



# LCI Advanced Workshop 2025: Fair Resource Allocation

Alan Chapman  
HPC Systems Analyst  
Arizona State University Research Computing  
alan.chapman@asu.edu

*This document is a result of work by volunteer LCI instructors and is licensed under CC BY-NC 4.0  
(<https://creativecommons.org/licenses/by-nc/4.0/>)*

# AGENDA

- Fairshare Fundamentals & Why It Matter
- Classic Fairshare Algorithm Deep-Dive
- Fairtree Algorithm Architecture
- Comparative Analysis & Trade-offs
- Configuration & Tuning Best Practice
- Q&A + Lab Preview

# WHY FAIRSHARE MATTERS

## The Problem: Shared Resources

Timeline: 8 compute nodes, 2 accounts (root, lci)

09:00 → root submits 100-node job (queues, needs 92 more nodes)

09:05 → lci submits 4-node job

09:10 → lci's job waits behind root's queued job

(even though lci has no running jobs!)

Without fairshare: FIFO (First-In-First-Out) → starvation

With fairshare: Fair priority based on historical usage

## Why It Matters

- Multi-tenant clusters: Fair allocation across departments/project
- Prevents hoarding: Users can't monopolize resources long-term
- Incentivizes efficiency: Heavy users get lower priority (temporary)
- Academic/Industry: Standard practice for HPC governance

# FAIRSHARE FUNDAMENTALS

Concept	Definition
Fairshare Factor	User's fair allocation ÷ actual consumption (0.0–1.0)
Decay Half-Life	Time for past usage to lose 50% weight (e.g., 7 days)
Account Hierarchy	Tree structure: root → dept → project → user
Priority Formula	Job priority = (fairshare × weight) + (age × weight) + (QOS × weight)
Usage Tracking	Slurm tracks CPU-minutes, memory, GPU-minutes per account

## Example Calculation

Account: lci/ml

Fair Share: 50% (assigned)

Usage: 10,000 CPU-min over 7 days

Fairshare Factor = (Fair Share - Usage) / Fair Share

= (50% - 45%) / 50% = 0.1 (LOW priority)

Account: lci/io

Fair Share: 50%

Usage: 5,000 CPU-min

Fairshare Factor = (50% - 22.5%) / 50% = 0.55 (HIGHER priority)

# CLASSIC FAIRSHARE ALGORITHM

## The Algorithm (Slurm $\leq$ 22.04)

FOR each pending job:

```
fairshare_factor = calculate_factor(job.account)
```

```
priority = (fairshare_factor × weight_fairshare)
```

```
+ (age_factor × weight_age)
```

```
+ (qos_factor × weight_qos)
```

```
queue_job_by_priority(job, priority)
```

FUNCTION calculate\_factor(account):

# Recursive: parent's factor affects children

```
parent_factor = calculate_factor(parent_account)
```

```
my_usage = sum(usage_since_decay_time)
```

```
my_share = my_fairshare_pct × parent_factor
```

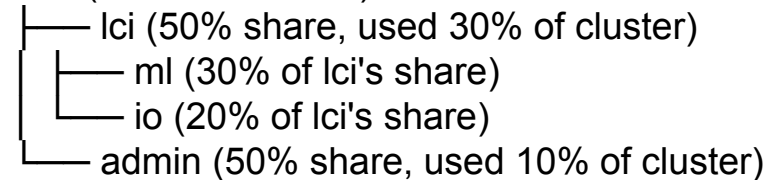
```
factor = (my_share - my_usage) / my_share
```

```
return max(0, factor) # Clamp to [0, 1]
```

# CLASSIC FAIRSHARE – REAL EXAMPLE

## Scenario: 2-Level Hierarchy

root (100% fair share)



## Job Priority Calculation

Job A: account=lci/ml

Parent factor (lci) =  $(50\% - 30\%) / 50\% = 0.4$

My usage (ml) = 5% of cluster

My fair share =  $30\% \times 0.4 = 12\%$

My factor =  $(12\% - 5\%) / 12\% = 0.58 \rightarrow$  MEDIUM priority

Job B: account=lci/io

Parent factor (lci) = 0.4 ← SAME as ml!

My usage (io) = 8% of cluster

My fair share =  $20\% \times 0.4 = 8\%$

My factor =  $(8\% - 8\%) / 8\% = 0.0 \rightarrow$  LOW priority

Job C: account=admin

Parent factor (root) =  $(100\% - 10\%) / 100\% = 0.9$

My usage (admin) = 10% of cluster

My fair share =  $50\% \times 0.9 = 45\%$

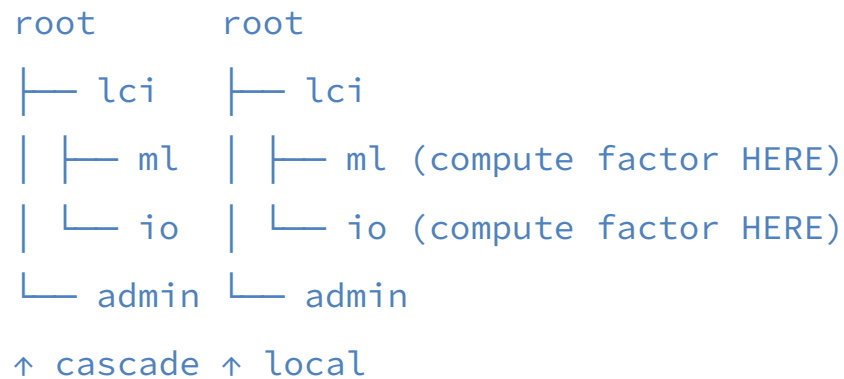
My factor =  $(45\% - 10\%) / 45\% = 0.78 \rightarrow$  HIGH priority

# FAIRTREE ALGORITHM (INTRO)

## What Changed in Slurm 22.05+

Goal: Eliminate cascade penalty; compute fairshare locally per node

Classic: Fairtree:



Key Innovation: Sibling Fairness

In Fairtree, each account's factor depends ONLY on:

1. Its own usage
2. Its own fair share
3. Its siblings' usage (not ancestors)

## Algorithm Sketch

```
FUNCTION fairtree_factor(account):
my_usage = sum(usage_since_decay_time)
my_share = account.fairshare_pct
# Key: compute against siblings, not parent
sibling_total_share =
sum(sibling.fairshare_pct)
sibling_total_usage = sum(sibling.usage)
# Normalize my share within sibling group
my_normalized_share = my_share /
sibling_total_share
factor = (my_normalized_share - my_usage) /
my_normalized_share
return max(0, factor)
```

# FAIRTREE – REAL EXAMPLE

## Same Scenario, Fairtree Algorithm

root (100% fair share)

|— lci (50% share, used 30% of cluster)

| |— ml (30% of lci's share)

| |— io (20% of lci's share)

|— admin (50% share, used 10% of cluster)

**Job B:** account=lci/io

Sibling group: {ml, io}

Sibling total share: 50%

My normalized share: 20% / 50% = 40%

My usage: 8% of cluster

My factor = (40% - 8%) / 40% = 0.80 →  
MEDIUM priority ✓

## Job Priority Calculation (Fairtree)

**Job A:** account=lci/ml

Sibling group: {ml, io}

Sibling total share: 30% + 20% = 50%

My normalized share: 30% / 50% = 60%

My usage: 5% of cluster

My factor = (60% - 5%) / 60% = 0.92 → HIGH  
priority ✓

**Job C:** account=admin

Sibling group: {lci, admin}

Sibling total share: 50% + 50% = 100%

My normalized share: 50% / 100% = 50%

My usage: 10% of cluster

My factor = (50% - 10%) / 50% = 0.80 →  
MEDIUM priority ✓



# COMPARATIVE ANALYSIS – COMPLEXITY

## Algorithm Complexity

Metric	Classic	Fairtree
Time per job	$O(\text{depth} \times n)$	$O(\text{children})$
Scheduling cycle (1000 jobs, 200 accounts)	~500ms	~50ms
Scalability	Poor (deep trees)	Excellent
Memory overhead	Low	Medium (sibling caches)
Decay recalc	Global (expensive)	Local (cheap)

## Behavior Differences

Scenario	Classic	Fairtree
Parent over-allocated	Children all penalized	Each child judged fairly
Sibling comparison	Indirect (via parent)	Direct
New account (zero usage)	High priority	High priority
Starvation risk	YES (cascade)	NO (local max)
Predictability	Low	High

# IMPLEMENTATION – SLURM CONFIG

## **Enable Fairtree (Slurm 22.05+)**

```
# In slurm.conf

PriorityType=priority/multifactor

PriorityFlags=FAIR_TREE                # Fair Tree
#PriorityFlags=DEPTH_OBLIVIOUS         # Classic Fairshare

# Tune weights (example for CPU cluster)
PriorityWeightFairshare=100000000 # Fairshare dominates priority
PriorityWeightAge=10000000 # Age: jobs wait, priority increases
PriorityWeightQOS=110000000 # QOS: can override fairshare

# Decay half-life (past usage fades over time)
PriorityDecayHalfLife=7-0 # 7 days; usage older than this loses weight

# Usage reset (optional; default: NONE)
PriorityUsageResetPeriod=NONE # Don't reset; continuous decay

# Backfill (allows smaller jobs to fill gaps)
SchedulerParameters=bf_window=43200,bf_resolution=300
```

# ACCOUNT HIERARCHY DESIGN

## Bad: Flat Hierarchy

```
root (100% fair share)
├─ alice (1% share)
├─ bob (1% share)
├─ charlie (1% share)
└─ ... (97 more users)
```

## Good: Hierarchical (Recommended)

```
root (100% fair share)
├─ admin (5% share)
│   └─ sysadmin (100% of admin's share)
├─ research (70% share)
│   ├── ml_team (40% of research)
│   │   ├── alice (50% of ml_team)
│   │   └─ bob (50% of ml_team)
│   └─ bio_team (30% of research)
│       ├── charlie (60% of bio_team)
│       └─ diana (40% of bio_team)
└─ teaching (25% share)
    └─ class_2024 (100% of teaching)
```

# PRACTICAL IMPACTS – ACADEMIC WORKLOAD

**Scenario:** Multi-PI Department

Department: 16 cores, 64 GB RAM (shared)

Account tree:

root

```
|— pi_alice (50% fair share)
|   |— phd_student_1 (50% of alice's)
|   |— phd_student_2 (50% of alice's)
|— pi_bob (50% fair share)
|   |— phd_student_3 (50% of bob's)
|   |— phd_student_4 (50% of bob's)
|— postdoc_1 (0%) ← Temporary, gets scraps
```

# PRACTICAL IMPACTS – ACADEMIC WORKLOAD

## Workload Pattern

Time	Job	Submitter	Cores	Status	Classic Factor	Fairtree Factor
09:00	phd_student_1	4	RUNNING	1.0	1.0	
09:30	phd_student_3	4	RUNNING	1.0	1.0	
10:00	phd_student_2	4	PENDING	0.5	0.9 ← Fairtree higher!	
(reason: alice over-allocated in classic)						
10:15	postdoc_1	2	PENDING	0.0	0.7 ← Fairtree higher!	
(postdoc has no fair share, but Fairtree still gives chance)						

# PRACTICAL IMPACTS – ACADEMIC WORKLOAD

## Wait Time Impact:

Classic:

phd\_student\_2: waits 45 min (cascade penalty from alice over-use)

postdoc\_1: waits indefinitely (zero fair share)

Fairtree:

phd\_student\_2: waits 10 min (fair within ml\_team)

postdoc\_1: waits 20 min (gets some priority, prevents starvation)

# PRACTICAL IMPACTS – CPU WORKLOADS

## Scenario: Batch Job Farm

Cluster: 128 cores, 2 accounts

```
root (100%)
├─ lci (60% share)
│   ├─ batch_inference (40% of lci)
│   └─ data_processing (60% of lci)
└─ admin (40% share)
```

## Job Submission Pattern

09:00 batch\_inference submits  $50 \times 4$ -core jobs (200 cores needed)  
Only 4 cores available; 49 jobs queued

09:05 data\_processing submits  $10 \times 2$ -core jobs (20 cores)  
All queued behind batch\_inference

09:10 admin submits  $1 \times 8$ -core job  
All queued

09:15 First batch\_inference job finishes (4 cores freed)

## Scheduling Decision

Classic Fairshare:

- lci parent factor:  $(60\% - 4\%) / 60\% = 0.93$
- batch\_inference factor: 0.85 (over-used relative to share)
- data\_processing factor: 0.95 (under-used)
- Schedules: data\_processing ( $0.95 > 0.85$ )
- Batch job waits longer ✗

Fairtree:

- batch\_inference vs. data\_processing (siblings)
- batch\_inference used 4%, share 40% → factor 0.90
- data\_processing used 0%, share 60% → factor 1.0
- Schedules: data\_processing ( $1.0 > 0.90$ )
- Same decision, but for right reason ✓



# TUNING FAIRSHARE – WEIGHTS

## Priority Formula (Multifactor)

```
job_priority = (fairshare_factor × weight_fs)  
+ (age_factor × weight_age)  
+ (qos_factor × weight_qos)  
+ (job_size_factor × weight_size)
```

## Weight Scenarios

### Scenario 1: Fairshare-Dominant (Research Cluster)

```
PriorityWeightFairshare=100000000  
PriorityWeightAge=10000000  
PriorityWeightQOS=1000000
```

**Effect:** Historical usage is primary; age and QOS minor.

**Use case:** Multi-PI research; want fair allocation of resources.

# TUNING FAIRSHARE – WEIGHTS

## Scenario 2: Age-Dominant (Responsiveness)

PriorityWeightFairshare=1000000

PriorityWeightAge=100000000

PriorityWeightQOS=1000000

**Effect:** Jobs waiting longer get boosted; fairshare secondary.

**Use case:** Interactive clusters; want quick turnaround for all users.

## Scenario 3: QOS-Dominant (SLAs)

PriorityWeightFairshare=1000000

PriorityWeightAge=1000000

PriorityWeightQOS=100000000

**Effect:** QOS classes override fairshare.

**Use case:** Production systems; high-priority jobs must run.

# TUNING FAIRSHARE – DECAY

## Decay Half-Life Impact

PriorityDecayHalfLife=7-0 # 7 days

## Timeline Example

Day 0: User runs 1000 CPU-min job  
Recorded usage: 1000 CPU-min

Day 7: (half-life reached)  
Effective usage: 500 CPU-min (50% decay)

Day 14: (2× half-life)  
Effective usage: 250 CPU-min (75% decay)

Day 21: (3× half-life)  
Effective usage: 125 CPU-min (87.5% decay)

# TUNING FAIRSHARE – DECAY

## Policy Implications

Half-Life	Policy	Effect
-----	-----	-----
1 day	Short memory	Users can "reset" by waiting 1 day; incentivizes efficiency
7 days	Balanced	Past sins fade over 1-2 weeks; standard.
30 days	Long memory	Historical usage matters for months; strict fairness
$\infty$ (NONE)	Infinite memory	All past usage counts forever; most conservative

## Lab Comparison

Config A: `PriorityDecayHalfLife=1-0`

Config B: `PriorityDecayHalfLife=7-0`

Config C: `PriorityDecayHalfLife=NONE`

Submit workload, measure priority evolution over time.

# BEST PRACTICES

## ✓ Do's

1. Use Fairtree (Slurm 22.05+)
  - Migrate from classic; performance + fairness gains
2. Design hierarchy to match org structure
  - root → department → project → user
  - Typically 3-4 levels; avoid >5 (diminishing returns)
3. Allocate fair shares proportionally to allocation
  - If ml\_team gets 40% of budget, give 40% fair share
  - Not 50% (too generous) or 30% (too stingy)
4. Set decay half-life to your cluster culture
  - Research: 7 days (balanced)
  - Production: 14-30 days (strict)
  - Dev: 1 day (forgiving)
5. Use PriorityWeightFairshare >> other weights
  - Fairshare should dominate (100M vs. 10M for age)
6. Monitor fairshare factors regularly
  - ``sshare -l`` or ``sacctmgr show assoc``
  - Alert if any account hits factor 0.0 (starvation risk)

## ✗ Don'ts

1. Don't use classic fairshare (pre-22.05)
  - Upgrade; Fairtree is proven, faster
2. Don't create >100 accounts per level
  - Scheduling overhead; merge into fewer buckets
3. Don't set fair share >100% at any level
  - Violates fairness semantics; set to 100% or less
4. Don't use NONE for decay half-life on interactive clusters
  - Users get "locked out" after big job; use 1-7 days
5. Don't ignore fairshare in QOS
  - QOS can override fairshare; use sparingly (SLAs only)
6. Don't change hierarchy without planning
  - Moving accounts can cause starvation; test first

# MIGRATION GUIDE

## From Classic to Fairtree

**Step 1:** Backup current slurm.conf

```
cp /etc/slurm/slurm.conf /etc/slurm/slurm.conf.bak
```

**Step 2:** Add FAIR\_TREE flag

```
sed -i 's/PriorityFlags=DEPTH_OBLIVIOUS/PriorityFlags=FAIR_TREE/' /etc/slurm/slurm.conf
```

**Step 3:** Reload Slurm config (no restart needed)

```
scontrol reconfigure
```

**Step 4:** Verify

```
scontrol show config | grep PriorityFlags
```

**Step 5:** Monitor for 1 week

```
watch -n 60 'sshare -l | head -20'
```

**Rollback if needed:**

```
sed -i 's/PriorityFlags=FAIR_TREE/PriorityFlags=DEPTH_OBLIVIOUS/' /etc/slurm/slurm.conf
```

```
scontrol reconfigure
```

# MIGRATION GUIDE

## Expected Changes in First Week

### Before (Classic):

- Fairshare factors: erratic, hard to explain
- Priority jumps: large, unpredictable
- Wait times: variable for same-sized jobs

### After (Fairtree):

- Fairshare factors: stable, intuitive
- Priority progression: smooth
- Wait times: more predictable
- Scheduling decisions: faster (50ms vs. 500ms per cycle)

# SUMMARY

## Key Takeaways

### 1. Fairshare is essential for multi-tenant clusters

- Prevents starvation, incentivizes efficiency, enables governance

### 2. Fairtree (22.05+) beats classic

- $O(n)$  vs.  $O(n^2)$ , local fairness vs. cascading penalties, predictable

### 3. Config matters

- Hierarchy design, fair share allocation, weights, decay half-life

### 4. Migration is easy

- One flag; no restart; immediate benefits



# QUESTIONS & RESOURCES

## Questions?

### Resources:

Fair Tree - [https://slurm.schedmd.com/fair\\_tree.html](https://slurm.schedmd.com/fair_tree.html)

Classic Fair Share - [https://slurm.schedmd.com/classic\\_fair\\_share.html](https://slurm.schedmd.com/classic_fair_share.html)

Multifactor Priority Plugin - [https://slurm.schedmd.com/priority\\_multifactor.html](https://slurm.schedmd.com/priority_multifactor.html)

### Lab Setup:

- Cluster: lciadv (2 compute nodes, 2 cores each)
- Accounts: root, lci
- Users: root, rocky (lci)
- Duration: 60 min
- All CPU workloads (no GPU)