Agenda

- Job Prioritization
- Fairshare
- Fair Tree
Agenda

● Job Prioritization
● Fairshare
● Fair Tree
Why Prioritize?

- Because **NOT** all clusters have 200,000 nodes
Why Prioritize?

- Because **NOT** all clusters have 200,000 nodes
Why Prioritize?

- Because **NOT** all clusters have 200,000 nodes
- **NOT** on the Top 500 list
General Boarding

Priority
- First Class
- Business Class
- Dividend Miles Preferred® Members
- American Airlines AAdvantage® Elite Members
- US Airways PreferredAccess
Slurm Job Prioritization Discussion

- FIFO by default
- Priority Plugins define Slurm’s behavior:
  - Priority/basic - Provides rudimentary FIFO scheduling (default)
  - Priority/multifactor - Sets priority based on:
    - Job Age
    - Fairshare
    - Job Size
    - Queue/Partition
    - QOS
    - TRES
    - Nice
    - Assoc (19.05)
    - Site (19.05)
Job Prioritization Discussion

- FIFO by default
- Priority Plugins define Slurm’s behavior:
  - Priority/basic - Provides rudimentary FIFO scheduling (default)
  - Priority/multifactor - Sets priority based on:
    - Job Age
    - Fairshare
    - Job Size
    - Queue/Partition
    - QOS
    - TRES
    - Nice
    - Assoc (19.05)
    - Site (19.05)
Define the plugin in slurm.conf:

```
PriorityType=priority/basic # Default
or
PriorityType=priority/multifactor
```
Slurm Quick Scheduling Iteration

Job Queue

Node 1 | Busy | Busy | Busy
--- | --- | --- | ---
Node 2
Node 3
Node 4
Node 5

Cluster
Slurm Quick Scheduling Iteration

1. Build List

Job Queue

Priority

Job 3
State: PD
Part: 1, 2

Job 5
State: R
Part: 2

Job 2
State: R
Part: 2

Job 1
State: PD
Part: 1

Job 6
State: PD
Part: 1

Job 7
State: PD
Part: 1

Cluster

Node 1: Busy

Node 2

Node 3

Node 4

Node 5
Slurm Quick Scheduling Iteration

1. Build List
   - Job 3: State:PD, Part: 1,2
   - Job 5: State:R, Part: 2
   - Job 2: State:R, Part: 2
   - Job 6: State:PD, Part: 2
   - Job 7: State:PD, Part: 1

2. Sort List
   - Job 3: State:PD, Part: 1
   - Job 6: State:PD, Part: 1
   - Job 1: State:PD, Part: 1
   - Job 1: State:PD, Part: 1
   - Job 3: State:PD, Part: 2

Priority Type (Precedence Order):
1. Preempt
2. Advanced Reservation
3. Priority Tier
4. Job Priority
5. Submit Time
6. Job ID

Priority/Multifactor:
- QOS
- FS
- JobSize
- ...

Cluster
- Node 1: Busy
- Node 2
- Node 3
- Node 4
- Node 5

Job Queue

Priority
Slurm Quick Scheduling Iteration

1. Build List
2. Sort List
3. Schedule
   SchedulerParameters=default_queue_depth=4

   until(DefaultQueueDepth) {
     pop 1 job from list
     try to schedule...
     call select plugin
   }

PriorityType
(Precedence Order):
1. Preempt
2. Advanced Reservation
3. PriorityTier
4. Job Priority
5. Submit Time
6. JobID

Priority/Multifactor:
QOS
FS
JobSize
...

Cluster
Slurm Quick Scheduling Iteration

1. Build List
2. Sort List
3. Schedule
   SchedulerParameters=default_queue_depth=4

   until(DefaultQueueDepth) {
     pop 1 job from list
     try to schedule...
     call select plugin
   }

PriorityType
(Precedence Order):
1. Preempt
2. Advanced Reservation
3. Priority Tier
4. Job Priority
5. Submit Time
6. Job ID

Priority/Multifactor:
QoS
FS
Job Size
...

Cluster

Node 1
- Busy
- Busy
- Busy

Node 2
- Job 3
  - State: PD
  - Part: 1

Node 3
- Job 6
  - State: PD
  - Part: 1

Node 4
- Job 1
  - State: PD
  - Part: 1

Node 5
- Job 3
  - State: PD
  - Part: 2
Job Prioritization Factors

**Age** - the length of time a job has been waiting in the queue, eligible to be scheduled

**Fairshare** - 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
Job Prioritization Factors - Cont’d

**TRES** - Each TRES Type has its own factor for a job which represents the number of requested/allocated TRES Type in a given partition

**Nice** - Users can adjust the priority of their own jobs by setting the nice value on their jobs

**Association** - Each association can be assigned an integer priority

**Site** - Can be set either using scontrol, through a job_submit or site_factor plugin
Job Prioritization Value Calculation

\[
\text{Job\_priority} = \\
(\text{PriorityWeightAge}) \times (\text{age\_factor}) + \\
(\text{PriorityWeightFairshare}) \times (\text{fair\_share\_factor}) + \\
(\text{PriorityWeightJobSize}) \times (\text{job\_size\_factor}) + \\
(\text{PriorityWeightPartition}) \times (\text{partition\_factor}) + \\
(\text{PriorityWeightQOS}) \times (\text{QOS\_factor}) + \\
\text{SUM} (\text{TRES\_weight\_cpu} \times \text{TRES\_factor\_cpu}, \\
\text{TRES\_weight\_<type>} \times \text{TRES\_factor\_<type>}, \\
...)
\]
Notes on Prioritization

- All **factors** in the formula are floating point numbers 0.0-1.0
- **Weights** are unsigned 32-bit integers
- Slurm priority is **normalized***

*Can be turned off for some priority calculations in 19.05.x code
Age Factor

- Length of time a job has been sitting in the queue, eligible to run
- Age factor for dependencies does not change while it waits on the depended job to complete
- At PriorityMaxAge, the age factor max’s out at 1.0. The default is 7 days, meaning after 7 days, all jobs get the same age-based priority
Fairshare Factor

- Fairshare calculation in Slurm requires the Slurm Accounting Database to provide the assigned shares and the consumed computing resources.
- Takes into consideration computing resources allocated and computing resources already consumed.
- The fairshare factor prioritizes queued jobs based on under/over utilization.
Fairshare Factor (Cont’d)

- Fairshare factor is a floating point number between 0.0 and 1.0
- You can configure TRESBillingWeights on a partition to account for consumed resources other than just CPUs
- 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:

<table>
<thead>
<tr>
<th></th>
<th>CPUs</th>
<th>Mem GB</th>
</tr>
</thead>
<tbody>
<tr>
<td>Job1</td>
<td>(1 *1.0)</td>
<td>(60*0.25)</td>
</tr>
<tr>
<td>Job2</td>
<td>(16*1.0)</td>
<td>(1 *0.25)</td>
</tr>
<tr>
<td>Job3</td>
<td>(16*1.0)</td>
<td>(60*0.25)</td>
</tr>
</tbody>
</table>
Job Size Factor

- Correlates to the number of nodes or CPUs requested
- Can favor either larger or smaller jobs, depending on PriorityFavorSmall=yes/no
- A job that requests all the nodes gets a job size factor of 1.0
- If PriorityFavorSmall=yes, a single node job will get a job size factor of 1.0
Can alter scheduler job size factor behavior by setting:

PriorityFlags=SMALL_RELATIVE_TO_TIME:

\[
\frac{\text{job size in CPUs}}{\text{time limit in minutes}} \div \text{total cpus in cluster} = \text{Job Size Factor}
\]

A full-system job with a time limit of 1 will receive a job size factor of 1.0, while a tiny job with a large time limit will receive a job size factor closer to 0.0.
Partition Factor (priority job factor on a partition)

- Each partition (queue) 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
- The priority 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 requesting that QOS
● The priority is then normalized to the highest priority of all the QOSs to become the QOS factor
TRES Factors

- Each TRES Type has its own priority factor which represents the amount of TRES Type requested/allocated in a given partition.
- For globals, like Licenses, the factor represents the number of TRES Type requested/allocated in the whole system.
- The more TRES Type is requested/allocated on a job, the greater the job priority will be for that job.
Nice Factor

- Users can adjust the priority of their own jobs by setting the nice value on their jobs.
- Like the system nice, positive values negatively impact a job's priority and positive values increase a job's priority.
- Only privileged users can specify a negative value.
- The adjustment range is +/-2147483645.
Association Factor

- Each association can be assigned an integer priority
- The larger the number, the greater the job priority will be for jobs that request this association
- This priority value is normalized to the highest priority of all the association to become the association factor
Site Factor

- The site factor is a factor that can be set either using scontrol, through a job_submit or site_factor plugin
- An example use case, might be a job_submit plugin that sets a specific priority based on how many resources are requested
QOS Priority Calculation Example

PriorityType=priority/multifactor
PriorityWeightQOS=1000

$>sprio -w

<table>
<thead>
<tr>
<th>JOBID PARTITION</th>
<th>PRIORITY</th>
<th>SITE</th>
<th>QOS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Weights</td>
<td></td>
<td>1</td>
<td>1000</td>
</tr>
</tbody>
</table>

Copyright 2019 SchedMD LLC
www.schedmd.com
SLUG Sep 17-18, 2019
### QOS Priority Calculation Example-Cont’d

```bash
$> sacctmgr -i add qos high set priority=1000
$> sacctmgr -i add qos medium set priority=500
$> sacctmgr -i add qos low set priority=100
$> sacctmgr -i modify account bedrock set qos=low
$> sacctmgr -i modify account bedrock set defaultqos=low
```

```bash
$> sacctmgr show qos format=name,priority
   Name   Priority
   --------
   normal 0
   high 1000
   medium 500
   low 100
```
QOS Priority Calculation Example-Cont’d

```bash
$sacctmgr -i modify user fred,barney set qos=high,medium,low
$sacctmgr -i modify user wilma,betty set qos=medium,low
```

```bash
$sacctmgr list assoc user=fred,barney,wilma,betty,bambam,pebbles format=Cluster,Account,User,QOS,defaultqos
```

<table>
<thead>
<tr>
<th>Cluster</th>
<th>Account</th>
<th>User</th>
<th>QOS</th>
<th>Def QOS</th>
</tr>
</thead>
<tbody>
<tr>
<td>cluster</td>
<td>bedrock</td>
<td>bambam</td>
<td>low</td>
<td>low</td>
</tr>
<tr>
<td>cluster</td>
<td>bedrock</td>
<td>barney</td>
<td>high,low,medium</td>
<td>low</td>
</tr>
<tr>
<td>cluster</td>
<td>bedrock</td>
<td>betty</td>
<td>low,medium</td>
<td>low</td>
</tr>
<tr>
<td>cluster</td>
<td>bedrock</td>
<td>fred</td>
<td>high,low,medium</td>
<td>low</td>
</tr>
<tr>
<td>cluster</td>
<td>bedrock</td>
<td>pebbles</td>
<td>low</td>
<td>low</td>
</tr>
<tr>
<td>cluster</td>
<td>bedrock</td>
<td>wilma</td>
<td>low,medium</td>
<td>low</td>
</tr>
</tbody>
</table>
## QOS Priority Calculation Example—Cont’d

```bash
$ sprio -o "%.15i %9r %.8u %.10Y"
```

<table>
<thead>
<tr>
<th>JOBID</th>
<th>PARTITION</th>
<th>USER</th>
<th>PRIORITY</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>debug</td>
<td>fred</td>
<td>1000000</td>
</tr>
<tr>
<td>3</td>
<td>debug</td>
<td>barney</td>
<td>1000000</td>
</tr>
<tr>
<td>4</td>
<td>debug</td>
<td>wilma</td>
<td>500000</td>
</tr>
<tr>
<td>5</td>
<td>debug</td>
<td>betty</td>
<td>500000</td>
</tr>
<tr>
<td>6</td>
<td>debug</td>
<td>bambam</td>
<td>100000</td>
</tr>
<tr>
<td>7</td>
<td>debug</td>
<td>pebbles</td>
<td>100000</td>
</tr>
</tbody>
</table>
QOS Priority Calculation Example-Cont’d

- The Math:

<table>
<thead>
<tr>
<th>Priority Weight QOS</th>
<th>QOS Factor</th>
<th>Job Priority</th>
</tr>
</thead>
<tbody>
<tr>
<td>1000</td>
<td>1000/1000</td>
<td>1.0 (for fred)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Or, 100% of Priority Weight QOS</td>
</tr>
<tr>
<td>From slurm.conf</td>
<td>priority/highest QOS Prio</td>
<td></td>
</tr>
<tr>
<td>1000</td>
<td>500/1000</td>
<td>.50 (for wilma)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Or, 50% of Priority Weight QOS</td>
</tr>
<tr>
<td>From slurm.com</td>
<td>priority/highest QOS Prio</td>
<td></td>
</tr>
<tr>
<td>1000</td>
<td>100/1000</td>
<td>.10 (for bambam)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Or, .10% of Priority Weight QOS</td>
</tr>
<tr>
<td>From slurm.conf</td>
<td>priority/highest QOS Prio</td>
<td></td>
</tr>
</tbody>
</table>
Priority Change in 19.05

- PriorityFlags = NO_NORMAL_ALL
  - NO_NORMAL_ALL If set, all NO_NORMAL_* flags are set.
  - NO_NORMAL_ASSOC If set, the association factor is **not** normalized against the highest association priority.
  - NO_NORMAL_PART If set, the partition factor is **not** normalized against the highest partition PriorityTier.
  - NO_NORMAL_QOS If set, the QOS factor is **not** normalized against the highest qos priority.
  - NO_NORMAL_TRES If set, the QOS factor is **not** normalized against the job's partition TRES counts.
Non-Normalized Math (using PriorityFlags = NO_NORMAL_ALL):

Job_priority =
... (PriorityWeightQOS) * (QOS_factor) + ...)

1 * 1000 = 1000 (1*1000 for fred)
From slurm.conf

1 * 500 = 500 (1*500 for wilma)
From slurm.com

1 * 100 = 100 (1*100 for bambam)
From slurm.conf
Agenda

- Job Prioritization
- Fairshare
- Fair Tree
Fairshare-Fair vs Equal
Fairshare

- In Slurm, Fairshare shares are normalized
- The fair-share hierarchy represents the portions of the computing resources 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
Fairshare-Usage Factor

- In Slurm, Fairshare usage is normalized
  - 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

\[ U_N = \frac{U_{\text{user}}}{U_{\text{total}}} \]

Where:
- \( U_N \) is normalized usage, between zero and one
- \( U_{\text{user}} \) is the processor*seconds consumed by all of a user's jobs in a given account for over a fixed time period
- \( U_{\text{total}} \) is the total number of processor*seconds utilized across the cluster during that same time period
Fairshare-Decay Factor

- Most workload spans multiple time periods. Slurm’s fairshare priority calculation places more importance on the most recent resource usage and less importance on usage from way back.
- The metric used is based on a half-life formula that favors most recent usage statistics, based on a decay factor ($D$):

$$U_H = U_{\text{current\_period}} + (D \times U_{\text{last\_period}}) + (D \times D \times U_{\text{period-2}}) + \ldots$$

Where:
- $U_H$ is the historical usage subject to the half-life decay
- $U_{\text{current\_period}}$ is the usage charged over the current measurement period
- $U_{\text{last\_period}}$ is the usage charged over the last measurement period
- $U_{\text{period-2}}$ 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
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 ranges from zero to one

- One represents the highest priority for a job
- A fair-share factor of 0.5 indicates that the user's jobs have used/not 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
Fairshare Factor Under 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.
Fairshare Factor Under Account Hierarchy

- 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 this example, 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 Fairshare Formula

- The Slurm Fairshare formula has been designed to provide fair scheduling to users based on the allocation and usage of every account. Now, the usage term is effective usage:

\[ F = 2^{(-U_E/S)} \]  (Effective Usage Formula)

\[ U_E = U_{Achild} + \\ \quad \quad \quad \quad \quad ((U_{Eparent} - U_{Achild}) \times S_{child}/S_{all_siblings}) \]

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
The Slurm Fairshare Formula

- 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.
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.
Slurm Fairshare Example

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.
Slurm Fairshare Example

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.
Slurm Fairshare Example

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
Slurm Fairshare Example

The effective usage for **all accounts** at the first tier under the root allocation is always equal to the actual usage:

\[ F = 2^{-U_E/S} \]  (Effective Usage Formula)

\[ U_E = U_{A\text{child}} + ((U_{E\text{parent}} - U_{A\text{child}}) \times S_{\text{child}}/S_{\text{all siblings}}) \]

Account A's effective usage is therefore equal to 0.45. Account D's effective usage is equal to 0.25.

Account B effective usage: \(0.2 + ((0.45 - 0.2) \times 30 / 40) = 0.3875\)

Account C effective usage: \(0.25 + ((0.45 - 0.25) \times 10 / 40) = 0.3\)

Account E effective usage: \(0.25 + ((0.25 - 0.25) \times 25 / 60) = 0.25\)

Account F effective usage: \(0.0 + ((0.25 - 0.0) \times 35 / 60) = 0.1458\)
Slurm Fairshare Example

The effective usage for each user is calculated using the same formula:

\[ F = 2^{\left( - \frac{U_E}{S} \right)} \]  

\[ U_E = U_{\text{Achild}} + \left( (U_{\text{Eparent}} - U_{\text{Achild}}) \times \frac{S_{\text{child}}}{S_{\text{all siblings}}} \right) \]

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
Slurm Fairshare Example

Using the Slurm fair-share formula,

\[ F = 2^{(-U_E/S)} \]  (Effective Usage Formula)

<table>
<thead>
<tr>
<th>User</th>
<th>Fair-share factor: (2^{(-U_E/S)})</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>(2^{(-.3875 / .3)})</td>
<td>0.408479</td>
</tr>
<tr>
<td>2</td>
<td>(2^{(-.275 / .05)})</td>
<td>0.022097</td>
</tr>
<tr>
<td>3</td>
<td>(2^{(-.15 / .05)})</td>
<td>0.125000</td>
</tr>
<tr>
<td>4</td>
<td>(2^{(-.25 / .25)})</td>
<td>0.500000</td>
</tr>
<tr>
<td>5</td>
<td>(2^{(-.1458 / .35)})</td>
<td>0.749154</td>
</tr>
</tbody>
</table>

From this example, once 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.
Slurm Fairshare Configuration Parameters

- **PriorityType**
  - priority/basic or priority/multifactor

- **PriorityDecayHalfLife**
  - min, hr:min:00, days-hr:min:00, or days-hr (Default=7 days)

- **PriorityCalcPeriod**
  - The period of time in minutes in which the half-life decay will be re-calculated. Default=5 minutes
Slurm Fairshare Configuration Parameters

- **PriorityUsageResetPeriod**
  - At this interval the usage of associations will be reset to 0
  - NONE, NOW, DAILY, WEEKLY, MONTHLY, QUARTERLY, YEARLY

- **PriorityFavorSmall**
  - Specifies that small jobs should be given preferential scheduling priority. Values=yes/no

- **PriorityMaxAge**
  - The job age which will be given the maximum age factor in computing priority. Default=7 Days
Slurm Fairshare Configuration Parameters

- **PriorityWeightAge**
  - the degree to which the queue wait time component contributes to the job's priority

- **PriorityWeightFairshare**
  - the degree to which the fair-share component contributes to the job's priority

- **PriorityWeightJobSize**
  - the degree to which the job size component contributes to the job's priority
Slurm Fairshare Configuration Parameters

- **PriorityWeightPartition**
  - Partition factor used by priority/multifactor plugin in calculating job priority

- **PriorityWeightQOS**
  - The degree to which the Quality Of Service component contributes to the job's priority

- **PriorityWeightTRES**
  - A comma-separated list of TRES Types and weights that sets the degree that each TRES Type contributes to the job's priority:
    
    \[
    \text{PriorityWeightTRES=CPU}=1000, \text{Mem}=2000, \text{GRES/gpu}=3000
    \]
This 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.

```bash
# 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
Agenda

- Job Prioritization
- Fairshare
- Fair Tree
Fair Tree

- Developed by Ryan Cox and Levi Morrison at BYU (Thanks!)
- Submitted to Slurm branch 14.03
- Algorithm includes a rooted plane tree (rooted ordered tree) being created then sorted by Level Fairshare
- Now the default fairshare algorithm in Slurm 19.05 release
- To revert to old-style fairshare:
  
  \[ \text{PriorityFlags=NO}\_\text{FAIR}\_\text{TREE} \]
Fair Tree - GO BYU!

Thanks for hosting SLUG!
Benefits of Fair Tree:

- All users from a higher priority account receive a higher fair share factor than all users from a lower priority account.
- Users are sorted and ranked to prevent errors due to precision loss. Ties are allowed.
- Account coordinators cannot accidentally harm the priority of their users relative to users in other account.
- Users are extremely unlikely to have exactly the same fairshare factor as another user due to loss of precision in calculations.
- New jobs are immediately assigned a priority.
To see the effect of Fair Tree:
  - sshare -l (Long) parameter now shows Level FS, showing fairshare calculated values for each association at each level

<table>
<thead>
<tr>
<th>Account</th>
<th>User</th>
<th>Raw Shares</th>
<th>Norm Shares</th>
<th>Raw Usage</th>
<th>Norm Usage</th>
<th>Effectv Usage</th>
<th>FairShare</th>
<th>Level FS</th>
</tr>
</thead>
<tbody>
<tr>
<td>root</td>
<td></td>
<td>0.000000</td>
<td>1230</td>
<td></td>
<td>1.000000</td>
<td></td>
<td></td>
<td>1.000000</td>
</tr>
<tr>
<td>bedrock</td>
<td></td>
<td>0.500000</td>
<td>676</td>
<td>0.549593</td>
<td>0.549593</td>
<td></td>
<td></td>
<td>0.909763</td>
</tr>
<tr>
<td>bedrock</td>
<td>fred</td>
<td>0.250000</td>
<td>301</td>
<td>0.244715</td>
<td>0.445266</td>
<td>0.200000</td>
<td></td>
<td>0.561462</td>
</tr>
<tr>
<td>bedrock</td>
<td>barney</td>
<td>0.250000</td>
<td>102</td>
<td>0.082927</td>
<td>0.150888</td>
<td>0.600000</td>
<td>1.656863</td>
<td></td>
</tr>
<tr>
<td>bedrock</td>
<td>wilma</td>
<td>0.250000</td>
<td>37</td>
<td>0.030081</td>
<td>0.054734</td>
<td>0.800000</td>
<td>4.567568</td>
<td></td>
</tr>
<tr>
<td>bedrock</td>
<td>betty</td>
<td>0.250000</td>
<td>236</td>
<td>0.191870</td>
<td>0.349112</td>
<td>0.400000</td>
<td>0.716102</td>
<td></td>
</tr>
<tr>
<td>managers</td>
<td></td>
<td>0.500000</td>
<td>554</td>
<td>0.450407</td>
<td>0.450407</td>
<td></td>
<td></td>
<td>1.110108</td>
</tr>
<tr>
<td>slate</td>
<td>slate</td>
<td>1.000000</td>
<td>554</td>
<td>0.450407</td>
<td>1.000000</td>
<td>1.000000</td>
<td></td>
<td>1.000000</td>
</tr>
</tbody>
</table>

To see the effect of Fair Tree:
  - sshare -l (Long) parameter now shows Level FS, showing fairshare calculated values for each association at each level
Summary

- You should now have working knowledge of Slurm Priority and Fairshare, and Fair Trees

- Questions?