Toward Dynamically Controlling Slurm's Classic Fairshare Algorithm

Conference Paper · July 2020

DOI: 10.1145/3311790.3399628

1 author:

Jason Yalim
Arizona State University
44 PUBLICATIONS **215** CITATIONS

SEE PROFILE

Toward Dynamically Controlling Slurm's Classic Fairshare Algorithm

Jason Yalim*
yalim@asu.edu
School of Mathematical and Statistical Sciences & Research Computing, Arizona State University, Tempe, AZ, USA

Abstract

The popular SchedMD scheduler, Slurm, provides an internal algorithm for assigning priorities to user's high-performance computing (HPC) jobs based on previous usage. This algorithm, aptly named fairshare, is classically an exponential function of a user's usage history relative to the HPC population. This study explores an option HPC centers can take to increase the transparency of the classic fairshare algorithm and shows how usage and classic fairshare may be dynamically modeled using a simple differential equations approach.

CCS Concepts: • **Computer systems organization**; • **Software and its engineering** → *Software usability*; **Real-time schedulability**; • **Mathematics of computing**;

Keywords: fair scheduling, scheduler control, research computing outreach, dynamical systems,

ACM Reference Format:
Jason Yalim. 2020. Toward Dynamically Controlling Slurm's Classic Fairshare Algorithm. In *Practice and Experience in Advanced Research Computing (PEARC '20), July 26–30, 2020, Portland, OR, USA.* ACM, New York, NY, USA, 5 pages. https://doi.org/10.1145/3311790.3399628

1 Introduction and Background

Among other services and activities, the Arizona State University Research Computing (ASURC) core maintains ASU's flagship centralized high-performance computing (HPC) cluster, Agave. Agave's jobs are currently scheduled by version 19.05.6 of SchedMD's Slurm [2], and in September of 2019 resource allocation switched from a monthly core-hour allowance to multifactor driven by Slurm's classic fairshare algorithm. Jobs in the queue are scheduled based on a calculated multifactor priority, using an admin defined linear combination of various normalized measures of job and user attributes such as job size, job pending time, and historical usage (fairshare).

At first on Agave, Slurm's default configuration for the classic fairshare algorithm was left mostly untouched. However, the default led to a number of issues, notably that mildly active users would quickly have their fairshare drop from the maximum (one) to the minimum (zero). From observations, it became clear that usage on Agave followed a Pareto-like distribution, e.g. 80% of usage was attributable to 20% of users. This heavily skewed distribution implies that one has to be careful when using the mean as a basis for distributing priority. Otherwise users with significantly different usage histories may share the same priority penalties.

Fairly scheduling HPC resources is an active research topic and a good overview is given by an interesting study into modifying Slurm's fairshare to be based on energy consumption rather than core hours [1]. Many different algorithms have been proposed and implemented with vastly ranging complexity. This present study seeks to better understand and control Agave's simple implementation of Slurm's classic fairshare algorithm, which is calculated by the following code (line 1944 of `priority_multifactor.c`):

```
priority_fs = pow(2.0, -((usage_efctv/
  shares_norm) / damp_factor));
```

It will be shown that the product of the 'dampening factor' and the system's mean usage are fundamental for calculating fairshare on Agave. Specifically, the ratio of a user's usage with this product determines how many times a user's fairshare score has halved. Guided control of the mean usage or dampening factor allow for the control and modeling of fairshare, ultimately improving the HPC experience for power users, active researchers, and administrators.

Control was first instituted by introducing an artificial user to the cluster, but an alternative control implementation through the 'dampening factor' is also presented. Finally, controlling fairshare significantly simplifies its dynamical modeling, which permits a powerful teaching or extrapolation framework of a vital scheduler operation, and ideally increases research throughput.

When utilizing HPC resources, it is very important to understand the direct relationship between usage and job priority. Currently on Agave, a user's fairshare will halve for roughly every 10 000 core-hours of cluster compute time.

2 Controlling Fairshare

The classic fairshare algorithm ultimately refers to the instantaneous calculation of a user's fairshare $F \in [0, 1]$,

$$F = 2^{-U/(Sd)}, \tag{1}$$

where $U \in [0, 1]$ is a user's effective usage, $S \in [0, 1]$ is a user's normalized shares, and $d \in (0, \infty)$ is a global control parameter called the fairshare dampening factor. By default $d = 1$ and until a later section the dampening factor will be ignored. Equation (1) can be inferred from the code (line 1944 of `priority_multifactor.c`), from Slurm's classic fairshare algorithm documentation, and from previous literature [1, eq. 4].

On Agave, every user has their own account and exactly one share. Thus for N users, $S = 1/N$. The values F and U are defined uniquely for each user, while the normalized shares value S is uniform and thus equation (1) is instantaneously equivalent to (ignoring half-life decay effects)

$$F = 2^{-UN}. \tag{2}$$

A user's effective usage U is their normalized raw usage,

$$U = \frac{u}{\sum_{v \in \text{Users}} v}, \tag{3}$$

and therefore $UN = u/\bar{u}$, where $\bar{u}$ is the instantaneous mean raw usage of the system. This mean is used to signify the "fair use" of the system: when a user's raw usage $u = \bar{u}$, $F = 1/2$ and a user is using their "fair" allocation. Further, every multiple of $\bar{u}$ usage corresponds to an additional fairshare halving until ~$20\bar{u}$ which results in a reporting underflow and an effective fairshare of zero. That is, while fairshare is computed and stored as an eight byte floating point number (a double), it is reported to users with only six decimals of precision (as if it were a single).

From these expanded default definitions the number of hours required to have a users fairshare halve $\bar{u}$ is abstract and difficult to communicate to researchers. The mean usage $\bar{u}$ may be reported, but is dependent on time and a number of system parameters, like the raw usage half-life λ and the system's population N, complicating predictive extrapolations.

Fairshare may be controlled in two ways: by dynamically controlling an artificial user or the dampening factor d. The first way is more clear pedagogically and was first implemented on Agave, and thus will be examined first.

2.1 Controlling fairshare with an artificial user

By padding the sum in the normalization (3) with an artificial number w, an artificial user is "added" to the system's population. Looking at the fairshare calculation (2), the effective usage U population N are inversely proportional. Setting $F = 0.5$ implies solving for a targeted effective usage as a function of users in the system,

$$F = 0.5 = 2^{-UN} \implies UN = 1 \implies U = 1/N. \tag{4}$$

(Note this implies that "fair usage" is $1/N$ of the system.) To build on this, consider the case when no usage on the system has occurred except for one user who has used u core-hours. Then their effective usage is given by $U = u/(u + w)$, where w is the artificial target padding which by default does not exist ($w = 0$). The user immediately has an effective fairshare of 0 for any nonzero raw usage $u > 0$.

However, consider $w > 0$ and the objective of setting w such that after u core-hours the pioneer user has a fairshare $F = 0.5$. Then since $U = 1/N$,

$$\frac{u}{u + w} = \frac{1}{N} \implies uN = u + w \implies w = u(N - 1). \tag{5}$$

So for the pioneering user to have a fairshare of 0.5 after 10 000 core-hours, $w = (N - 1)10^4$, which is 25×10^6 core-hours for an example system with 2 501 users including the artificial user.

Looking at the Slurm documentation, one cannot simply use `sacctmgr` to inflate a control user's hours, as the only valid admin-set value for raw usage is 0 (admins may only reset a user's raw usage). So the source code requires modification or one may introduce a dummy, admin controlled user to enforce the padding.

2.1.1 The dummy account. Implementation through a dummy account requires the consideration of several factors: (a) an admin-only partition, to be only accessed by the dummy account, (b) the `TRES` on that partition must be set up to quickly and easily generate the raw usage padding w, and (c) the `TRES` on that partition must be set up to quickly and easily keep w constant over time. The `TRES` Slurm parameter is a factor multiplied by the core-time of an allocation, which despite the implication of the name, determines a user's raw usage. E.g., `TRES=100` implies that every one second of time is 100 core-seconds of raw usage. On Agave, the current admin-only partition is setup for a single node with 28 cores. Table 1 enumerates some example maintenance paddings under different scenarios. For fairshare halving target hours $u^\star = 10\,000$ and total users $N = 2501$, the raw usage padding requirement is $w = 10\,000 \times 2\,500$ or exactly 25×10^6 hours. This was implemented for dummy admin-user `rcjdoeuser` on Agave successfully by running the following job on the admin partition as the user with `TRES` for the CPU set to 1.5×10^9:

```
sbatch -wrap="sleep 60; exit" -t 1 -p rcjdoeuser
```

2.1.2 Dummy user upkeep. Agave's Slurm is currently configured to 'forget' raw usage at an exponentially decreasing rate with a half-life of seven days. Empirically or code wise, the exact implementation of the decay must be determined, but observational evidence suggests that the raw usage is updated on a five-to-ten minute time interval and may be modeled continuously as a function of time in days $u(t)$, with decay rate λ inferred from the one week half-life.

Total Users	Target Hours ($u^\star$)	Raw Usage Padding (seconds)	Decay First Day no Upkeep (seconds)
2 501	10 000	90×10^9	$\approx 8.485 \times 10^9$
2 501	20 000	180×10^9	$\approx 16.970 \times 10^9$
3 001	10 000	108×10^9	$\approx 10.182 \times 10^9$
5 001	10 000	180×10^9	$\approx 16.970 \times 10^9$

Table 1. Example paddings and daily decays for specified target hours and user populations. Note the total users includes the dummy.

That is,

$$u(t) = u(0)2^{-t/7}. \tag{6}$$

Thus, after a one day, the initial raw usage $u(0)$ decays by $D = u(0)\,(1 - 2^{-1/7}) \approx 0.0943\,u(0)$, corresponding to roughly 9.5% of decay per day. To keep the dummy user's padding w roughly constant on a daily time scale, this decay must be mitigated by rerunning the following job as the user every 24 hours with the CPU `TRES` set to $D = 8\,484\,870.2484$ (for target $u^\star = 10\,000$ hours with $N = 2\,501$ users):

```
sbatch -wrap="sleep 1; exit" -t 00:01 -p rcjdoeuser
```

Using the `TRES` originally set (1.5×10^9 per CPU second), the dummy user's padding w can be kept roughly constant on a four-hour time window. From the target $w = 90 \times 10^9$, it takes roughly four hours and five minutes for w to decay 1.5×10^9 raw seconds. Thus by checking every four hours if the decay has reached this threshold before running an upkeep job, w may be kept within 2% of the target. This is illustrated in figure 1 in black.

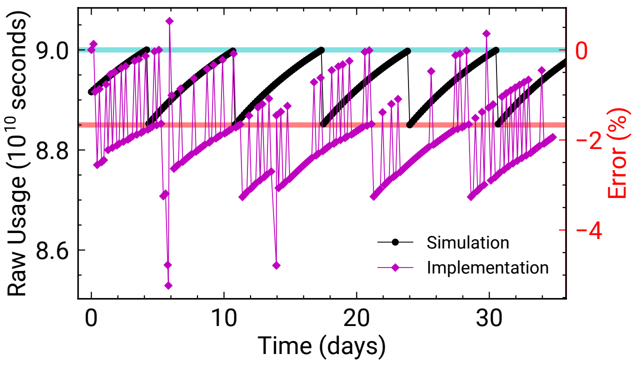

Figure 1. Simulation (black circles) and implementation (magenta diamonds) of dummy user upkeep mechanism with raw usage in seconds illustrated every four hours. By running a check every four hours, the simulated dummy user's raw usage stays within 2% of the target, but in practice slippage was closer to 5%.

2.1.3 Results. Recall that equation (2) and (3) imply that the system's mean raw usage $\bar{u}$ is the system's effective fairshare halving usage. Note as well that this value may be sampled from a single user, that is $F = 2^{-UN/d} = 2^{-u/\bar{u}}$ implies $\bar{u} = u/\log_2 F$. As an example, consider a Slurm implementation with a user that has a fairshare score of 0.131 575 after 29 798.8 compute hours. Then $u^\star = 10\,182.3$ hours, implying that any user on the system must compute at least 10 182.3 hours to have their fairshare decay from 1 to 0.5. See table 2 for additional examples from data taken from Agave after the admin controlled dummy user was activated. Since job submission is a continuous process and the scheduler does not instantly update a user's fairshare, this measure varies over users by a small amount. Keeping track of such a measure across a subset of active users may provide a meaningful heartbeat on the system's relationship between fairshare and compute time.

Users	Fairshare	Raw Usage (hours)	$\bar{u}$
u0	0.000 043	147 787.583	10 188.520
u1	0.000 169	127 655.753	10 187.449
u2	0.000 367	116 211.920	10 183.369
u3	0.001 165	99 399.131	10 199.538
u4	0.003 002	85 328.210	10 182.534
u5	0.026 259	53 479.864	10 184.615
u6	0.057 793	41 881.796	10 182.881
u7	0.105 710	33 016.212	10 184.480
u8	0.127 456	30 261.039	10 182.290
u9	0.131 575	29 793.799	10 182.284
		MEAN:	10 188.681

Table 2. Snapshot of the top ten users on Agave (ignoring dummy admin user rcjdoeuser) with associated $u^\star$ calculations post implementation.

Figure 1 compares a simulation of dummy user upkeep in black to a month long implementation in magenta. As can be seen, the implementation followed the simulation fairly well, slipping at most by about 5% when balancing long-tail events such as a system outages (day 6) or controlling node reset (day 14).

2.2 Controlling fairshare with a dampening factor

Recall that equation (1) has a dampening factor $d = 1$ that up until now was ignored. Rather than create a user and provide upkeep through periodic one-second jobs, d may be set based on system usage and admin specified target halving

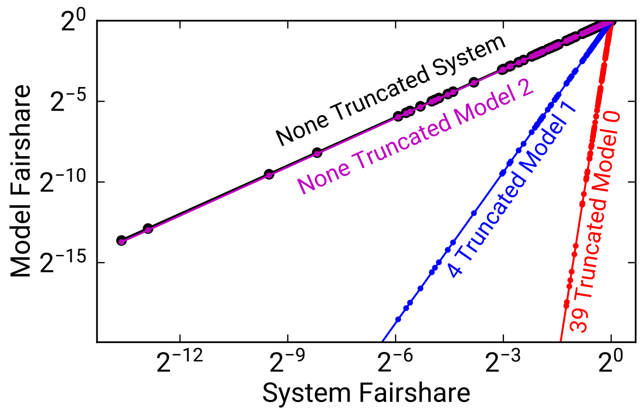

Figure 2. Comparison of system implementation (dummy user controlled) fairshare (black) compared to three different models on the same usage data (ignoring the dummy user) with $d = 1$ (Model 0, red), $d = N/M$ (Model 1, blue), and $d = u^\star/\bar{u}$ (Model 2, magenta). The number of users with underflowed fairshare scores is indicated.

compute time $u^\star$. Note $F = 0.5 = 2^{-u^\star/(d\bar{u})}$ implies that

$$d = u^\star/\bar{u}. \tag{7}$$

Further, by periodically computing $\bar{u}$, the fairshare dampening factor d may be updated with the admin command:

```
scontrol fsdampeningfactor $computed_d
```

Figure 2 illustrates four competing models, with y–axis truncated to 2^{-20}, where fairshare F reporting underflows to zero. Each point represents a user's fairshare within the system relative to a competing model. The system's (dummy account implemented) distribution of fairshare is plotted in black relative to itself, and no user has a fairshare of zero. Model 0 (red) would be the fairshare distribution without a dummy user and no controls added to the system, and as is illustrated, 39 users would have a fairshare of zero. The 39th user had 15 000 core-hours of usage versus the top user with 140 000, yet the default implementation would prioritize both users' jobs equally in the lens of previous usage! This is a result of the system's natural mean usage $\bar{u} \approx 700$, which further results in 145 users with $F \leq 0.5$. Model 1 sets $d = N/M$, where M is the number of active users (raw usage greater than zero) on the system. Model 1 (blue) still results in four users with the degenerate zero fairshare, and eighty-five users with $F \leq 0.5$. Finally, Model 2 is in magenta with $d = u^\star/\bar{u}$, and is nearly indistinguishable from the dummy system implementation. Both Model 2 and the system have forty-five users with $F \leq 0.5$, and no users with truncated fairshare scores.

A quick audit of the Slurm source code shows that the mathematics are indeed defined this way (see line 1944 in `priority_multifactor.c`). However, there are a couple of issues with the current Slurm (v19.05.6 on ASU's Agave) implementation. Remarkably, `scontrol` only reads the dampening factor as a sixteen bit unsigned integer despite later casting it as a long double when computing fairshare. Further in our practice, `scontrol` would not truly modify the dampening factor. Despite claims from the documentation and the appropriate info logs showing that an appropriate RPC request had occurred when `scontrol` was called, users' fairshare scores were unaffected by anything other than a direct and undesirable update of `/etc/slurm/slurm.conf` (by setting `FairShareDampeningFactor`, another integer!). In testing, after updating `slurm.conf` and running `scontrol reconfig`, users' fairshares would update immediately. Also note that since the dampening factor is currently forced to be an integer at read time, the precision of the method decreases as the mean usage $\bar{u}$ approaches target usage $u^\star$. Regardless, a periodic, floating-point implementation of this method would provide a very powerful and precise method for the control of fairshare.

3 Dynamic Properties of Fairshare

Not much attention has been paid to the behavior of fairshare over time. When controlling fairshare, the product of the dampening factor and the mean usage $d \cdot \bar{u}$ is kept close to an admin defined number of target hours, e.g. $u^\star = 10\,000$ core hours. By keeping this relatively constant over time, the dynamic behavior of fairshare becomes much more straightforward to model. This section assumes that $d \cdot \bar{u}$ is constant and equivalent to $u^\star = 10\,000$ core hours.

More can be done to educate users on the nature of Slurm job priority, as fairshare evolves deterministically based on a piecewise constant function of user core usage. To build the model, consider a user's raw usage to be a continuous in time process. Then the raw usage changes instantaneously based on the flux of usage, i.e. by the difference of outflows from inflows. Raw usage 'outflows' at a rate proportional to current usage and 'inflows' at a piecewise constant rate determined by the associated hardware allocation(s), thus,

$$\dot{u}(t) = c(t) - \lambda u(t), \qquad u(t_0) = u_0, \tag{8}$$

where u is the raw usage as a function of time t, $\dot{u}$ is the time derivative of u, c is a constant piecewise function of core usage, and λ is the usage half-life. The exact solution to this first order initial value problem for each piecewise constant interval of $c(t) = c$ is given by

$$u(t) = \frac{c}{\lambda} + \left(u_0 - \frac{c}{\lambda}\right)\exp(-\lambda[t - t_0]). \tag{9}$$

From equation (8) and the solution (9) it's clear that c/λ is a stable solution for the raw usage and thus that raw usage is bounded: $u \to c/\lambda$ as $t \to \infty$ for all $c \geq 0$.

With this solution, coupled with the control of fairshare to halve every multiple of $u^\star = d\bar{u}$, it becomes easy to model

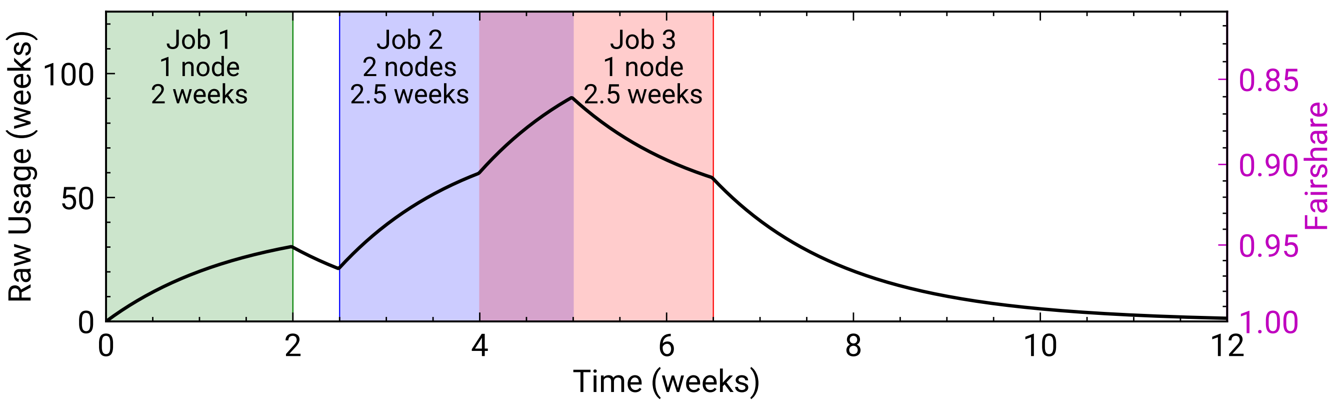

Figure 3. Simulation of a user running jobs. Jobs 1 and 3 use one node (28 cores) while the second uses two.

fairshare as a function of time,

$$F(t) = 2^{-u(t)/u^\star}.$$

Figure 3 illustrates a simulation of a user running three jobs, with $u^\star = 10^4$ and a usage half-life of one week. That is,

$$c(t) = \begin{cases} 28 & \text{if } t \in [0,2] \text{ weeks,} \\ 56 & \text{if } t \in [2.5,4) \text{ weeks,} \\ 84 & \text{if } t \in [4,5) \text{ weeks,} \\ 28 & \text{if } t \in [5,6.5] \text{ weeks,} \\ 0 & \text{else.} \end{cases}$$

Raw usage grows no more than sublinearly while a constant core allocation is running, and quickly decays thereafter. In this simulation, the user's fairshare never goes below 0.85, but clearly increased simultaneous usage is necessary to bring fairshare close to zero. Note that job 3 illustrates how running a job doesn't necessarily imply that raw usage will increase over time, a nice demonstration of the power of exponential decay and how usage is characterized by the stable solution c/λ.

4 Conclusion

Previous to this study, fairshare on the Arizona State University's Agave cluster had an unknown and uncontrolled elasticity. Control was introduced by an admin set parameter: the number of core-hours required to halve a user's fairshare. One control method implemented an admin controlled dummy user to pad the reported usage of the system, utilizing six seconds of compute time on an admin node per day. A second method utilized an additional parameter to control fairshare without any compute time, and the upsides and downsides of both were compared to two other implementations, including an uncontrolled model.

By controlling fairshare, three principal benefits are immediate. The first is transparency and consistency to users. Fairshare halving every 10 000 core-hours is a significant communication improvement relative to the default, and requires significantly less mathematical overhead in presentations. Secondly, without controlling fairshare, power users would have nonunique fairshare scores (zero). This causes obvious frustration to users that share zero fairshare yet have computed orders of magnitude less than their power-user peers. Thus, the new controlled method is more clear to all researchers and more fair to those who use it most. Finally, by specifying the target number of hours for users' fairshare scores to halve, a dynamical model of fairshare representing actual cluster behavior becomes simple. Without control, the model would depend on historical and extrapolated mean usage data. With control, fairshare becomes a simple function of core usage.

The dynamical model of usage and fairshare illustrate how fairshare may be further communicated to users. With a robust model for usage and fairshare, methods to communicate multifactor priority also improve. Upon job submission, multifactor priorities may be reported back to the user, with a simple report on expected fairshare when the last job in the queue completes. Additionally, an extrapolation tool via an interactive (web) application could ask for a user's input on job parameters, returning a report of priorities and even a chart of expected usage and fairshare over time. The control of fairshare leads to researchers being better able to plan and predict their usage of high-performance computing resources.

References

[1] Y. Georgiou, D. Glesser, K. Rzadca, and D. Trystram. 2015. A Scheduler-Level Incentive Mechanism for Energy Efficiency in HPC. In *2015 15th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*. 617–626.

[2] Andy B. Yoo, Morris A. Jette, and Mark Grondona. 2003. SLURM: Simple Linux Utility for Resource Management. In *Job Scheduling Strategies for Parallel Processing*, Dror Feitelson, Larry Rudolph, and Uwe Schwiegelshohn (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 44–60.