FAIRSHARE MANAGEMENT FOR ALTAIR® PBS PROFESSIONAL®

Joe Miller III – Technical Support Engineer, Altair / August 7, 2020

Introduction to PBS Professional

Altair PBS Professional is a fast, powerful workload manager designed to improve productivity, optimize utilization and efficiency, and simplify administration for HPC clusters, clouds, and supercomputers. It automates job scheduling, management, monitoring, and reporting, and it's the trusted solution for complex Top500 systems and smaller clusters.

Introduction to Fairshare

Fairshare is a PBS Professional scheduling tool designed to share a cluster's limited resources according to cluster usage history and allocated percentage. Fairshare is the most direct option to grant a percentage of shared resources on the cluster to users, projects, or groups.

Scope

This white paper covers the creation and management of the fairshare tree structure for your PBS complex. It also covers basic settings and describes concepts of shares within the fairshare tree. Understanding, calculating, and managing entity usage data will not be covered herein. It assumes your familiarity with basic fairshare setup along with the limitations and caveats fairshare has with other PBS Professional scheduling tools.

Challenges

Enabling basic fairshare, as described in the Altair PBS Professional Admin Guide and Basic Fairshare for Altair PBS Professional white paper, does not consider fairshare tree capabilities and structure or describe how to create and manage the tree.

Granting a percentage of the cluster to individual users may not be viable; the use of your cluster may instead be defined with respect to queues, projects, or accounting. Further, some users may span across multiple groups.

Your site's policy may require the restriction of cluster use to only known entities (definition below). PBS Professional's weighted *rooted tree structure* provides the opportunity to group entities together and finely manage the balance of resources on your cluster. Adapting resource usage allows you to dictate tracked resources and apply a formula to balance those resources according to your site's needs.

Definition of Terms

Entity – An entity is a PBS job owner according to fairshare; a leaf on the fairshare tree. Entities are what need fairness or balancing with fairshare. Only one type of entity is used throughout fairshare. An entity can be based on:

- System UID
- System GID
- Both GID and UID (if a user is part of multiple groups)
- Accounting string at job submission
- Execution queue from which the job was run

Rooted Tree Structure – A model used in graph theory which connects vertices (or nodes) with each other using a simple acyclic path (a line). A single vertex is labeled as "root" for reference to other vertices.

For simplicity in this document, we'll refer to terminal vertices (those with only 1 line connected) as **leaves**, inner vertices (those with more than 1 line connected; having children) as **branches**, and the root vertex as **root** or **TREEROOT**.

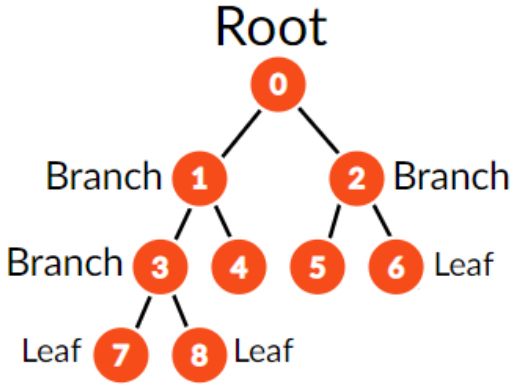

Figure 1: Labeled Rooted Tree Structure

See figure 1 for an example with many vertices labeled.

Fairshare Tree – A representation of a labeled rooted tree structure with shares and usage defined. The *resource_group* file is used to define each vertex, both leaves and branches, and to grant shares (allocating weight). Vertices are given unique names and IDs (labeled) for fairshare to reference when processing usage data.

Leaves are represented by entities without children. Branches are represented by assigning entities to them, being a parent to any number of vertices. Siblings are vertices (both branches and leaves) with the same direct parent, even if that parent is root (rooted).

Shares – Branches and leaves are given shares when defined in the *resource_group* file. Shares are weighted against direct siblings to determine percentage of allocation when the parent is granted resources. The actual value of shares is only weighted against siblings. Effectively, shares are only used to calculate the percentage of use within a sibling group.

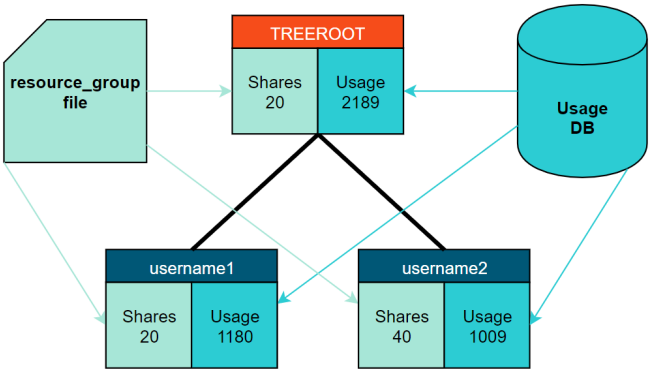

Figure 2: Fairshare Tree With resource_group File and Usage Database References

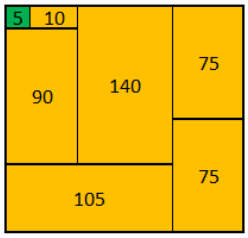

Figure 3A: 1% Share

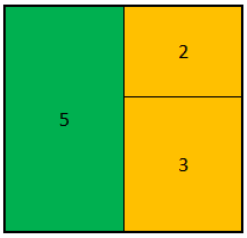

Figure 3B: 50% Share

Example: If the total number of shares among siblings is 500 and a specific sibling has 5, that entity is granted 1% of whatever allocation of resources comes through the parent branch. However, if that same entity continues to have 5 shares but the sum of self and siblings is 10 shares, it will be allocated 50% of resources granted to the parent.

Configuring the Fairshare Tree

Fairshare has two files to configure behavior to your requirements, both in *$PBS_HOME/sched_priv* directory.

Only parameters and files to define the structure of the fairshare tree and allocation of shares are described here. Usage formula and decay settings are not included.

File: *sched_config*

Parameter: fairshare_entity

The `fairshare_entity` parameter defines how fairshare views a leaf on the fairshare tree; how to define "entity" when recording usage. By default, it is set to "`euser`" (system user identified by UID).

- `euser` (default) – PBS job owner system username (based on UID)
- `egroup` – PBS job owner system group name (based on GID)
- `egroup:euser` – PBS job owner system username and default group name or job's group_list string
- `account_name` – PBS job's accounting string
- `queue` – PBS job's execution queue

A system user must use qsub's `-Wgroup_list <groupname>` argument to submit to a non-default group.

You can use `egroup:euser` to grant system users multiple leaves in the fairshare tree, each leaf with its own shares.

Job submission will fail if attempting to submit without the system user belonging to the desired group:

```
$ qsub -Wgroup_list=bar -- /bin/sleep 20
qsub: Bad GID for job execution
```

Parameter: unknown_shares

All entities not labeled in the *resource_group* file will be defined as a leaf of the unknown branch in the fairshare tree. The leaves of the unknown branch attempt to share equally with their siblings; there are no weighted leaves here. This is the percentage of the cluster you are willing to allocate to all unknown entities.

By default, the `unknown_shares` parameter is commented out (effectively set to `0` shares).

```
#unknown_shares: 0
```

Parameter: fairshare_enforce_no_shares

This parameter determines what to do when a leaf is allocated 0 shares in the fairshare tree.

A leaf may be allocated 0 shares in the *resource_group* file by:

- Leaf allocated with 0 shares
- Any parent branch of the leaf allocated with 0 shares (including `unknown` branch)

By default, `fairshare_enforce_no_shares` is `TRUE` (including when commented out).

```
fairshare_enforce_no_shares: TRUE
```

- If `TRUE`, jobs will never run from this entity, but stay in queued state with job comment:

```
Can Never Run: Job has zero shares for fairshare
```

- If `FALSE`, the entity's jobs will be considered as lowest priority after all other share-allocated entities

File: *resource_group*

This file defines both the structure and the allotted shares of the fairshare tree but not accumulated entity usage. Each line of code in the file will be considered for processing a vertex on the tree: its name, ID, parent, and shares.

File conventions:

- One vertex defined per line
- Children cannot be labeled before their parents
- All text is case-sensitive in the *resource_group* file
- Blank lines, tabs, comments, and extra whitespace act only as beautification

Field	Description
<vertex name>	Either: • Branch name (B# in Figure 3 below) • Leaf name (L# in Figure 3 below)
<vertex fairshare ID>	Unique ID to be associated with the vertex. This ID is only used within fairshare. No value will be referenced elsewhere. No structure needs to be adhered to.
<parent of vertex>	The <vertex name> of this vertex's direct parent; "root" if a child of root
<#shares>	Shares allocated to this vertex; will be weighed against all siblings

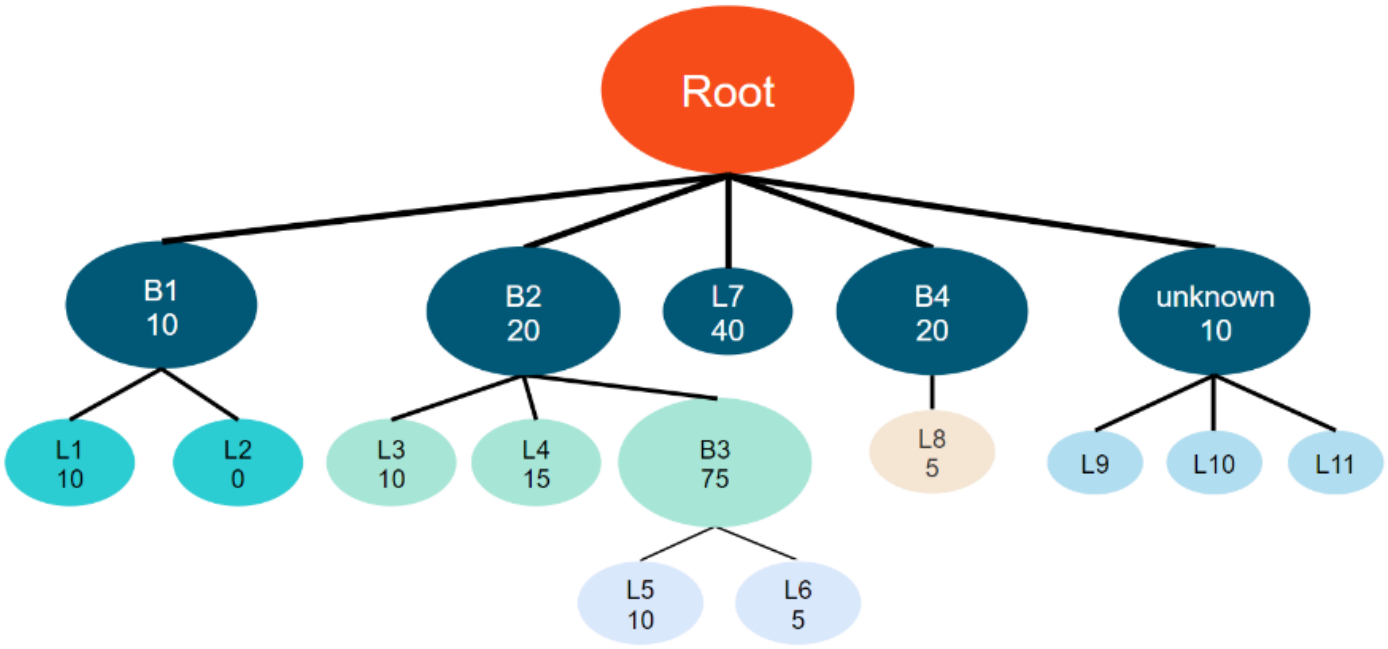

Figure 4: Example Fairshare Tree

Example *resource_group* file of Fairshare Tree (from Figure 4):

Notice how `root`, `unknown`, and `unknown`'s children are not defined here. **Headings added for reference only; they are not in *resource_group* file**.

<vertex name>	<vertex fairshare ID>	<parent of vertex>	<#shares>
B1	100	root	10
L1	101	B1	10
L2	102	B1	0
B2	200	root	20
L3	201	B2	10
L4	202	B2	15
B3	210	B2	75
L5	211	B3	10
L6	212	B3	5
L7	001	root	40
B4	300	root	20
L8	301	B4	5

Example Scenario:

Many users are working on different projects and sub-projects. Some users work on multiple projects (lynne). One entity (bob) takes general precedence over the server, running critical or time-sensitive jobs. One user (lynne again) is granted server time for testing against a project (Impact) only when resources are available (lynne still accumulates usage). Cross-training and new software testing are encouraged; unknown users (scott, joe, lisa, and possibly more) are welcome to test their tasks on the cluster.

Placeholder names have been switched to show naming features. Notice siblings are colored in respective color groups.

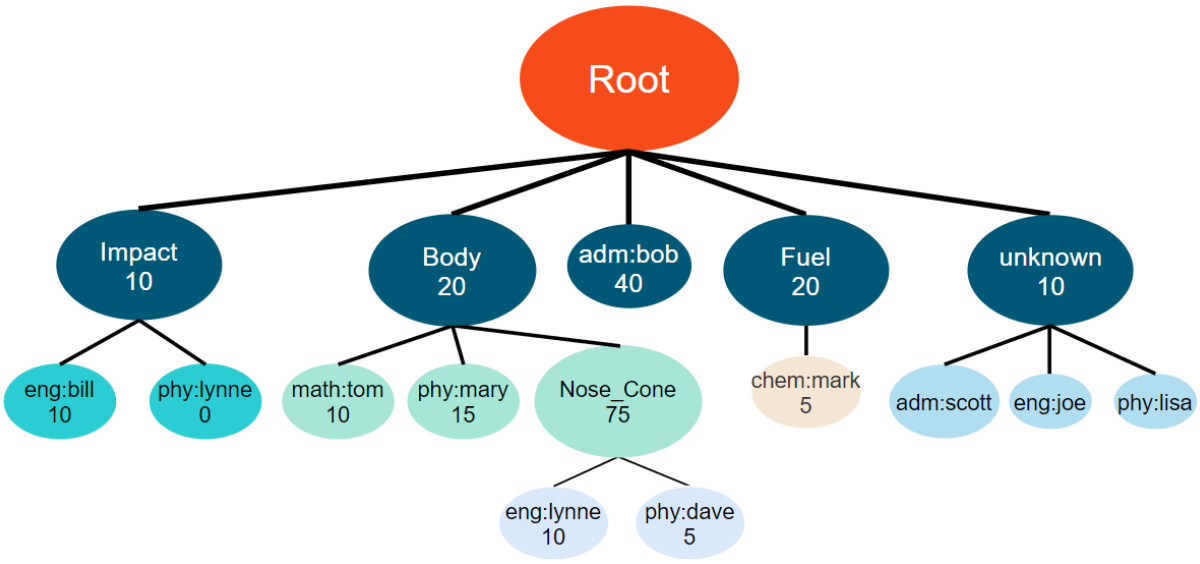

Figure 5: Example Scenario Fairshare Tree With Placeholder Names

This would be the updated lines of code in *resource_group* file. Notice the system user lynne having two positions in the tree:


```
Impact        100      root         10
eng:bill      101      Impact       10
phy:lynne     102      Impact       0
Body          200      root         20
math:tom      201      Body         10
phy:mary      202      Body         15
Nose_Cone     210      Body         75
eng:lynne     211      Nose_Cone    10
phy:dave      212      Nose_Cone    5
adm:bob       001      root         40
Fuel          300      root         20
chem:mark     301      Fuel         5
```

Any changes made to the *resource_group* file or *sched_config* file require sending a HANGUP signal to the pbs_sched daemon before they take effect.

```
# kill -HUP $(pgrep -f pbs_sched)
```

You can view your current fairshare tree structure using:

```
# pbsfs -t
```

Here is the output for our example tree (with added unknown leaves):

```
TREEROOT(0)
      Fuel(300)
            chem:mark(301)
      adm:bob(001)
      Body(200)
            Nose_Cone(210)
                  phy:dave(212)
                  eng:lynne(211)
            phy:mary(202)
            math:tom(201)
      Impact(100)
            phy:lynne(102)
            eng:bill(101)
      unknown(1)
            adm:scott(-1)
            eng:joe(-1)
            phy:lisa(-1)
```


Managing the Fairshare Tree

We at Altair Support suggest pausing the execution of new jobs from the scheduler while updating the fairshare tree.

```
# qmgr -c "set sched scheduling = False"
    -- Make Changes to resource_group file --
# kill -HUP $(pgrep -f pbs_sched)
# qmgr -c "set sched scheduling = True"
```

You can change the fairshare tree by altering the resource_group file. The scheduler daemon will re-read this configuration file when a HANGUP signal is sent to pbs_sched daemon process. Only the leaf names are compared to the usage database (changing the vertex fairshare ID does not change usage). Entity and branch shares are calculated against the newly read configuration file from then on.

Use the process above to:

- Change share count (allocation) to branches and leaves
- Move leaves and branches
- Add/remove branches
- Add new leaves

Removing Leaves

Any leaf which has usage recorded in the usage database will survive all changes made to the resource_group file and HANGUP signal. If a leaf is not expressly named, the leaf and its usage will be allocated to the unknown group of the fairshare tree. You will still be able to see the leaf's usage using the `pbsfs` command and its place in the tree using `pbsfs -t` command.

Leaves allocated to the unknown branch (after HANGUP signal) will be removed, along with all their usage, when using:

```
# pbsfs -e
```

All known and defined leaves of the fairshare tree will maintain their usage through this command.

In the Figure 5 example, the leaves adm:scott; eng:joe; and phy:lisa would be removed from the tree and all accumulated usage erased.

Conclusion

The fairshare tree gives you fine-grained management over your cluster. It can be used to balance resources among projects, queues, individual users, or your own accounting structure.

