. Copy the unit file into place to be used by systemd:  
`cp /lab_scripts/prometheus-slurm-exporter.service /etc/systemd/system/`
8. Enable and start the service:  
`systemctl daemon-reload`

```
systemctl enable prometheus-slurm-exporter
systemctl start prometheus-slurm-exporter
```

- Copy the updated Prometheus YML for the new scraper:

```
cp /lab_scripts/prometheus.update-2.yml /etc/prometheus/prometheus.yml
```

#### Confirm the overwrite

**NOTE:** Make sure all the targets are populated in the /etc/prometheus/prometheus.yml before continuing. You may have to re-copy the prometheus.update-2.yml to make sure

- Restart the Prometheus server:

```
systemctl restart prometheus
```

- Back in your browser (pointing at Prometheus), see the added targets:

| node_exporter (11/11 up)      |       |                                               |               |                 |       |
|-------------------------------|-------|-----------------------------------------------|---------------|-----------------|-------|
| Endpoint                      | State | Labels                                        | Last Scrape   | Scrape Duration | Error |
| http://10.11.5.9:9100/metrics | UP    | instance="10.11.5.9:9100" job="node_exporter" | 9.192s ago    | 42.030ms        |       |
| http://10.11.5.1:9100/metrics | UP    | instance="10.11.5.1:9100" job="node_exporter" | 568.000ms ago | 33.352ms        |       |
| http://10.11.5.5:9100/metrics | UP    | instance="10.11.5.5:9100" job="node_exporter" | 13.360s ago   | 41.417ms        |       |
| http://10.11.5.7:9100/metrics | UP    | instance="10.11.5.7:9100" job="node_exporter" | 8.668s ago    | 38.025ms        |       |
| http://localhost:9100/metrics | UP    | instance="localhost:9100" job="node_exporter" | 7.284s ago    | 36.923ms        |       |
| http://10.11.5.4:9100/metrics | UP    | instance="10.11.5.4:9100" job="node_exporter" | 3.409s ago    | 41.711ms        |       |
| http://10.11.5.8:9100/metrics | UP    | instance="10.11.5.8:9100" job="node_exporter" | 6.634s ago    | 35.393ms        |       |
| http://10.11.5.2:9100/metrics | UP    | instance="10.11.5.2:9100" job="node_exporter" | 12.736s ago   | 35.865ms        |       |
| http://10.11.5.6:9100/metrics | UP    | instance="10.11.5.6:9100" job="node_exporter" | 2.275s ago    | 31.949ms        |       |
| http://10.11.5.0:9100/metrics | UP    | instance="10.11.5.0:9100" job="node_exporter" | 2.151s ago    | 33.853ms        |       |
| http://10.11.5.3:9100/metrics | UP    | instance="10.11.5.3:9100" job="node_exporter" | 9.511s ago    | 37.959ms        |       |

| prometheus (1/1 up)           |       |                                            |             |                 |       |
|-------------------------------|-------|--------------------------------------------|-------------|-----------------|-------|
| Endpoint                      | State | Labels                                     | Last Scrape | Scrape Duration | Error |
| http://localhost:9090/metrics | UP    | instance="localhost:9090" job="prometheus" | 11.395s ago | 18.964ms        |       |

| prometheus-slurm-exporter (1/1 up) |       |                                                           |             |                 |       |
|------------------------------------|-------|-----------------------------------------------------------|-------------|-----------------|-------|
| Endpoint                           | State | Labels                                                    | Last Scrape | Scrape Duration | Error |
| http://localhost:8080/metrics      | UP    | instance="localhost:8080" job="prometheus-slurm-exporter" | 4.189s ago  | 38.133ms        |       |

## Exercise 4: Configure Slurm for multiple accounts and generate some load

With the Prometheus service installed, it's time to configure Slurm with accounts and put some load on it.

- As root, change directory to /lab\_scripts on Mgmtnode

- Run the following to import users and account data into the database:  
`sacctmgr -i load file=cluster.cfg`

- Update the `/etc/slurm/slurm.conf` with the following changes:

```
From:
#PriorityType=priority/multifactor
#PriorityDecayHalfLife=14-0
#PriorityUsageResetPeriod=MONTHLY
#PriorityWeightFairshare=100000
#PriorityWeightAge=1000
#PriorityWeightPartition=10000
#PriorityWeightJobSize=1000
#PriorityMaxAge=1-0
#PriorityCalcPeriod=1

To:
PriorityType=priority/multifactor #UNCOMMENT THIS
PriorityDecayHalfLife=10 #CHANGE AND UNCOMMENT THIS
#PriorityUsageResetPeriod=MONTHLY
PriorityWeightFairshare=100000 #UNCOMMENT THIS
#PriorityWeightAge=1000
#PriorityWeightPartition=10000
#PriorityWeightJobSize=1000
#PriorityMaxAge=1-0
PriorityCalcPeriod=1 #UNCOMMENT THIS
```

- Restart the Slurm controller by typing:

```
/lab_scripts/restart.sh
```

- Now load the cluster with jobs:

```
bash /lab_scripts/test.sh
```

## Exercise 5: Integrate Prometheus with Grafana

Now that you are collecting node and Slurm data, let's graph it.

- Initiate another tunnel, this time to the Grafana service (on port 3000 in the Grafana docker container) in the AWS environment (SEE NOTE BELOW):

```
ssh -L 3000:10.11.1.20:3000 -i slurm_training.pem ubuntu@<YOUR AWS IP>
```

NOTE: If you are running Windows and PuTTy, the ports (9090 and 3000) opened in a previous step are good. No need to do anything else here

- Open a browser on your own computer and point to:  
`http://localhost:3000`
- Login as user=**admin**, password=**admin**.

4. When asked to change password, keep the username and password as "admin".
5. On the main page, click "Add your first data source" in the COMPLETE box.
6. Select Prometheus in the Time series databases.
7. In the HTTP section, change the Prometheus server URL to <http://10.11.1.1:9090>
8. Save and Test the data source
9. Go back to the main Grafana page. Press the 3 horizontal lines drop-down next to the word Home
10. Click "Dashboards". Select the New button and select Import.
11. In the field labeled "Import via Grafana.com" enter:  
`4323`  
Click Load.
12. On the Import screen, choose the Prometheus data source of Prometheus and click Import.
13. Change the refresh to 5 seconds and time range of 30 minutes.
14. Try taking a node offline and see the output in Grafana!

## Cleanup

1. Reset the Docker Slurm cluster environment back to its original, pristine state by exiting the login Docker container back to the Ubuntu AWS prompt. And as ubuntu:  
`cd ~/docker-scale-out  
make clean && make`

## InfluxDB Plugin

InfluxDB is a time series database designed to handle high write and query loads. It is an integral component of the TICK stack. InfluxDB is meant to be used as a backing store for any use case involving large amounts of timestamped data, including DevOps monitoring, application metrics, IoT sensor data, and real-time analytics.

Here are some of the features that InfluxDB currently supports that make it a great choice for working with time series data like Slurm performance metrics:

- Custom high performance datastore written specifically for time series data. The TSM engine allows for high ingest speed and data compression
- Written entirely in Go. It compiles into a single binary with no external dependencies.
- Simple, high performing write and query HTTP APIs.
- Plugins support for other data ingestion protocols such as Graphite, collectd, and OpenTSDB.
- Expressive SQL-like query language tailored to easily query aggregated data.
- Tags allow series to be indexed for fast and efficient queries.
- Retention policies efficiently auto-expire stale data.
- Continuous queries automatically compute aggregate data to make frequent queries more efficient.

## Influx Installation

Installing InfluxDB is a fairly straightforward endeavor. Once you have created a local repo, you can yum install all the required components.

Once you have started the InfluxDB service, you can create a database using the included toolkit.

## Integration with Slurm

The Slurm integration with InfluxDB requires a special plugin. It is called:

```
AcctGatherprofileType=acct_gather_profile/influxdb
```

This enables the influxdb plugin. The influxdb instance host, port, database, retention policy and which values are collected are configured in the `acct_gather.conf` file

In addition to the required changes in the Slurm configuration file, a file named `acct_gather.conf` with the following in it:

```
ProfileInfluxDBDatabase=slurm
ProfileInfluxDBDefault=All
ProfileInfluxDBHost=mgmtnode:8086
ProfileInfluxDBRTPolicy=autogen
```

## Graphing InfluxDB data with Grafana

Grafana is open source visualization and analytics software. It allows you to query, visualize, alert on, and explore your metrics no matter where they are stored. In plain English, it provides you with tools to turn your time-series database (TSDB) data into beautiful graphs and visualizations.

After creating a dashboard, there are many possible things you might do next. It all depends on your needs and your use case.

For example, if you want to view weather data and statistics about your smart home, then you might create a playlist. If you are the administrator for a corporation and are managing Grafana for multiple teams, then you might need to set up provisioning and authentication.

Integrating Slurm accounting records captured via the InfluxDB database with a graphing tool provides administrators a great way to visualize the utilization of scheduled cluster resources.

## InfluxDB and Grafana Lab Exercises

The training environment comes with InfluxDB v2.0 and Grafana installed. This lab will show you how to create a simple dashboard and panels to monitor Slurm performance in real-time.

Remember that these are 3<sup>rd</sup> party products (InfluxDB and Grafana) and are not directly support by SchedMD. They are used here as examples of monitoring capabilities only.

### Exercise 1: Verify the InfluxDB Installation

**These exercises need to be performed in the login docker container.**

1. Edit `/etc/slurm/slurm.conf` and confirm the following entry:

```
AcctGatherProfileType=acct_gather_profile/influxdb
```

2. Confirm that the file called `/etc/slurm/acct_gather.conf` exists and contains the following:

```
ProfileInfluxDBDatabase=scaleout
ProfileInfluxDBHost=influxdb:8086
ProfileInfluxDBDefault=All
ProfileInfluxDBRTPolicy=scaleout
ProfileInfluxDBPass=password
ProfileInfluxDBUser=user
```

3. Confirm the database is created and is listening by issuing the following command (**the following is all one line**):

```
curl -G http://10.11.1.19:8086/query -u user:password --data-urlencode "q=SHOW DATABASES" | jq
```

It should show:

```

% Total % Received % Xferd Average Speed Time Time Time Current
 Dload Upload Total Spent Left Speed
100 108 100 108 0 0 54000 0 --::-- --::-- --::-- 105k
{
 "results": [
 {
 "statement_id": 0,
 "series": [
 {
 "name": "databases",
 "columns": [
 "name"
],
 "values": [
 [
 "scaleout"
]
]
 }
]
 }
]
}

```

4. Before being able to query the database on any measurement, there has to be some data in the influxdb.

As fred, submit a couple of jobs:

```
srun -N1 hostname
```

```
srun -N2 hostname
```

```
srun -N3 hostname
```

5. Query the influxdb to see what are the measurement (schema) defined (**the following is all one line**):

```
curl --get http://10.11.1.19:8086/query?db=scaleout --header
"Authorization: Token token" --data-urlencode "q=SHOW measurements" |
jq
```

It should show:

```

% Total % Received % Xferd Average Speed Time Time Time Current
 Dload Upload Total Spent Left Speed
100 198 100 198 0 0 99000 0 --::-- --::-- --::-- 99000
{
 "results": [
 {
 "statement_id": 0,
 "series": [
 {
 "name": "measurements",
 "columns": [
 "name"
],
 "values": [
 [
 "CPUFrequency"
],
 [
 "CPUTime"
],
 [
 "CPUUtilization"
],
 [
 "Pages"
],
 [
 "RSS"
],
 [
 "ReadMB"
],
 [
 "VMSize"
],
 [
 "WriteMB"
]
]
 }
]
 }
]
}

```

- Now run a query in the login container to see if the data made it into the database (**the following is all one line**):

```

curl -G http://10.11.1.19:8086/query?db=scaleout --header
"Authorization: Token token" -u user:password --data-urlencode
"q=SELECT * FROM CPUUtilization"

```

Should show:

```
{"results": [{"statement_id": 0, "series": [{"name": "CPUUtilization", "columns": ["time", "host", "job", "step", "task", "value"], "values": [[{"time": "2021-10-01T16:32:39Z", "host": "node00", "job": "1", "step": "-4", "task": "0", "value": 0}, {"time": "2021-10-01T16:32:39Z", "host": "node00", "job": "1", "step": "0", "task": "0", "value": 0}]]}]}]
```

7. Next, generate some usage on the cluster so we can graph it. **As fred**, in the login container, enter the following (**the following is all one line**):

```
for x in {1..1000} ; do sbatch -N1 --exclusive --mem=1000 -t1 --wrap="srun -c 1 -i 1 -m 1 --vm-bytes=100M -t 5s ; sleep 1" ; sleep 2 ; done
```

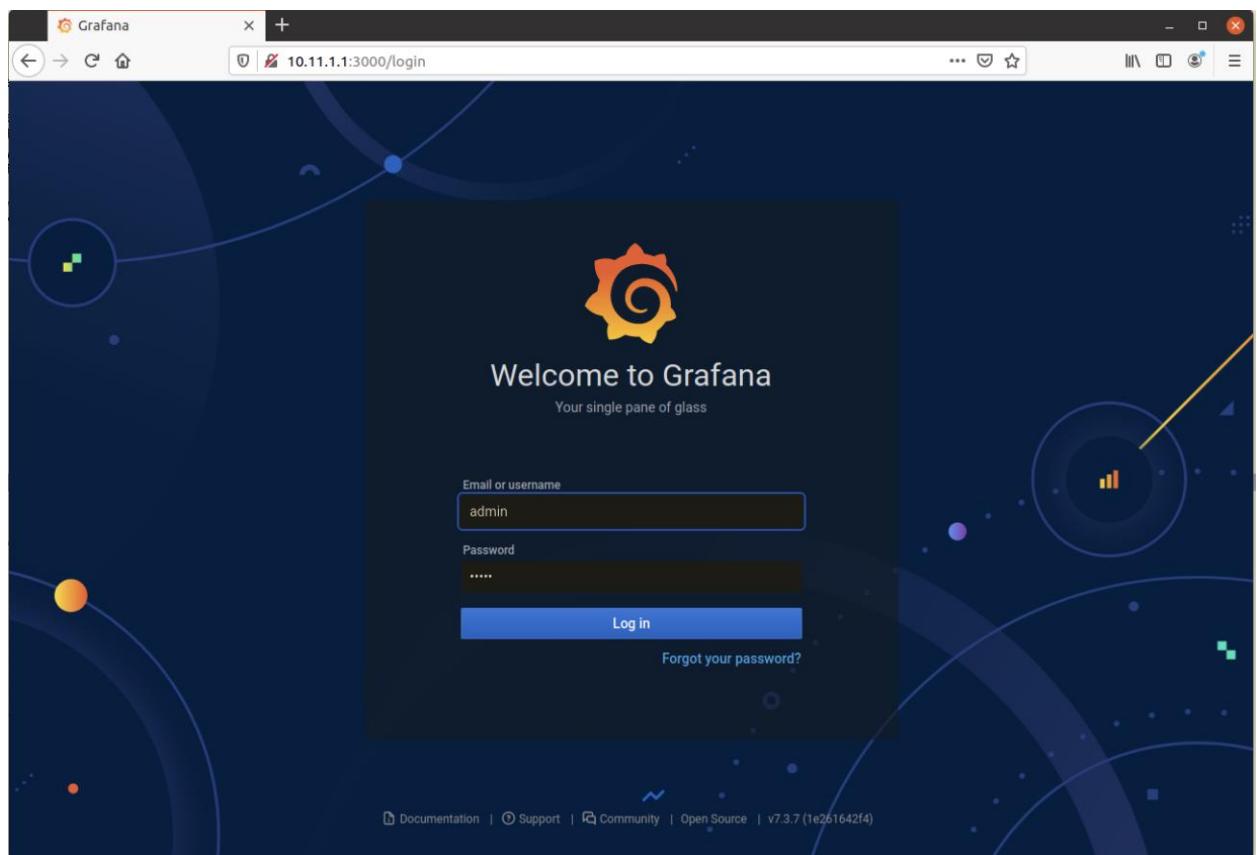
## Exercise 2: Access the Grafana Dashboard

Grafana presents its collected information via a web browser. In the AWS training environment, there is no GUI. So if you have an Xserver on your computer that is accessing the learning environment, you can create an SSH tunnel and launch a browser protocol from AWS to display on your computer.

To initiate this, do the following:

1. Create an SSH tunnel to attach to the AWS instance from your own terminal on your computer:  
`ssh -L 3000:10.11.1.20:3000 -i slurm_training.pem ubuntu@<YOUR AWS IP>`
2. Open a browser on your own computer and point to:  
`http://localhost:3000`

You should see:



3. Enter the username and password of **admin**.
4. When asked, re-enter the password of **admin** so I don't forget it.
5. Click "**Add your first data source**"
6. Click **InfluxDB**
7. **Add the following values:**

|                      |                                                             |
|----------------------|-------------------------------------------------------------|
| Name:                | InfluxDB-1 (Default)                                        |
| Query Language:      | InfluxQL (Default)                                          |
| URL:                 | <a href="http://10.11.1.19:8086">http://10.11.1.19:8086</a> |
| Basic auth:          | Disabled (Default)                                          |
| TLS Client Auth:     | Disabled (Default)                                          |
| Skip TLS Verify:     | Disabled (Default)                                          |
| Forward OAuth:       | Disabled (Default)                                          |
| Custom HTTP Headers: | NONE                                                        |
| Database:            | scaleout                                                    |
| User:                | user                                                        |
| Password:            | password                                                    |
| HTTP Method:         | GET                                                         |
| Min Time Interval:   | 1s (for 1 second)                                           |

Max series: [Blank](#) (Default)

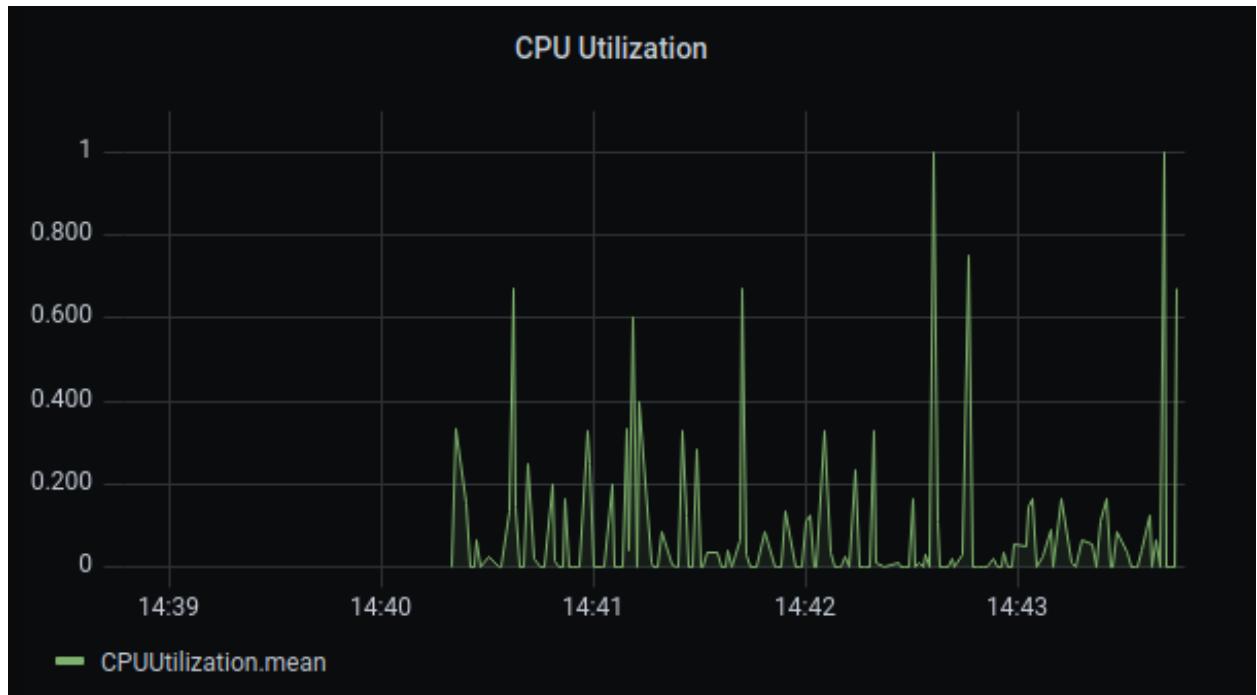
Press **Save & Test**.

Your should see:

datasource is working. 10 measurements found

8. Click home.
9. Click **COMPLETE** -> Create your first dashboard.
10. Click Add visualization.
11. Chose the Data source of **InfluxDB-1**.
12. Under Query near the bottom of the screen, chose the "select measurement" dropdown -> **CPUUtilization**
13. In the Panel Settings on the right, give the panel a name like "**CPU Utilization**"
14. Chose the time series to "**Last 5 minutes**".
15. Click **Apply**.

You will see a graph that looks like this:



16. If you add all the visualizations (there are only 10 schema objects tracked in InfluxDB) you will see:



## Cleanup

1. Reset the Docker Slurm cluster environment back to its original, pristine state by exiting the login Docker container back to the Ubuntu AWS prompt. And as **ubuntu**:  
`cd ~/docker-scale-out`

```
make clean && make
```

## Slurm Triggers Discussion

Slurm triggers can be used to evaluate events such as a node failing, a job reaching its time limit or a job terminating. These events can cause actions such as the execution of an arbitrary script. Typical uses include notifying system administrators of node failures and gracefully terminating a job when it's time limit is approaching. A hostlist expression for the nodelist or job ID is passed as an argument to the program.

Trigger events are not processed instantly, but a check is performed for trigger events on a periodic basis (currently every 15 seconds). Any trigger events which occur within that interval will be compared against the trigger programs set at the end of the time interval. The trigger program will be executed once for any event occurring in that interval. The record of those events (e.g. nodes which went DOWN in the previous 15 seconds) will then be cleared. The trigger program must set a new trigger before the end of the next interval to ensure that no trigger events are missed OR the trigger must be created with an argument of "--flags=PERM". If desired, multiple trigger programs can be set for the same event.

**IMPORTANT NOTE:** The `trigger` command can only set triggers if run by the user SlurmUser unless SlurmUser is configured as user root. This is required for the slurmctld daemon to set the appropriate user and group IDs for the executed program. Also note that the trigger program is executed on the same node that the slurmctld daemon uses rather than some allocated compute node. To check the value of SlurmUser, run the command:

```
scontrol show config | grep SlurmUser
```

`trigger` is used to set, get or clear Slurm trigger information. The following are the valid command-line switches to `trigger`:

### Arguments:

#### **-a, --primary\_slurmctld\_failure**

Trigger an event when the primary slurmctld fails.

#### **-A, --primary\_slurmctld\_resumed\_operation**

Trigger an event when the primary slurmctld resuming operation after failure.

#### **-b, --primary\_slurmctld\_resumed\_control**

Trigger an event when primary slurmctld resumes control.

#### **-B, --backup\_slurmctld\_failure**

Trigger an event when the backup slurmctld fails.

#### **-c, --backup\_slurmctld\_resumed\_operation**

Trigger an event when the backup slurmctld resumes operation after failure.

#### **-C, --backup\_slurmctld\_assumed\_control**

Trigger event when backup slurmctld assumes control.

#### **--burst\_buffer**

Trigger event when burst buffer error occurs.

#### **--clear**

Clear or delete a previously defined event trigger. The --id, --jobid or --user option must be specified to identify the trigger(s) to be cleared. Only user root or the trigger's creator can delete a trigger.

**-d, --down**

Trigger an event if the specified node goes into a DOWN state.

**-D, --drained**

Trigger an event if the specified node goes into a DRAINED state.

**--draining**

Trigger an event if the specified node goes into a DRAINING state

**-e, --primary\_slurmctld\_acct\_buffer\_full**

Trigger an event when primary slurmctld accounting buffer is full.

**-F, --fail**

Trigger an event if the specified node goes into a FAILING state.

**-f, --fini**

Trigger an event when the specified job completes execution.

**--flags=type** (e.g. --flags=PERM)

Associate flags with the reservation. Multiple flags should be comma separated. Valid flags include:

PERM -- Make the trigger permanent. Do not purge it after the event occurs.

**--front\_end**

Trigger events based upon changes in state of front end nodes rather than compute nodes. Applies to ALPS architectures only, where the slurmd daemon executes on front end nodes rather than the nodes. Use this option with either the --up or --down option.

**-g, --primary\_slurmdbd\_failure**

Trigger an event when the primary slurmdbd fails. The trigger is launched by slurmctld in the occasions where it tries to connect to slurmdbd, but receives no response on the socket.

**-G, --primary\_slurmdbd\_resumed\_operation**

Trigger an event when the primary slurmdbd resumes operation after failure. This event is triggered opening the connection from slurmctld to slurmdbd results in a response. It can happen also in situations, periodically every 15 seconds when checking the connection status, when saving state, agent queue is filling, and so on.

**--get** Show registered event triggers. Options can be used for filtering purposes.**-h, --primary\_database\_failure**

Trigger an event when the primary database fails. This event is triggered when the accounting plugin to open a connection with mysql and it fails and the slurmctld needs the database for some operations.

**-H, --primary\_database\_resumed\_operation**

Trigger an event when the primary database resumes operation after failure. It happens when connection to mysql from the accounting plugin is restored.

**-i, --id=id**

Trigger ID number.

**-I, --idle**

Trigger an event if the specified node remains in an IDLE state for at least the time period specified by the --offset option. This can be useful to hibernate a node that remains idle, thus reducing power consumption.

**-j, --jobid=id**

Job ID of interest. NOTE: The --jobid option can not be used in conjunction with the --node option. When the --jobid option is used in conjunction with the --up or --down option, all nodes allocated to that job will be the nodes used as a trigger event.

**-M, --clusters=<string>**

Clusters to issue commands to. Note that the SlurmDBD must be up for this option to work properly.

**-n, --node[=host]**

Host name(s) of interest. By default, all nodes associated with the job (if --jobid is specified) or on the system are considered for event triggers. NOTE: The --node option can not be used in conjunction with the --jobid option. When the --jobid option is used in conjunction with the --up, --down or --drained option, nodes allocated to that job will be considered the nodes used as a trigger event. Since this option's is optional, for proper parsing the single letter option must be followed immediately with the host name and not include a space between them. For example  
"-ntux" and not "-n tux".

**-N, --noheader**

Do not print the header when displaying a list of triggers.

**-o, --offset=seconds**

The specified action should follow the event by this time interval. Specify a negative value if action preceded the event. The default value is zero if no --offset option is specified. The resolution of this time is about 20 seconds, so to execute a script not less than five minutes prior to a job reaching its time limit specify --offset=320 (5 minutes plus 20 seconds).

**-p, --program=path**

Execute the program at the specified fully qualified pathname when the event occurs. You may change the path and include extra program arguments if desired. The program will be executed as the user

sets the trigger. If the program fails to terminate within 5 minutes, it will be killed along with its spawned processes.

**-Q, --quiet**

Do not report non-fatal errors. This can be useful to clear triggers which may have already been purged.

**-r, --reconfig**

Trigger an event when the system configuration changes. This is triggered when the slurmcld reads its configuration file or when a node state changes.

**--set** Register an event trigger based upon the supplied options. NOTE: An event is only triggered once. A event trigger must be set established for future events of the same type to be processed. Triggers can only be set if the command is run by the user SlurmUser unless SlurmUser is configured as user root.

**-t, --time**

Trigger an event when the specified job's time limit is reached. This must be used in conjunction with the `--jobid` option.

**-u, --up**

Trigger an event if the specified node is returned to service from a DOWN state.

**--user=user\_name\_or\_id**

Clear or get triggers created by the specified user. For example, a trigger created by user root for a job created by user adam could be cleared with an option `--user=root`. Specify either a user name or user ID.

**-v, --verbose**

Print detailed event logging. This includes time-stamps on data structures, record counts, etc.

**-V, --version**

Print version information and exit.

## Output Field Descriptions

**TRIG\_ID** Trigger ID number.

**RES\_TYPE** Resource type: job or node

**RES\_ID** Resource ID: job ID or host names or "\*" for any host

**TYPE** Trigger type: time or fini (for jobs only), down or up (for jobs or nodes), or drained, idle or reconfig (for nodes only)

**OFFSET** Time offset in seconds. Negative numbers indicated the action should occur before the event (if possible)

**USER** Name of the user requesting the action

**PROGRAM** Pathname of the program to execute when the event occurs

## Slurm System Triggers Discussion

### Examples

Execute the program `"/usr/sbin/primary_slurmctld_failure"` whenever the primary slurmctld fails.

```
> cat /usr/sbin/primary_slurmctld_failure
#!/bin/bash
Submit trigger for next primary slurmctld failure event
trigger --set --primary_slurmctld_failure \
--program=/usr/sbin/primary_slurmctld_failure
Notify the administrator of the failure using by e-mail
/bin/mail slurm_admin@site.com -s Primary_SLURMCTLD_FAILURE
> trigger --set --primary_slurmctld_failure \
--program=/usr/sbin/primary_slurmctld_failure
```

Execute the program "/usr/sbin/slurm\_admin\_notify" whenever any node in the cluster goes down. The subject line will include the node names which have entered the down state (passed as an argument to the script by Slurm).

```
> cat /usr/sbin/slurm_admin_notify
#!/bin/bash
Submit trigger for next event
trigger --set --node --down \
--program=/usr/sbin/slurm_admin_notify
Notify administrator using by e-mail
/bin/mail slurm_admin@site.com -s NodesDown:$*
> trigger --set --node --down --program=/usr/sbin/slurm_admin_notify
```

Execute the program "/usr/sbin/slurm\_suspend\_node" whenever any node in the cluster remains in the idle state for at least 600 seconds.

```
> trigger --set --node --idle --offset=600 \
--program=/usr/sbin/slurm_suspend_node
```

## Slurm Job Triggers Discussion

### Examples

Execute the program "/home/joe/clean\_up" when job 1234 is within 10 minutes of reaching its time limit.

```
> trigger --set --jobid=1234 --time --offset=-600 --program=/home/joe/clean_up
```

Execute the program "/home/joe/node\_died" when any node allocated to job 1234 enters the DOWN state.

```
> trigger --set --jobid=1234 --down --program=/home/joe/node_died
```

## Slurm Triggers Lab Exercises

In Slurm, it is possible to set triggers on a number of events. Some are job-driven, and some are event-driven. In this set of labs, you will configure one of each: Node Failure Trigger and a Job Trigger

### Exercise 1: Enable Node Failure Trigger

In this exercise, you will enable a node trigger failure that will send email notification to the root user when a node has reached the state of "Down"

1. As root, create the file `/etc/slurm/slurm_admin_notify` and add the following:

```
#!/bin/bash
Submit trigger for next event
trigger --set --node --down --program=/etc/slurm/slurm_admin_notify --flags=PERM
Notify administrator using by e-mail
/bin/mail -s "Node $* is Down" root <<< "Node $* is Down, better do something about it"
```

2. Flag the script executable:

```
chmod +x /etc/slurm/slurm_admin_notify
```

3. Add the following to `/etc/slurm/slurm.conf`:

```
DebugFlags=Triggers
```

4. Change the `SlurmctldTimeout` and `SlurmdTimeout` from 30 to 1 in the `/etc/slurm/slurm.conf`:

```
SlurmctldTimeout=1
SlurmdTimeout=1
```

(The default of 5 minutes is perfect for production, but way too long for a training exercise.)

5. Reload the Slurm configuration:

```
scontrol reconfigure
```

6. Switch to the user "slurm" and start the trigger:

```
su - slurm
```

```
trigger --set --node --down --program=/etc/slurm/slurm_admin_notify
```

7. Exit back to root user:

```
exit
```

8. View the trigger by entering the following:

```
trigger --get
```

You should see:

```
[root@login slurm]# trigger --get
TRIG_ID RES_TYPE RES_ID TYPE OFFSET USER FLAGS PROGRAM
 1 node * down 0 slurm PERM /etc/slurm/slurm_admin_notify
```

9. Look at the log and see the trigger entries being added by the scheduler. In the Ubuntu host, run the following command (**the following is all one line**):

```
ssh mgmtnode cat /var/log/slurmctld.log | grep "`date '+%y-%m-%d'`" | grep -i --color=auto trigger
```

You should see something similar to this:

```
[2021-10-01T10:25:18.930] debug: create_mmap_buf: Failed to open file `/var/spool/slurm/statesave/trigger_state`, No such file or directory
[2021-10-01T10:25:18.930] error: Could not open trigger state file `/var/spool/slurm/statesave/trigger_state: No such file or directory
[2021-10-01T10:25:18.930] error: NOTE: Trying backup state save file. Triggers may be lost!
[2021-10-01T10:25:18.930] debug: create_mmap_buf: Failed to open file `/var/spool/slurm/statesave/trigger_state.old`, No such file or directory
[2021-10-01T10:25:18.930] No trigger state file `/var/spool/slurm/statesave/trigger_state.old` to recover
[2021-10-01T10:25:22.614] debug3: create_mmap_buf: loaded file `/var/spool/slurm/statesave/trigger_state` as buf_t
[2021-10-01T10:25:22.614] State of 0 triggers recovered
[2021-10-01T10:25:34.258] debug2: Processing RPC: REQUEST_TRIGGER_PULL from UID=1001
[2021-10-01T10:25:44.276] debug3: create_mmap_buf: loaded file `/var/spool/slurm/statesave/trigger_state` as buf_t
[2021-10-01T10:25:44.276] State of 0 triggers recovered
[2021-10-01T10:29:39.816] debug2: Processing RPC: REQUEST_TRIGGER_SET from UID=1001
[2021-10-01T10:29:39.817] trigger_set
[2021-10-01T10:29:39.817] trigger[0] 0 node * down 0 4294967294 /etc/slurm/slurm_admin_notify
[2021-10-01T10:29:45.701] debug2: Processing RPC: REQUEST_TRIGGER_GET from UID=0
[2021-10-01T10:29:45.701] trigger_get
[2021-10-01T10:29:45.701] Trigger has no entries
[2021-10-01T10:29:45.702] trigger_got
[2021-10-01T10:29:45.702] trigger[0] 1 node * down 0 1001 /etc/slurm/slurm_admin_notify
```

10. Now exit the cluster and as the ubuntu user, kill off one of the nodes:

```
docker-compose kill node07
```

11. After a moment, you should see the following in the log:

```
Feb 13 09:38:39 ubuntu acb448ebbd2b[858]: slurmctld: trigger[0] 2 node * down 0 1001 /etc/slurm/slurm_admin_notify
```

12. Docker attach back into the login node (as root) and check the mail:

```
docker-compose exec login bash
mail
```

You should see:

```
[root@login ~]# mail
Heirloom Mail version 12.5 7/5/10. Type ? for help.
"/var/spool/mail/root": 1 message 1 unread
>U 1 slurm@cluster Thu Feb 13 17:38 19/566 "Node node07 is
Down"
&
```

13. Tap Enter read the email and it should show:

```
& 1
Message 1:
From slurm@cluster.localdomain Tue Feb 11 21:43:27 2020
Return-Path: <slurm@cluster.localdomain>
X-Original-To: root
Delivered-To: root@login.localdomain
From: slurm@cluster
Date: Tue, 11 Feb 2020 21:43:02 +0000
To: root
Subject: Node node07 is Down
User-Agent: Heirloom mailx 12.5 7/5/10
Content-Type: text/plain; charset=us-ascii
Status: RO

Node node07 is Down, better do something about it
```

## Exercise 2: Enable a Job Trigger

In this exercise, you will enable a job trigger that will fire off an email (through a script) when a job has less than 1 minutes remaining.

- As root, create the file `/etc/slurm/trigger_script_1_minute_left` and add the following:

```
#!/bin/bash
Notify administrator using by e-mail that a job only has a minute left
/bin/mail -s "Job:$* only has a minute left" root <<< "Job:$* has only 1
minute left"
```

- Flag it executable:

```
chmod +x /etc/slurm/trigger_script_1_minute_left
```

3. Switch to the **fred user** and submit a 5-minute job:

```
sbatch -N1 -t300 --wrap="sleep 300"
```

Note the JOBID

4. Switch to the **slurm user** and set the trigger (**the following is all one line**):

```
trigger --set --jobid=<YOUR JOBID> --time --offset=-60 --
program=/etc/slurm/trigger_script_1_minute_left
```

5. Validate the job trigger was set:

```
trigger --get
```

Should show the following:

| TRIG_ID | RES_TYPE | RES_ID | TYPE | OFFSET | USER  | FLAGS | PROGRAM                                 |
|---------|----------|--------|------|--------|-------|-------|-----------------------------------------|
| 2       | node     | *      | down | 0      | slurm | PERM  | /etc/slurm/slurm_admin_notify           |
| 3       | job      | 2      | time | -60    | slurm |       | /etc/slurm/trigger_script_1_minute_left |

6. Watch the job with **squeue**.

**NOTE:** When it has 1 minute left, you should see the trigger fire and send the email notification. And, since this trigger does not have the flags=PERM setting, it is a fire-once-only type of trigger. So the trigger --get should not see it any more

## Cleanup

1. Reset the Docker Slurm cluster environment back to its original, pristine state by exiting the login Docker container back to the Ubuntu AWS prompt. And **as the ubuntu user**:

```
cd ~/docker-scale-out
make clean && make
```

## **Break-Fix Labs**

These labs will break some aspect of the controller. You will need to find out what broke and how to fix it.

## Break-Fix Lab Exercises

In these exercises you will attempt to submit workload, it will fail. Your mission, Jim, should you choose to accept it, will be to figure out why your job fails and fix it. As always, should you or any of your IM Force be caught or killed, the Secretary will disavow any knowledge of your actions. This tape will self-destruct in ten seconds. Good luck, Jim...

### Exercise 1: Why won't my job run?

It would appear that the entire cluster is available right now

1. **As root**, execute the following script:

```
bash /lab_scripts/demo1.sh
```

**NOTE:** This will set up the lab environment for troubleshooting the first scenario.

2. **As fred**, confirm the cluster is running jobs taking up 9 nodes and 36 cores by running sinfo -Nel:

```
sinfo -Nel -pdebug
```

It should show:

| Fri Jan 14 15:13:17 2022 |       |           |           |      |       |        |          |        |          |        |
|--------------------------|-------|-----------|-----------|------|-------|--------|----------|--------|----------|--------|
| NODELIST                 | NODES | PARTITION | STATE     | CPUS | S:C:T | MEMORY | TMP_DISK | WEIGHT | AVAIL_FE | REASON |
| node00                   | 1     | debug*    | allocated | 4    | 1:4:1 | 15979  | 0        | 1      | (null)   | none   |
| node01                   | 1     | debug*    | allocated | 4    | 1:4:1 | 15979  | 0        | 1      | (null)   | none   |
| node02                   | 1     | debug*    | allocated | 4    | 1:4:1 | 15979  | 0        | 1      | (null)   | none   |
| node03                   | 1     | debug*    | allocated | 4    | 1:4:1 | 15979  | 0        | 1      | (null)   | none   |
| node04                   | 1     | debug*    | allocated | 4    | 1:4:1 | 15979  | 0        | 1      | (null)   | none   |
| node05                   | 1     | debug*    | allocated | 4    | 1:4:1 | 15979  | 0        | 1      | (null)   | none   |
| node06                   | 1     | debug*    | allocated | 4    | 1:4:1 | 15979  | 0        | 1      | (null)   | none   |
| node07                   | 1     | debug*    | allocated | 4    | 1:4:1 | 15979  | 0        | 1      | (null)   | none   |
| node08                   | 1     | debug*    | allocated | 4    | 1:4:1 | 15979  | 0        | 1      | (null)   | none   |
| node09                   | 1     | debug*    | idle      | 4    | 1:4:1 | 15979  | 0        | 1      | (null)   | none   |

3. Check the number of free cores by running the sinfo command against the debug partition with special formatting arguments:

```
sinfo -pdebug -o "%n %e %m %a %c %C" | column -t
```

It should show:

| HOSTNAMES | FREE_MEM. | MEMORY | AVAIL | CPUS | CPUS (A/I/O/T) |
|-----------|-----------|--------|-------|------|----------------|
| node00.   | 2140      | 15979  | up    | 4    | 4/0/0/4        |
| node01    | 2140      | 15979  | up    | 4    | 4/0/0/4        |
| node02    | 2140      | 15979  | up    | 4    | 4/0/0/4        |
| node03    | 2140      | 15979  | up    | 4    | 4/0/0/4        |
| node04    | 2140      | 15979  | up    | 4    | 4/0/0/4        |
| node05    | 2140      | 15979  | up    | 4    | 4/0/0/4        |
| node06    | 2140      | 15979  | up    | 4    | 4/0/0/4        |
| node07    | 2140      | 15979  | up    | 4    | 4/0/0/4        |
| node08    | 2140      | 15979  | up    | 4    | 4/0/0/4        |
| node09    | 2140      | 15979  | up    | 4    | 0/4/0/4        |

**Note:** node09 has 4 free cores

4. Submit 3 jobs:

```
sbatch -N1 -n1 --wrap "sleep 1000"
sbatch -N1 -n1 --wrap "sleep 1000"
sbatch -N1 -n1 --wrap "sleep 1000"
```

5. Check the cluster availability:

```
sinfo -Nel -p debug ; squeue -la ; sinfo -p debug -o "%n %e %m %a %c %C"
```

It should show:

| Fri Jan 14 15:18:28 2022                           |           |           |           |         |         |           |          |                  |          |        |  |  |  |  |  |
|----------------------------------------------------|-----------|-----------|-----------|---------|---------|-----------|----------|------------------|----------|--------|--|--|--|--|--|
| NODELIST                                           | NODES     | PARTITION | STATE     | CPUS    | S:C:T   | MEMORY    | TMP_DISK | WEIGHT           | AVAIL_FE | REASON |  |  |  |  |  |
| node00                                             | 1         | debug*    | allocated | 4       | 1:4:1   | 15979     | 0        | 1                | (null)   | none   |  |  |  |  |  |
| node01                                             | 1         | debug*    | allocated | 4       | 1:4:1   | 15979     | 0        | 1                | (null)   | none   |  |  |  |  |  |
| node02                                             | 1         | debug*    | allocated | 4       | 1:4:1   | 15979     | 0        | 1                | (null)   | none   |  |  |  |  |  |
| node03                                             | 1         | debug*    | allocated | 4       | 1:4:1   | 15979     | 0        | 1                | (null)   | none   |  |  |  |  |  |
| node04                                             | 1         | debug*    | allocated | 4       | 1:4:1   | 15979     | 0        | 1                | (null)   | none   |  |  |  |  |  |
| node05                                             | 1         | debug*    | allocated | 4       | 1:4:1   | 15979     | 0        | 1                | (null)   | none   |  |  |  |  |  |
| node06                                             | 1         | debug*    | allocated | 4       | 1:4:1   | 15979     | 0        | 1                | (null)   | none   |  |  |  |  |  |
| node07                                             | 1         | debug*    | allocated | 4       | 1:4:1   | 15979     | 0        | 1                | (null)   | none   |  |  |  |  |  |
| node08                                             | 1         | debug*    | allocated | 4       | 1:4:1   | 15979     | 0        | 1                | (null)   | none   |  |  |  |  |  |
| node09                                             | 1         | debug*    | mixed     | 4       | 1:4:1   | 15979     | 0        | 1                | (null)   | none   |  |  |  |  |  |
| Fri Jan 14 15:18:28 2022                           |           |           |           |         |         |           |          |                  |          |        |  |  |  |  |  |
| JOBID                                              | PARTITION | NAME      | USER      | STATE   | TIME    | TIME_LIMI | NODES    | NODELIST(REASON) |          |        |  |  |  |  |  |
| 21                                                 | debug     | wrap      | fred      | RUNNING | 5:18    | UNLIMITED | 9        | node[00-08]      |          |        |  |  |  |  |  |
| 22                                                 | debug     | wrap      | fred      | RUNNING | 2:03    | UNLIMITED | 1        | node09           |          |        |  |  |  |  |  |
| 23                                                 | debug     | wrap      | fred      | RUNNING | 0:03    | UNLIMITED | 1        | node09           |          |        |  |  |  |  |  |
| 24                                                 | debug     | wrap      | fred      | RUNNING | 0:00    | UNLIMITED | 1        | node09           |          |        |  |  |  |  |  |
| HOSTNAMES FREE_MEM MEMORY AVAIL CPUS CPUS(A/I/O/T) |           |           |           |         |         |           |          |                  |          |        |  |  |  |  |  |
| <u>node09 2130 15979 up 4 3/1/0/4</u>              |           |           |           |         |         |           |          |                  |          |        |  |  |  |  |  |
| node00                                             | 2130      | 15979     | up        | 4       | 4/0/0/4 |           |          |                  |          |        |  |  |  |  |  |
| node01                                             | 2130      | 15979     | up        | 4       | 4/0/0/4 |           |          |                  |          |        |  |  |  |  |  |
| node02                                             | 2130      | 15979     | up        | 4       | 4/0/0/4 |           |          |                  |          |        |  |  |  |  |  |
| node03                                             | 2130      | 15979     | up        | 4       | 4/0/0/4 |           |          |                  |          |        |  |  |  |  |  |
| node04                                             | 2130      | 15979     | up        | 4       | 4/0/0/4 |           |          |                  |          |        |  |  |  |  |  |
| node05                                             | 2130      | 15979     | up        | 4       | 4/0/0/4 |           |          |                  |          |        |  |  |  |  |  |
| node06                                             | 2130      | 15979     | up        | 4       | 4/0/0/4 |           |          |                  |          |        |  |  |  |  |  |
| node07                                             | 2130      | 15979     | up        | 4       | 4/0/0/4 |           |          |                  |          |        |  |  |  |  |  |

**Note:** With 1 core free, you should be able to run another job

6. Run the last job:

```
sbatch -N1 -n1 --wrap "sleep 1000"
```

7. Check the cluster again:

```
Fri Jan 14 15:22:47 2022
NODELIST NODES PARTITION STATE CPUS S:C:T MEMORY TMP_DISK WEIGHT AVAIL_FE REASON
node00 1 debug* allocated 4 1:4:1 15979 0 1 (null) none
node01 1 debug* allocated 4 1:4:1 15979 0 1 (null) none
node02 1 debug* allocated 4 1:4:1 15979 0 1 (null) none
node03 1 debug* allocated 4 1:4:1 15979 0 1 (null) none
node04 1 debug* allocated 4 1:4:1 15979 0 1 (null) none
node05 1 debug* allocated 4 1:4:1 15979 0 1 (null) none
node06 1 debug* allocated 4 1:4:1 15979 0 1 (null) none
node07 1 debug* allocated 4 1:4:1 15979 0 1 (null) none
node08 1 debug* allocated 4 1:4:1 15979 0 1 (null) none
node09 1 debug* mixed 4 1:4:1 15979 0 1 (null) none

Fri Jan 14 15:22:47 2022
JOBID PARTITION NAME USER STATE TIME TIME_LIMI NODES NODELIST(REASON)
25 debug wrap fred PENDING 0:00 UNLIMITED 1 (Resources)
21 debug wrap fred RUNNING 9:37 UNLIMITED 9 node[00-08]
22 debug wrap fred RUNNING 6:22 UNLIMITED 1 node09
23 debug wrap fred RUNNING 4:22 UNLIMITED 1 node09
24 debug wrap fred RUNNING 4:19 UNLIMITED 1 node09

HOSTNAMES FREE_MEMORY_AVAIL CPUS CPUS(A/I/O/T)
node09 2108 15979 up 4 3/1/0/4
node00 2107 15979 up 4 4/0/0/4
node01 2107 15979 up 4 4/0/0/4
node02 2107 15979 up 4 4/0/0/4
node03 2108 15979 up 4 4/0/0/4
node04 2107 15979 up 4 4/0/0/4
node05 2107 15979 up 4 4/0/0/4
node06 2107 15979 up 4 4/0/0/4
node07 2107 15979 up 4 4/0/0/4
node08 2107 15979 up 4 4/0/0/4
```

8. Why isn't the last job running?

## Cleanup

- Reset the Docker Slurm cluster environment back to its original, pristine state by exiting the login Docker container back to the Ubuntu AWS prompt. And as **ubuntu**:

```
cd ~/docker-scale-out
make clean && make
```

## Exercise 2: Why aren't the job details accessible via sacct?

In this exercise, you will try to determine why jobs can be submitted, and checkjob'd, and running, but their details are not shown in sacct

- As **root**, execute the following script:

```
bash /lab_scripts/demo2.sh
```

**NOTE:** This will set up the lab environment for troubleshooting the second scenario.

- As **fred**, submit a few jobs:

```
srun -N1 hostname
srun -N1 hostname
srun -N1 hostname
srun -N1 hostname
```

Notice that they all get responses and serviced nicely.

3. Run sdiag... and look for the **DBD Agent queue size**.  
`sdiag`
4. **Look for the DBD agent queue size.**  
Is the DBD Agent queue growing with every job submission?
5. Run sacct, does it return any values?  
`sacct`

**How do you fix this?**

## Cleanup

1. Reset the Docker Slurm cluster environment back to its original, pristine state by exiting the login Docker container back to the Ubuntu AWS prompt. And **as ubuntu**:  
`cd ~/docker-scale-out`  
`make clean && make`

## Exercise 3: Why aren't my user Access Control rights being honored?

In Slurm, it's a simple matter of assigning rights in the database. Rights such as maxjob, maxsubmitjob, maxtres(cpu=XX), for example. In this exercise, you will assign some Access Control rights and see if they are honored. And if not, why.

1. **As root**, execute the following script:  
`bash /lab_scripts/demo3.sh`

**NOTE:** This will set up the lab environment for troubleshooting the third scenario.

2. Create an access control permission for Bambam so that he can only submit 2 jobs at a time to the cluster.  
The third would go rejected:  
`sacctmgr -i update user where name=bambam set maxsubmit=2`
3. **Submit several jobs as bambam.**

Does the third get rejected?

4. **How do you fix this?**

## Cleanup

1. Reset the Docker Slurm cluster environment back to its original, pristine state by exiting the login Docker container back to the Ubuntu AWS prompt. And **as ubuntu**:

```
cd ~/docker-scale-out
make clean && make
```

## Exercise 4: Why don't my jobs pack?

In Slurm, it is possible to submit a job to the cluster and have that job use resources already allocated to another job, if there is room. In this exercise, submit a couple of 2-task jobs and see why they land on nodes that have nothing running on them. **NOTE:** This lab does NOT require a lab setup script to be run prior to performing the steps below.

1. As **Fred**, submit 2 jobs that will be allocated 2 tasks each:

```
sbatch -n2 -t4 -J"Shared Node Job" --wrap "srun sleep 300"
sbatch -n2 -t4 -J"Shared Node Job" --wrap "srun sleep 300"
```

2. Check the job queue:

```
sinfo -Nel -pdebug ; squeue -la
```

It should show:

```
Thu Jun 02 09:05:12 2022
NODELIST NODES PARTITION STATE CPUS S:C:T MEMORY TMP_DISK WEIGHT AVAIL_FE REASON
node00 1 debug* mixed 4 1:4:1 15979 0 1 (null) none
node01 1 debug* mixed 4 1:4:1 15979 0 1 (null) none
node02 1 debug* idle 4 1:4:1 15979 0 1 (null) none
node03 1 debug* idle 4 1:4:1 15979 0 1 (null) none
node04 1 debug* idle 4 1:4:1 15979 0 1 (null) none
node05 1 debug* idle 4 1:4:1 15979 0 1 (null) none
node06 1 debug* idle 4 1:4:1 15979 0 1 (null) none
node07 1 debug* idle 4 1:4:1 15979 0 1 (null) none
node08 1 debug* idle 4 1:4:1 15979 0 1 (null) none
node09 1 debug* idle 4 1:4:1 15979 0 1 (null) none
Thu Jun 02 09:05:12 2022
JOBID PARTITION NAME USER STATE TIME TIME_LIMI NODES NODELIST(REASON)
1 debug Shared N fred RUNNING 1:36 4:00 1 node00
2 debug Shared N fred RUNNING 1:33 4:00 1 node01
```

Why doesn't Slurm pack job2 on node00?

3. Try submitting another 2 jobs with a **--mem=1000** flag.

Does that help? Why?

## Cleanup

1. Reset the Docker Slurm cluster environment back to its original, pristine state by exiting the login Docker container back to the Ubuntu AWS prompt. And as **ubuntu**:

```
cd ~/docker-scale-out
make clean && make
```

## Running Jobs in Containers

Containers are being adopted in HPC workloads. Containers rely on existing kernel features to allow greater user control over what applications see and can interact with at any given time. For HPC Workloads, these are usually restricted to the mount namespace. Slurm natively supports the requesting of unprivileged OCI Containers for jobs and steps.

### Prerequisites

The host kernel must be configured to allow user land containers:

```
$ sudo sysctl -w kernel.unprivileged_userns_clone=1
```

Docker also provides a tool to verify the kernel configuration:

```
$ dockerd-rootless-setuptool.sh check -force
[INFO] Requirements are satisfied
```

### Required software:

- Fully functional OCI runtime. It needs to be able to run outside of Slurm first.
- Fully functional OCI bundle generation tools. Slurm requires OCI Container compliant bundles for jobs.

To see the entire supported software stack for OCI compliance on Slurm, please visit the following web site:

<https://slurm.schedmd.com/containers.html>

Slurm provides for different name spaces and containerized workload as part of it's OCI compliance. One of the more popular namespace container systems is Enroot, sponsored by NVIDIA. This solution is hosted in the gitlabs project which utilizes Slurm's SPANK plugin mechanism. While not difficult to install/configure, it's use cases are huge. One of the more popular integrations with Slurm is by means of pyxis.

## Slurm, SPANK, and pyxis Container Lab Exercises

The training environment is entirely configured using Docker containers. The added functionality of containerized workload within the Docker containers is meant as an example, not suggested for production environments.

### Exercise 1: Install enroot

Enroot is a container tool which provides container/OS images into unprivileged sandboxes. It works very nicely with well-known container image formats like Docker and Pyxis. The enroot utility needs to be installed on each of the compute nodes (nodes00 through node09) since that is where Slurm will launch the containerized workloads.

1. Log into the **login container, as root**.
2. Install epel-release into the compute nodes, which have a cross-mounted filesystems, by running the following command **on the login node**:  
`pdsh "sudo yum install -y epel-release"`
3. Run the following command to install enroot into the node00 – node09 containers:

```
pdsh 'sudo yum localinstall -y \
https://github.com/NVIDIA/enroot/releases/download/v3.4.1/enroot{,\n+caps}-3.4.1-1.el8.x86_64.rpm'
```

### Exercise 2: Install pyxis

Pyxis is a Slurm SPANK plugin which enables unprivileged cluster users to run containerized tasks through the Slurm `srun` command. In this exercise, you will install pyxis into each container node.

1. Git clone the pyxis repository on each of the compute nodes:  
`pdsh "git clone https://github.com/NVIDIA/pyxis.git" 2>/dev/null`
2. Build pyxis on each of the compute nodes:  
~~`pdsh "cd pyxis; make"`~~
3. Install pyxis on each of the compute nodes:  
~~`pdsh "cd pyxis ; make install"`~~
4. Restart the slurmd's on each of the compute nodes:  
~~`pdsh systemctl restart slurmd`~~

### Exercise 3: Run a job in a container

Now that enroot and pyxis are installed, you can run containerized workload through `srun`.

1. As **fred**, submit a job which will query the container OS and report the version:  
`srun --container-image=centos grep PRETTY /etc/os-release`

You should see:

```
pyxis: importing docker image: centos
pyxis: imported docker image: centos
PRETTY_NAME="CentOS Linux 8"
```

2. Try running an application (binary) through pyxis:

```
srun -N1 -n4 --container-image=centos hostname
```

You should see:

```
pyxis: imported docker image: centos
node00
node00
node00
node00
```

Pretty cool

## Cleanup

1. Reset the Docker Slurm cluster environment back to its original, pristine state by exiting the login Docker container back to the Ubuntu AWS prompt. And **as ubuntu**:

```
cd ~/docker-scale-out
make clean && make
```

## License Management in Slurm

Slurm can help with software license management by assigning available licenses to jobs at scheduling time. If the licenses are not available, jobs are kept pending until licenses become available. Licenses in Slurm are essentially shared resources, meaning configured resources that are not tied to a specific host but are associated with the entire cluster.

Licenses in Slurm can be configured in two ways:

- **Local Licenses:** Local licenses are local to the cluster using the *slurm.conf* in which they are configured.
- **Remote Licenses:** Remote licenses are served by the database and are configured using the *sacctmgr* command. Remote licenses are dynamic in nature as upon running the *sacctmgr* command, the *slurmdbd* updates all clusters the licenses are assigned to.

### Local Licenses

Local licenses are defined in the *slurm.conf* using the *Licenses* option.

*slurm.conf*:

```
Licenses=fluent:30,ansys:100
```

Configured licenses can be viewed using the *scontrol* command.

```
$ scontrol show lic
LicenseName=ansys
 Total=100 Used=0 Free=100 Remote=no
LicenseName=fluent
 Total=30 Used=0 Free=30 Remote=no
```

Requesting licenses is done by using the *-L*, or *--licenses*, submission option.

```
$ sbatch -L ansys:2 script.sh
Submitted batch job 5212

$ scontrol show lic
LicenseName=ansys
 Total=100 Used=2 Free=98 Remote=no
LicenseName=fluent
 Total=30 Used=0 Free=30 Remote=no
```

## Remote Licenses

### USE CASE

A site has two license servers, one serves 100 Nastran licenses provided by FlexNet and the other serves 50 Matlab licenses from Reprise License Management. The site has two clusters named "fluid" and "pdf" dedicated to run

simulation jobs using both products. The managers want to split the number of Nastran licenses equally between clusters, but assign 70% of the Matlab licenses to cluster "pdf" and the remaining 30% to cluster "fluid".

## CONFIGURING SLURM FOR THE USE CASE

Here we assume that both clusters have been configured correctly in the *slurmdbd* using the *sacctmgr* command

```
$ sacctmgr show clusters format=cluster,controlhost
 Cluster ControlHost

 fluid 143.11.1.3
 pdf 144.12.3.2
```

The licenses are added using the *sacctmgr* command, specifying the total count of licenses and the percentage that should be allocated to each cluster. This can be done either in one step or through a multi-step process.

One step:

```
$ sacctmgr add resource name=nastran cluster=fluid,pdf \
 count=100 allowed=50 server=flex_host servertype=flexlm type=license
Adding Resource(s)
nastran@flex_host
 Cluster - fluid 50
 Cluster - pdf 50
Settings
 Name = Nastran
 Server = flex_host
 Description = Nastran
 ServerType = flexlm
 Count = 100
 Type = License
```

Multi-step:

```
$ sacctmgr add resource name=matlab count=50 server=rlm_host servertype=rlm type=license
Adding Resource(s)
matlab@rlm_host
Settings
 Name = matlab
 Server = rlm_host
 Description = matlab
 ServerType = rlm
 Count = 50
 Type = License

$ sacctmgr add resource name=matlab server=rlm_host cluster=pdf allowed=70
Adding Resource(s)
matlab@rlm_host
 Cluster - pdf 70
Settings
 Name = matlab
 Server = rlm_host
 Count = 50
 LastConsumed = 0
```

```

Flags = (null)
Type = License

$ sacctmgr add resource name=matlab server=rlm_host cluster=fluid allowed=30
Adding Resource(s)
matlab@rlm_host
Cluster - fluid 30
Settings
Name = matlab
Server = rlm_host
Count = 50
LastConsumed = 0
Flags = (null)
Type = License

```

The *sacctmgr* command will now display the grand total of licenses.

```

$ sacctmgr show resource
 Name Server Type Count LastConsumed Allocated ServerType Flags
-----+-----+-----+-----+-----+-----+-----+-----+
 nastran flex_host License 100 0 100 flexlm
 matlab rlm_host License 50 0 100 rlm
$ sacctmgr show resource withclusters
 Name Server Type Count LastConsumed Allocated ServerType Cluster Allowed Flags
-----+-----+-----+-----+-----+-----+-----+-----+-----+
 nastran flex_host License 100 0 100 flexlm fluid 50
 nastran flex_host License 100 0 100 flexlm pdf 50
 matlab rlm_host License 50 0 100 rlm fluid 30
 matlab rlm_host License 50 0 100 rlm pdf 70

```

The configured licenses are now visible on both clusters using the *scontrol* command.

```

On cluster "pdf":
$ scontrol show lic
LicenseName=matlab@rlm_host
 Total=35 Used=0 Free=35 Reserved=0 Remote=yes
 LastConsumed=0 LastDeficit=0 LastUpdate=2023-02-28T17:01:44
LicenseName=nastran@flex_host
 Total=50 Used=0 Free=50 Reserved=0 Remote=yes
 LastConsumed=0 LastDeficit=0 LastUpdate=2023-02-28T17:01:44

On cluster "fluid":
$ scontrol show lic
LicenseName=matlab@rlm_host
 Total=15 Used=0 Free=15 Reserved=0 Remote=yes
 LastConsumed=0 LastDeficit=0 LastUpdate=2023-02-28T17:01:44
LicenseName=nastran@flex_host
 Total=50 Used=0 Free=50 Reserved=0 Remote=yes
 LastConsumed=0 LastDeficit=0 LastUpdate=2023-02-28T17:01:44

```

When submitting jobs to remote licenses, the name and server must be used.

```
$ sbatch -L nastran@flex_host script.sh
Submitted batch job 5172
```

License percentages and counts can be modified as shown below:

```
$ sacctmgr modify resource name=matlab server=rlm_host set count=200
Modified server resource ...
matlab@rlm_host
Cluster - fluid - matlab@rlm_host
Cluster - pdf - matlab@rlm_host

$ sacctmgr modify resource name=matlab server=rlm_host \
cluster=pdf set allowed=60
Modified server resource ...
Cluster - pdf - matlab@rlm_host

$ sacctmgr show resource withclusters
 Name Server Type Count LastConsumed Allocated ServerType Cluster Allowed Flags
-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
nastran flex_host License 100 0 100 flexlm fluid 50
nastran flex_host License 100 0 100 flexlm pdf 50
matlab rlm_host License 200 0 90 rlm fluid 30
matlab rlm_host License 200 0 90 rlm pdf 60
```

Licenses can be deleted either on the cluster or all together as shown:

```
$ sacctmgr delete resource where name=matlab server=rlm_host cluster=fluid
Deleting resource(s)...
Deleting resource(s)...
Cluster - fluid - matlab@rlm_host

$ sacctmgr delete resource where name=nastran server=flex_host
Deleting resource(s)...
nastran@flex_host
Cluster - fluid - nastran@flex_host"
Cluster - pdf - nastran@flex_host

$ sacctmgr show resource withclusters
 Name Server Type Count LastConsumed Allocated ServerType Cluster Allowed Flags
-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
matlab rlm_host License 200 0 60 rlm pdf 60
```

Starting with Slurm 23.02, a new *Absolute* flag is available that indicates the license allowed values for each cluster are to be treated as absolute license counts rather than percentages.

Some brief examples of license management using this flag.

```
$ sacctmgr -i add resource name=deluxe cluster=fluid,pdf count=150 allowed=70 server=flex_host servertype=flexlm flags=absolute
Adding Resource(s)
deluxe@flex_host
Cluster - fluid 70
Cluster - pdf 70
Settings
Name = deluxe
Server = flex_host
Description = deluxe
ServerType = flexlm
Count = 150
Flags = Absolute
Type = Unknown
```

```

$ sacctmgr show resource withclusters
 Name Server Type Count LastConsumed Allocated ServerType Cluster Allowed
Flags

 deluxe flex_host License 150 0 140 flexlm fluid 70
 deluxe flex_host License 150 0 140 flexlm pdf 70 Absolute
Absolute

$ sacctmgr -i update resource deluxe set allowed=25 where cluster=fluid
Modified server resource ...
Cluster - fluid - deluxe@flex_host

$ sacctmgr show resource withclusters
 Name Server Type Count LastConsumed Allocated ServerType Cluster Allowed
Flags

 deluxe flex_host License 150 0 95 flexlm fluid 25 Absolute
 deluxe flex_host License 150 0 95 flexlm pdf 70 Absolute
Absolute

```

This can also be established as the default for all newly created licenses by adding `AllResourcesAbsolute=yes` to `slurmdbd.conf` (and restarting SlurmDBD to make the change take effect).

## Dynamic Licenses

Starting with Slurm 23.02, the `LastConsumed` field for remote licenses is designed to be periodically updated with the active use count from a license server. An example script for FlexLM's lmstat command is provided below — similar scripts can be easily constructed for other license management stacks.

```

#!/bin/bash

set -euxo pipefail

LMSTAT=/opt/foobar/bin/lmstat
LICENSE=foobar

consumed=$(${LMSTAT} | grep "Users of ${LICENSE}" | sed "s/.Total of \([0-9]+\)\) licenses in use)/\1/")
sacctmgr -i update resource ${LICENSE} set lastconsumed=${consumed}

```

When the `LastConsumed` value is changed through `sacctmgr` an update is automatically pushed to the Slurm controllers. They will use this value to calculate a `LastDeficit` value — this value indicates how many licenses that have "gone missing" from the cluster's perspective and will need to be set aside temporarily.

E.g., on this cluster 100 "foobar" licenses are available, and we are allocating access to 80 of them on the "blackhole" cluster:

```

$ sacctmgr add resource foobar count=100 flags=absolute cluster=blackhole allowed=80
Adding Resource(s)
 foobar@slurmdb
 Cluster - blackhole 80
Settings
 Name = foobar
 Server = slurmdb
 Description = foobar

```

```

Count = 100
Flags = Absolute
Type = Unknown
Would you like to commit changes? (You have 30 seconds to decide)
(N/y): y
$ scontrol show license
LicenseName=foobar@slurmdb
 Total=80 Used=0 Free=80 Reserved=0 Remote=yes
 LastConsumed=0 LastDeficit=0 LastUpdate=2023-02-28T16:36:55

```

Now, our cron job comes in and updates the LastConsumed value to 30, while the cluster has yet to allocate any licenses to jobs:

```

$ sacctmgr -i update resource foobar set lastconsumed=30
Modified server resource ...
foobar@slurmdb
Cluster - blackhole - foobar@slurmdb
$ scontrol show license
LicenseName=foobar@slurmdb
 Total=80 Used=0 Free=70 Reserved=0 Remote=yes
 LastConsumed=30 LastDeficit=10 LastUpdate=2023-02-28T16:39:27

```

Note that the cluster has now calculated a deficit of 10 licenses, and has noticed that it should only schedule up to 70 licenses at the moment. The cluster knows that up to 20 licenses are reserved for other clusters or external use at the moment. However, since LastConsumed was set to 30 this implies an additional 10 licenses have "gone rogue" and their usage cannot be accounted for. Thus the cluster must not assign those to any pending jobs, as it's likely that the job would fail to acquire the desired licenses.

If a further update (likely driven through cron) now reduces the LastConsumed count to 10, the deficit is now considered to have disappeared, and the cluster will make all 80 assigned licenses available again:

```

$ sacctmgr -i update resource foobar set lastconsumed=20
Modified server resource ...
foobar@slurmdb
Cluster - blackhole - foobar@slurmdb
$ scontrol show license
LicenseName=foobar@slurmdb
 Total=80 Used=0 Free=80 Reserved=0 Remote=yes
 LastConsumed=20 LastDeficit=0 LastUpdate=2023-02-28T16:44:26

```

## A note on job preemption with licenses

Add new PreemptParameters=reclaim\_licenses option which will allow higher priority jobs to preempt jobs to free up used licenses. (This is only enabled for with PreemptModes of CANCEL and REQUEUE, as Slurm cannot guarantee suspended jobs will release licenses correctly.)

## Slurm Cloud Scheduling Guide

Slurm has the ability to support a cluster that grows and shrinks on demand, typically relying upon a service such as Amazon Elastic Computing Cloud (Amazon EC2), Google Cloud Platform or Microsoft Azure for resources. These resources can be combined with an existing cluster to process excess workload (cloud bursting) or it can operate as an independent self-contained cluster. Good responsiveness and throughput can be achieved while you only pay for the resources needed.

The rest of this document describes details about Slurm's infrastructure that can be used to support cloud scheduling.

Slurm's cloud scheduling logic relies heavily upon the existing power save logic. Review of Slurm's Power Saving Guide is strongly recommended. This logic initiates programs when nodes are required for use and another program when those nodes are no longer required. For cloud scheduling, these programs will need to provision the resources from the cloud and notify Slurm of the node's name and network address and later relinquish the nodes back to the cloud. Most of the Slurm changes to support cloud scheduling were changes to support node addressing that can change.

## Slurm Configuration

```
SLURM_RESUME_FILE=/proc/1647372/fd/7:
{
 "all_nodes_resume" : "cloud[1-3,7-8]",
 "jobs" : [
 {
 "extra" : "An arbitrary string from --extra",
 "features" : "c1,c2",
 "job_id" : 140814,
 "nodes_alloc" : "cloud[1-4]",
 "nodes_resume" : "cloud[1-3]",
 "oversubscribe" : "OK",
 "partition" : "cloud",
 "reservation" : "resv_1234"
 },
 {
 "extra" : null,
 }
]
}
```

```

 "features" : "c1,c2",
 "job_id" : 140815,
 "nodes_alloc" : "cloud[1-2]",
 "nodes_resume" : "cloud[1-2]",
 "oversubscribe" : "OK",
 "partition" : "cloud",
 "reservation" : null
 }
 {
 "extra" : null,
 "features" : null
 "job_id" : 140816,
 "nodes_alloc" : "cloud[7-8]",
 "nodes_resume" : "cloud[7-8]",
 "oversubscribe" : "NO",
 "partition" : "cloud_exclusive",
 "reservation" : null
 }
]
}

```

### **ResumeTimeout**

Maximum time permitted (in seconds) between when a node resume request is issued and when the node is actually available for use. Nodes which fail to respond in this time frame will be marked DOWN and the jobs scheduled on the node requeued. Nodes which reboot after this time frame will be marked DOWN with a reason of "Node unexpectedly rebooted." The default value is 60 seconds.

### **SlurmctldParameters=cloud\_dns**

By default, Slurm expects that the network addresses for cloud nodes won't be known until creation of the node and that Slurm will be notified of the node's address (e.g. scontrol update nodename=<name> nodeaddr=<addr>). Since Slurm communications rely on the node configuration found in the slurm.conf, Slurm will tell the client

command, after waiting for all nodes to boot, each node's IP address. However, in environments where the nodes are in DNS, this step can be avoided by configuring this option.

#### **SlurmctlParameters=idle\_on\_node\_suspend**

Mark nodes as idle, regardless of current state, when suspending nodes with SuspendProgram so that nodes will be eligible to be resumed at a later time.

#### **SuspendExcNodes**

Nodes not subject to suspend/resume logic. This may be used to avoid suspending and resuming nodes which are not in the cloud. Alternately the suspend/resume programs can treat local nodes differently from nodes being provisioned from cloud. Use Slurm's hostlist expression to identify nodes with an optional ":" separator and count of nodes to exclude from the preceding range. For example "nid[10\‐20]:4" will prevent 4 usable nodes (i.e IDLE and not DOWN, DRAINING or already powered down) in the set "nid[10\‐20]" from being powered down. Multiple sets of nodes can be specified with or without counts in a comma separated list (e.g "nid[10\‐20]:4,nid[80\‐90]:2"). By default, no nodes are excluded. This value may be updated with scontrol.

See **ReconfigFlags=KeepPowerSaveSettings** for setting persistence.

#### **SuspendExcParts**

List of partitions with nodes to never place in power saving mode. Multiple partitions may be specified using a comma separator. By default, no nodes are excluded. This value may be updated with scontrol.

See **ReconfigFlags=KeepPowerSaveSettings** for setting persistence.

#### **SuspendExcStates**

Specifies node states that are not to be powered down automatically. Valid states include CLOUD, DOWN, DRAIN, DYNAMIC\_FUTURE, DYNAMIC\_NORM, FAIL, INVALID\_REG, MAINTENANCE, NOT\_RESPONDING, PERFCTRS, PLANNED, and RESERVED. By default, any of these states, if idle for **SuspendTime**, would be powered down. This value may be updated with scontrol. See **ReconfigFlags=KeepPowerSaveSettings** for setting persistence.

#### **SuspendProgram**

The program executed when a node is no longer required and can be relinquished to the cloud.

#### **SuspendTime**

The time interval that a node will be left idle or down before a request is made to relinquish it. Units are in seconds.

#### **SuspendTimeout**

Maximum time permitted (in seconds) between when a node suspend request is issued and when the node is shutdown. At that time the node must be ready for a resume request to be issued as needed for new work. The default value is 30 seconds.

#### **TCPTimeout**

Time permitted for TCP connections to be established. This value may need to be increased from the default (2 seconds) to account for latency between your local site and machines in the cloud.

#### **TreeWidth**

Since the slurmd daemons are not aware of the network addresses of other nodes in the cloud, the slurmd daemons on each node should be sent messages directly and not forward those messages between each other. To do so, configure TreeWidth to a number at least as large as the maximum node count. The value may not exceed 65533.

Some node parameters that are of interest include:

#### **Feature**

A node feature can be associated with resources acquired from the cloud and user jobs can specify their preference for resource use with the "--constraint" option.

#### **NodeName**

This is the name by which Slurm refers to the node. A name containing a numeric suffix is recommended for convenience. The NodeAddr and NodeHostname should not be set, but will be configured later using scripts.

#### **State**

Nodes which are to be added on demand should have a state of "CLOUD". These nodes will not actually appear in Slurm commands until after they are configured for use.

#### **Weight**

Each node can be configured with a weight indicating the desirability of using that resource. Nodes with lower weights are used before those with higher weights.

Some partition parameters that are of interest include:

#### **PowerDownOnIdle**

If set to YES, then nodes allocated from this partition will immediately be requested to power down upon becoming IDLE. A power down request prevents further scheduling to the node until it has been put into power save mode by SuspendProgram.

#### **ResumeTimeout, SuspendTimeout and SuspendTime**

These can be applied at the partition level. If a node is in multiple partitions and these options are configured on the partitions, the higher value will be used for the node. If the option is not set, it will use the global setting. Configuring SuspendTime on a partition will enable power save mode if the global SuspendTime is not set.

Nodes to be acquired on demand can be placed into their own Slurm partition. This mode of operation can be used to use these nodes only if so requested by the user. Note that jobs can be submitted to multiple partitions and will use resources from whichever partition permits faster initiation. A sample configuration in which nodes are added from the cloud when the workload exceeds available resources. Users can explicitly request local resources or resources from the cloud by using the "--constraint" option.

```

Excerpt of slurm.conf

SelectType=select/cons_res

SelectTypeParameters=CR_CORE_Memory

SuspendProgram=/usr/sbin/slurm_suspend

ResumeProgram=/usr/sbin/slurm_suspend

SuspendTime=600

SuspendExcNodes=tux[0-127]

TreeWidth=128

NodeName=DEFAULT Sockets=1 CoresPerSocket=4 ThreadsPerCore=2

NodeName=tux[0-127] Weight=1 Feature=local State=UNKNOWN

NodeName=ec[0-127] Weight=8 Feature=cloud State=CLOUD

PartitionName=debug MaxTime=1:00:00 Nodes=tux[0-32] Default=yes

PartitionName=batch MaxTime=8:00:00 Nodes=tux[0-127],ec[0-127] Default=no

```

By default, when power save mode is enabled Slurm attempts to "suspend" all nodes unless excluded by **SuspendExcNodes** or **SuspendExcParts**. For bursting from on-premise scenarios this can be tricky to have to remember to add on-premise nodes to the excluded options. By setting the global **SuspendTime** to **INFINITE** and configuring **SuspendTime** on cloud specific partitions, you can avoid having to exclude nodes.

If power save mode is enabled globally, disable power save mode for a specific partition, by setting **SuspendTime** to **INFINITE**.

## Operational Details

When the slurmctld daemon starts, all nodes with a state of CLOUD will be included in its internal tables, but these node records will not be seen with user commands or used by applications until allocated to some job. After allocated, the *ResumeProgram* is executed and should do the following:

1. Boot the node
2. Configure and start Munge (depends upon configuration)

3. Install the Slurm configuration file, slurm.conf, on the node. Note that configuration file will generally be identical on all nodes and not include NodeAddr or NodeHostname configuration parameters for any nodes in the cloud. Slurm commands executed on this node only need to communicate with the slurmcld daemon on the SlurmctldHost.
4. Notify the slurmcld daemon of the node's hostname and network address:  
`scontrol update nodename=ec0 nodeaddr=123.45.67.89 nodehostname=whatever`  
Note that the node address and hostname information set by the scontrol command are preserved when the slurmcld daemon is restarted unless the "-c" (cold-start) option is used.
5. Start the slurmd daemon on the node

The *SuspendProgram* only needs to relinquish the node back to the cloud.

An environment variable SLURM\_NODE\_ALIASES contains sets of node name, communication address and hostname. The variable is set by salloc, sbatch, and srun. It is then used by srun to determine the destination for job launch communication messages. This environment variable is only set for nodes allocated from the cloud. If a job is allocated some resources from the local cluster and others from the cloud, only those nodes from the cloud will appear in SLURM\_NODE\_ALIASES. Each set of names and addresses is comma separated and the elements within the set are separated by colons. For example:

SLURM\_NODE\_ALIASES=ec0:123.45.67.8:foo,ec2:123.45.67.9:bar

## Cloud Node Lifecycle

A cloud node moves through different states when enabled with Power Saving mode. A node can have multiple states associated with it at one time. States associated with Power Saving are labeled with a symbol when viewing node details in "sinfo".

List of Node States a cloud node may have during autoscaling operations:

| <u>STATE</u> | <u>Power</u>  | <u>Description</u>                                                 |
|--------------|---------------|--------------------------------------------------------------------|
|              | <u>Saving</u> | <u>Symbol</u>                                                      |
| IDLE         |               | The node is not allocated to any jobs and is available for use.    |
| POWERED_DOWN | ~             | The node is powered off (node has been relinquished to the cloud). |
| ALLOCATED    |               | All CPUS on the node have been allocated to one or more jobs.      |
| MIXED        |               | The node has some of its CPUs allocated while others are idle.     |

|               |   |                                                                                                                                                                                                                                                                                                                                                                                                                        |
|---------------|---|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| POWERING_UP   | # | The node is in the process of being powered up.                                                                                                                                                                                                                                                                                                                                                                        |
| COMPLETING    |   | All jobs associated with this node are in the process of <i>COMPLETING</i> . This node state will be removed when all of the job's processes have terminated and the Slurm epilog program (if any) has terminated.                                                                                                                                                                                                     |
| POWERING_DOWN | % | The node is in the process of powering down and not capable of running any jobs.                                                                                                                                                                                                                                                                                                                                       |
| DOWN          |   | The node is unavailable for use. Slurm can automatically place nodes in this state if some failure occurs. System administrators may also explicitly place nodes in this state. If a node resumes normal operation, Slurm can automatically return it to service (See <i>ReturnToService</i> ). A DOWN Cloud node can be changed to IDLE using the scontrol command: <i>scontrol update node=[nodename] state=idle</i> |

Additional states nodes can have during manual power saving operations:

| <u>STATE</u> | <u>Power</u> | <u>Saving</u> | <u>Description</u>                                                       |
|--------------|--------------|---------------|--------------------------------------------------------------------------|
|              |              |               | <u>Symbol</u>                                                            |
| DRAINING     |              |               |                                                                          |
| POWER_DOWN   | !            |               | When the node is no longer running jobs, run the <b>SuspendProgram</b> . |
| DRAINED      |              |               | The node is unavailable for use per system administrator request.        |

A cloud node starts out in *IDLE* and *POWERED\_DOWN* (~) state. In this state, no actual node exists, as the node is in the cloud waiting to be provisioned. However, the *IDLE* state indicates that it is eligible for work.

```
PARTITION AVAIL TIMELIMIT NODES STATE NODELIST
debug* up infinite 100 idle~ c1-compute-0-[0-99]
```

When Slurm schedules a job to a node in this state, the node will move to *ALLOCATED* or *MIXED* (based on whether all the CPUs are allocated) and *POWERING\_UP (#)*.

| PARTITION | AVAIL | TIMELIMIT | NODES | STATE  | NODELIST            |
|-----------|-------|-----------|-------|--------|---------------------|
| debug*    | up    | infinite  | 1     | alloc# | c1-compute-0-0>     |
| debug*    | up    | infinite  | 99    | idle~  | c1-compute-0-[1-99] |

The scheduled job is in *CONFIGURING* state as it waits for the node to power up.

| JOBID | PARTITION | NAME | USER     | ST | TIME | NODES | NODELIST (REASON) |
|-------|-----------|------|----------|----|------|-------|-------------------|
| 182   | debug     | wrap | nick_sch | CF | 0:01 | 2     | c1-compute-0-0    |

After power up, the node registers with the Slurm controller and the *POWERING\_UP (#)* is removed. The job will start running after the node registers + MAX of 30 seconds, based on when the node registers and job starts begin.

| PARTITION | AVAIL | TIMELIMIT | NODES | STATE | NODELIST            |
|-----------|-------|-----------|-------|-------|---------------------|
| debug*    | up    | infinite  | 1     | alloc | c1-compute-0-0      |
| debug*    | up    | infinite  | 99    | idle~ | c1-compute-0-[1-99] |

| JOBID | PARTITION | NAME | USER     | ST | TIME | NODES | NODELIST (REASON) |
|-------|-----------|------|----------|----|------|-------|-------------------|
| 182   | debug     | wrap | nick_sch | R  | 0:04 | 2     | c1-compute-0-0    |

Once the job completes, if no other job is scheduled to the node, the node moves to *IDLE* state and the timer for the **SuspendTime** begins. The "scontrol show node" output displays *LastBusyTime*, the timestamp when the node stopped running a job and became idle.

When *SuspendTime* is reached, the node will move to *IDLE+POWERING\_DOWN (%)*. At this time, the **SuspendProgram** will execute. The node is not eligible for allocation until *POWERING\_DOWN* state is removed.

| PARTITION | AVAIL | TIMELIMIT | NODES | STATE | NODELIST            |
|-----------|-------|-----------|-------|-------|---------------------|
| debug*    | up    | infinite  | 1     | idle% | c1-compute-0-0      |
| debug*    | up    | infinite  | 99    | idle~ | c1-compute-0-[1-99] |

## POWER UP AND POWER DOWN FAILURES

A node may move into *DOWN* state if there is a failure during the *POWERING\_UP* or *POWERING\_DOWN* phase. The **ResumeTimeout** controls how long Slurm waits until nodes are expected to register. The node is

marked *DOWN* if it fails to register in that time and the jobs scheduled on the node are requeued. To move the node back into *IDLE* state, run:

```
scontrol update node=[nodename] state=resume
```

**SuspendTimeout** controls when Slurm will assume the node has been relinquished to the cloud. The node is marked *IDLE* and *POWERED\_DOWN* at this point. This step can be accelerated by running the "scontrol update node=[nodename] state=resume" manually, or in the **SuspendProgram** script.

#### MANUAL POWER SAVING

A node can be manually powered on and off by setting the state of the node to the following states using "scontrol update":

*POWER\_DOWN*, *POWER\_UP*, *POWER\_DOWN\_ASAP*, or *POWER\_DOWN\_FORCE*

*POWER\_DOWN* or *POWER\_UP* use the configured **SuspendProgram** and **ResumeProgram** programs to explicitly place a node in or out of a power saving mode. If a node is already in the process of being powered up or down, the command will only change the state of the node but won't have any effect until the configured **ResumeTimeout** or **SuspendTimeout** is reached.

For *POWER\_UP*, the node goes through a similar lifecycle as noted above, except it will be in *IDLE* state instead of *MIXED* or *ALLOCATED* as it is not allocated to a job yet.

*POWER\_DOWN* is signified by the (!) symbol. It marks the node to be powered down as soon as no jobs are running on the node anymore. Powering down may be delayed if more jobs are scheduled to the node while it is currently running other jobs. As soon as all jobs complete on the node, the node will move to *POWERING\_DOWN* (%).

*POWER\_DOWN\_ASAP* sets the node state as *DRAINING* and marks it to be powered off with the *POWER\_DOWN* state. *DRAINING* allows for currently running jobs to complete, but no additional jobs will be allocated to it. When the jobs are completed, the node will move to *POWERING\_DOWN* (%).

*POWER\_DOWN\_FORCE* cancels all jobs on the node and suspends the node immediately, placing the node in *IDLE* and *POWER\_DOWN* (!) state and then *POWERING\_DOWN* (%) state.

```
scontrol update nodename=c1-compute-0-0 state=power_down_asap
```

| PARTITION | AVAIL | TIMELIMIT | NODES | STATE | NODELIST            |
|-----------|-------|-----------|-------|-------|---------------------|
| debug*    | up    | infinite  | 1     | drng! | c1-compute-0-0      |
| debug*    | up    | infinite  | 99    | idle~ | c1-compute-0-[1-99] |

#### NODEFEATURESPLUGIN AND CLOUD NODES

Features defined by a [NodeFeaturesPlugin](#) and attached to a cloud node in the **slurm.conf**, will be available but not active when the node is powered down. If a job requests NodeFeaturesPlugin features, the controller will allocate nodes that are powered down and have the features as available. At allocation, the features will be made active. A cloud node will remain with the active features until the node is powered down (i.e. the node can't be rebooted to get other features until the node is powered down). When the node is powered down, the active NodeaFeaturesPlugin features are cleared. Any non-NodeFeaturesPlugin features are active by default and can be used as a label.

Example:

```
slurm.conf:

NodeFeaturesPlugin=node_features/helpers

NodeName=cloud[1-5] ... State=CLOUD Feature=f1,f2,l1
NodeName=cloud[6-10] ... State=CLOUD Feature=f3,f4,l2

helpers.conf:

NodeName=cloud[1-5] Feature=f1,f2 Helper=/bin/true
NodeName=cloud[6-10] Feature=f3,f4 Helper=/bin/true
```

Features f1, f2, f3, and f4 are changeable features and are defined on the node lines in the slurm.conf because **CLOUD** nodes do not register before being allocated. By setting the *Helper* script to /bin/true, the slurmd's will not report any active features to the controller and the controller will manage all the active features. If the *Helper* is set to a script that reports the active features, the controller will validate that the reported active features are a super set of the node's active changeable features in the controller. Features l1 and l2 will always be active and can be used as selectable labels.

Slurm v23.02 updates:

1. Run the following scripts in slurmscriptd instead of slurmctld: ResumeProgram, ResumeFailProgram, SuspendProgram, ResvProlog, ResvEpilog, and RebootProgram (only with SlurmctldParameters=reboot\_from\_controller).
2. Add PowerDownOnIdle partition option to power down nodes after nodes become idle.
3. Make node weight usable for powered down and rebooting nodes.
4. Allow nodefeatures plugin features to work with cloud nodes.
5. Add ability to update SuspendExc\* parameters with scontrol.
6. Add ability to restore SuspendExc\* parameters on restart with slurmctld -R option.
7. Add new "getnameinfo\_cache\_timeout=<number>" option to CommunicationParameters to adjust or disable caching the results of getnameinfo().
8. Add SuspendExcStates option to slurm.conf to avoid suspending/powering down specific node states



## Slurm Cloud Lab Exercise

The Docker Scaleout platform, in which you are doing these lab exercises, provides the ability to launch the scheduler and related environment in a “cloud” configuration. The cloud bursting capabilities are somewhat limited in this environment, since it is not designed to reach out to a cloud provider (AWS, GCP, Azure, etc), rather it can spin up additional Docker containers to do the workload. This method is the same, however, if you are reaching out to a cloud provider to spin up additional nodes.

The purpose of this environment and configuration is to show demonstrate how the process works.

### Exercise: Boot the Docker Scaleout environment in “Cloud” configuration

In this exercise, you will start the environment in a cloud-ready state. You will need 3 separate terminal connections to the AWS learning environment.

1. In the first terminal, exit out of any Docker containers and launch cloud-ready environment as the ubuntu user by typing the following command at the Ubuntu prompt:

```
cd /docker-scale-out
```

```
export SLURM_RELEASE=slurm-23-02-1-1
```

```
make clean && make cloud
```

After a moment you should see:

```
...
#: Container docker-scale-out-node06-1 Started 81.7s
#: Container docker-scale-out-node02-1 Started 83.2s
#: Container docker-scale-out-rest-1 Started 86.1s
#: Container docker-scale-out-node05-1 Started 80.6s
#: Container docker-scale-out-node07-1 Started 83.3s
#: Container docker-scale-out-node00-1 Started 82.8s
...
```

2. Now launch another terminal window and get into the login Docker container. In it, as root, launch the following to watch the behavior of the cloud nodes when being provisioned:

```
watch sinfo -Nel -pcloud
```

You should see:

```
Every 2.0s: sinfo -Nel -pcloud

Thu Mar 09 08:49:23 2023
NODELIST NODES PARTITION STATE CPUS S:C:T MEMORY TMP_DISK WEIGHT AVAIL_FE REASON
cloud0000 1 cloud idle~ 4 1:4:1 15967 0 8 cloud none
cloud0001 1 cloud idle~ 4 1:4:1 15967 0 8 cloud none
cloud0002 1 cloud idle~ 4 1:4:1 15967 0 8 cloud none
cloud0003 1 cloud idle~ 4 1:4:1 15967 0 8 cloud none
cloud0004 1 cloud idle~ 4 1:4:1 15967 0 8 cloud none
```

**NOTE** The tilde symbol (~) indicates that the cloud nodes are in a powered-off state. In addition to that, they are labeled as idle because they are not allocated any resources

3. In your third terminal window, as user fred in the login container, launch a job that requires some cloud nodes:

```
sbatch -N5 -t10 --mem=1000 -pcloud --wrap "hostname"
```

You should see:

```
Every 2.0s: sinfo -Nel -pcloud

Thu Mar 09 08:49:23 2023
NODELIST NODES PARTITION STATE CPUS S:C:T MEMORY TMP_DISK WEIGHT AVAIL_FE REASON
cloud0000 1 cloud mixed# 4 1:4:1 15967 0 8 cloud none
cloud0001 1 cloud mixed# 4 1:4:1 15967 0 8 cloud none
cloud0002 1 cloud mixed# 4 1:4:1 15967 0 8 cloud none
cloud0003 1 cloud Mixed# 4 1:4:1 15967 0 8 cloud none
cloud0004 1 cloud mixed# 4 1:4:1 15967 0 8 cloud none
```

**NOTE** The pound/hash symbol (#) indicates that the cloud nodes are in a powering-on state. In addition to that, they are labeled as mixed because they are allocated any resources

After the nodes are powered on, you should see:

| Every 2.0s: sinfo -Nel -pcloud |       |           |       |      |       |        |          |        |          |        |
|--------------------------------|-------|-----------|-------|------|-------|--------|----------|--------|----------|--------|
| NODELIST                       | NODES | PARTITION | STATE | CPUS | S:C:T | MEMORY | TMP_DISK | WEIGHT | AVAIL_FE | REASON |
| cloud0000                      | 1     | cloud     | mixed | 4    | 1:4:1 | 15967  | 0        | 8      | cloud    | none   |
| cloud0001                      | 1     | cloud     | mixed | 4    | 1:4:1 | 15967  | 0        | 8      | cloud    | none   |
| cloud0002                      | 1     | cloud     | mixed | 4    | 1:4:1 | 15967  | 0        | 8      | cloud    | none   |
| cloud0003                      | 1     | cloud     | Mixed | 4    | 1:4:1 | 15967  | 0        | 8      | cloud    | none   |
| cloud0004                      | 1     | cloud     | mixed | 4    | 1:4:1 | 15967  | 0        | 8      | cloud    | none   |

The # symbol is gone, meaning the nodes are fully up and running workload

4. When the nodes have reached SuspendTime (100 seconds in this environment) the node states will change to idle%. The % indicates a powering-down state

## Cleanup

1. Reset the Docker Slurm cluster environment back to its original, pristine state by exiting the login Docker container back to the Ubuntu AWS prompt. And **as the ubuntu user**:

**Break out of the first terminal make process  
make clean && make**

## Appendix

## File Movement Discussion

In Slurm, there are a few ways to move data files around. Using the scp underlying technology, Slurm can automatically move data and program files around the cluster for computational functionality

### **sbcast**

**sbcast** is used to transmit a file to all nodes allocated to the currently active Slurm job. This command should only be executed from within a Slurm batch job or within the shell spawned after a Slurm job's resource allocation.

SOURCE is the name of a file on the current node. DEST should be the fully qualified pathname for the file copy to be created on each node. If a fully qualified pathname is not provided, the file will be created in the directory specified in the SbcastParameters parameter in the slurm.conf file (if available) otherwise it will be created in the current working directory from which the **sbcast** command is invoked. DEST should be on a file system local to that node.

**NOTE:** A parallel file system may provide better performance than **sbcast** can provide, although performance will vary by file size, degree of parallelism, and network type.

## OPTIONS

|                                                  |                                                                                                                                                                                                                                                                                                                                                                             |
|--------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>-C [library], --compress[=library]</b>        | Compress the file being transmitted. The optional argument specifies the data compression library to be used. Supported values are "lz4" (default), "none" and "zlib". Some compression libraries may be unavailable on some systems. The default compression library (and enabling compression itself) may be set in the slurm.conf file using the SbcastParameter option. |
| <b>-f, --force</b>                               | If the destination file already exists, replace it.                                                                                                                                                                                                                                                                                                                         |
| <b>-F number, --fanout=number</b>                | Specify the fanout of messages used for file transfer. Maximum value is currently eight.                                                                                                                                                                                                                                                                                    |
| <b>-j jobID[.stepID], --jobid=jobID[.stepID]</b> | Specify the job ID to use with optional step ID. If run inside an allocation this is unneeded as the job ID will read from the environment.                                                                                                                                                                                                                                 |
| <b>-p, --preserve</b>                            | Preserves modification times, access times, and modes from the original file.                                                                                                                                                                                                                                                                                               |
| <b>-s size, --size=size</b>                      | Specify the block size used for file broadcast. The size can have a suffix of k or m for kilobytes or megabytes respectively (defaults to bytes). This size subject to rounding and range limits to maintain good performance. The default value is the file size or 8MB, whichever is smaller. This value may need to be set on systems with very limited memory.          |
| <b>-t seconds, fB--timeout=seconds</b>           | Specify the message timeout in seconds. The default value is MessageTimeout as reported by "scontrol show config". Setting a higher value may be necessitated by relatively slow I/O performance on the compute node disks.                                                                                                                                                 |
| <b>-v, --verbose</b>                             | Provide detailed event logging through program execution.                                                                                                                                                                                                                                                                                                                   |
| <b>-V, --version</b>                             | Print version information and exit.                                                                                                                                                                                                                                                                                                                                         |

## EXAMPLE

Using a batch script, transmit local file my.prog to /tmp/my.proc on the local nodes and then execute it.

```
> cat my.job
#!/bin/bash
sbcast my.prog /tmp/my.proc
srun /tmp/my.proc

> sbatch --nodes=8 my.job
srun: jobid 12345 submitted
```

## sgather

sgather is used to transmit a file from all nodes allocated to the currently active SLURM job. This command should only be executed from within a SLURM batch job or within the shell spawned after a SLURM job's resource allocation. SOURCE should be the fully qualified pathname for the file copy to be fetched from each node. SOURCE should be on a file system local to that node. DEST is the name of the file to be created on the current node where the source node name will be appended. Note that parallel file systems may provide better performance than sgather can provide, although performance will vary by file size, degree of parallelism, and network type.

## OPTIONS

- C, --compress** Compress the file being transmitted.
- f, --force** Ignore nonexistent source file.
- F number, --fanout=number** Specify the fanout of messages used for file transfer. Maximum value is currently eight.
- k, --keep** Do not remove the source file after transmission.
- p, --preserve** Preserves modification times, access times, and modes from the original file.
- r, --recursive** Copy directories recursively.
- t seconds, --timeout=seconds** Specify the message timeout in seconds. The default value is 60 seconds.
- v, --verbose** Provide detailed event logging through program execution
- V, --version** Print version information and exit

## EXAMPLE

Using a batch script, execute a program that produces /tmp/my.data on all nodes allocated to the SLURM job and then transmit these files to the batch node.

```
> cat my.job
#!/bin/bash
srun my.prog --output /tmp/my.data
sgather /tmp/my.data all_data

> sbatch --nodes=8 my.job
srun: jobid 12345 submitted
```

## File Movement Exercises

In this set of labs, you will learn how to manage the movement of files with jobs

### Exercise 1: Using `sbcast` and `sgather`

- As fred, in his home directory, create a 1GB data file that will be sent to the compute nodes for processing:

```
perl -le 'print for 1..1000000000' > ~/dataset
```

**NOTE:** This file is just a series of numbers, but you will use it during a job to follow.

- Copy the dataprocessingjob.sh file from /lab\_scripts:

```
cp /lab_scripts/dataprocessingjob.sh .
```

**NOTE:** This is a job submit script. It will use the Slurm `sbcast` and `sgather` commands to send out the `dataprocessingprogram.sh` script to each compute nodes' `/tmp` directory, then run the `dataprocessingprogram.sh` script.

- As fred, copy the `dataprocessingprogram.sh` file from /lab\_scripts:

```
cp /lab_scripts/dataprocessingprogram.sh
```

This will be the job you submit when you `sbatch` the submission script.

- Make those two files executable:

```
chmod +x dataprocessingjob.sh dataprocessingprogram.sh
```

**NOTE:** It's important to flag the `dataprocessingprogram.sh` as executable because it is to be run on the compute nodes from `/tmp`!

- In a separate terminal window, watch the files being distributed and programs being run (**enter commands exactly as shown -- you will see a continuation prompt (>) as you do:**)

```
while true ; do
 for x in node{00..04} ; do
 ssh $x ls -al /tmp/dataset*
 done
 sleep 2
 clear
done
```

**NOTE:** Since you are listing the files in a directory that does NOT contain files yet (until you run your job submission script), bash will throw errors until they exist. This is perfectly normal, do not panic.

- Now, back to fred user, launch the job:

```
sbatch -N5 dataprocessingjob.sh
```

**NOTE:** You will see the dataset file being `sbcast`'ed to the nodes, then the calculations will be made on the jobs, then the `.out` files will be SCP'd back to fred's home folder.

**ALSO NOTE:** By default, the *sgather* will append the nodeID of where the jobs ran.

7. Using squeue, look at the job information:

**squeue**

You should see:

| JOBID | PARTITION | NAME     | USER | ST | TIME | NODES | NODELIST (REASON) |
|-------|-----------|----------|------|----|------|-------|-------------------|
| 2     | debug     | dataproc | fred | R  | 1:26 | 5     | node[00-04]       |

**NOTE:** The job is running across node00-node04 (a 5-node job)

8. You will be able to see the job output by looking at the resultant “tac’d” files:

**ls -al ~/**

You should see:

```
total 5208372
-rwxr-xr-x 1 fred users 190 Nov 26 23:49 dataprocessingjob.sh
-rwxr-xr-x 1 fred users 50 Nov 26 23:49 dataprocessingprogram.sh
-rw-r--r-- 1 fred users 888888898 Nov 26 23:48 dataset
-rw-r--r-- 1 fred users 888888898 Nov 26 23:54 dataset.out.node00
-rw-r--r-- 1 fred users 888888898 Nov 26 23:54 dataset.out.node01
-rw-r--r-- 1 fred users 888888898 Nov 26 23:54 dataset.out.node02
-rw-r--r-- 1 fred users 888888898 Nov 26 23:54 dataset.out.node03
-rw-r--r-- 1 fred users 888888898 Nov 26 23:54 dataset.out.node04
-rw-r--r-- 1 fred users 0 Nov 26 23:51 slurm-2.out
```

## Cleanup

1. Reset the Docker Slurm cluster environment back to its original, pristine state by exiting the login Docker container back to the Ubuntu AWS prompt. And **as the ubuntu user**:

**cd ~/docker-scale-out  
make clean && make**