Slurm simulations for investigating Fair Share settings

Dustin Lang

January 7, 2021

1 Simulation

I wrote a simple notebook (`slurm-sim.ipynb`) that tries to reproduce the fair-share and priority queuing algorithms we have running on Symmetry.

In this simulation, I'm assuming every job takes exactly one hour and uses a single node. We'll simulate a case where one user has a lot of jobs queued, and then, after a week, another user submits a bunch of jobs. Specifically: User 1 has submitted enough jobs to keep the whole cluster for 2 weeks. Then, after 1 week, User 2 submits enough jobs to keep the cluster busy for a week.

Configuration settings Currently, on Symmetry we use the `Priority/multifactor` plugin to compute job priorities. The way we have it configured, a job's priority is the weighted sum of an Age factor (how long the job has been queued) and a Fair-share factor (which is large if the user has not been using the cluster, and decreases as the user uses more and more compute time). The Age and Fair-share factors are each between 0 and 1.

Parameters:

`PriorityMaxAge = 10-0` This settings (10 days) determines how long it takes the Age priority factor to go from 0.0 to 1.0. That is, the Age priority factor increases by 0.1 each day, up to a max of ten days.

`PriorityDecayHalflife = 7-0` This setting determines how quickly a user's past usage is ignored in computing the Fair-share factor.

`FairShareDampeningFactor = 20` The Fair-share algorithm computes a user's CPU use relative to what you would get if you split the cluster evenly between the total number of users. On Symmetry, we have over 150 user accounts, but usually only a handful of people active at any time, so the default makes a user's fair share tiny. This factor corrects for that, effectively scaling each user's even share up by this factor.

`PriorityWeightAge = 10000` This determines how much the Age factor is scaled to compute the priority. A larger value makes Age (time in queue) more important.

`PriorityWeightFairShare = 4000` This determines how much the Fair-share factor is scaled to compute the priority. A larger value makes a user's past cluster (over)use more important.

1.1 Run 1

The first run is with the current values as given above.

Figure 1 below shows the Age factor for each user's jobs. Recall that User 1 submits a bunch of jobs a time 0, and User 2 submits a bunch of jobs after 1 week. Each user's jobs gain Age priority as time goes on, up to the max after 10 days.

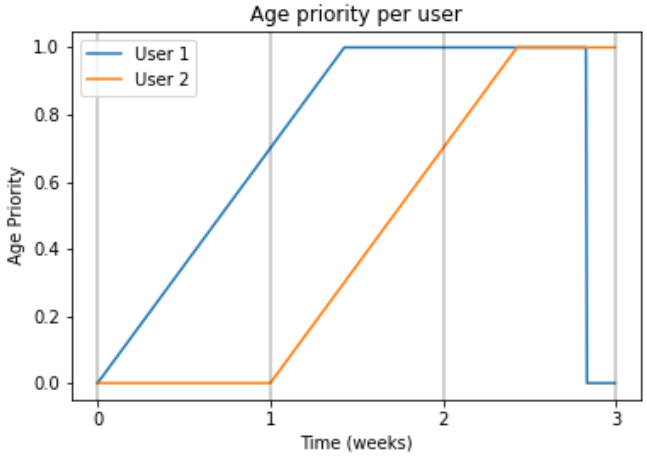

Figure 1: Run 1: Maximum Age factor for jobs submitted by each user.

Figure 2 below shows the Fair-share factor for each user. If the user has not had any jobs running, that user's Fair-share factor will be 1, and as the user dominates the cluster over time, it drops to zero. Here, we see that as User 1 is using the entire cluster for the first week, the Fair-share factor drops quickly toward zero. Once User 2's jobs start running, that user's Fair-share also drops.

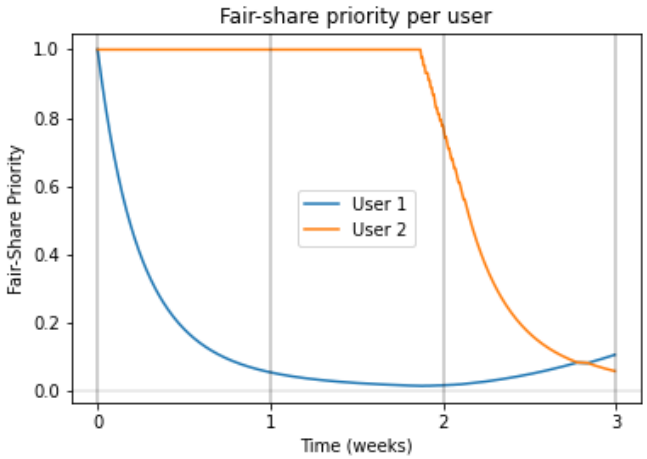

Figure 2: Run 1: Fair-share factor for each user.

Figure 3 below shows the total priority: a weighted sum of the Age factor and Fair-share factor. At the end of the first week, User 1's Age priority has reached $0.7 \times 10,000$, and Fair-share priority is small. User 2's jobs then begin with no Age priority and the maximum Fair-share priority, which, because of the relatively smaller weighting, gives a smaller total weighting of only 4000. Over the next three days, User 1 continues to get all the compute time, until that user's jobs hit the 10-day max for accumulating Age priority. Then User 2's jobs continue accumulating Age priority, eventually catching up the ~ 3000-point gap over about three days. On day ~ 6, User 2's jobs finally start running.

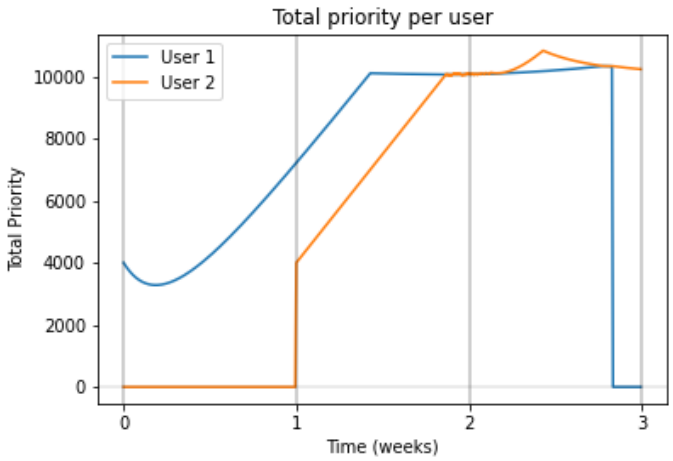

Figure 3: Run 1: Total priority for each user's jobs.

Figure 4 shows the scheduler's behaviour. As mentioned above, User 1's jobs continue to get scheduled until nearly a full week after User 2's jobs are submitted.

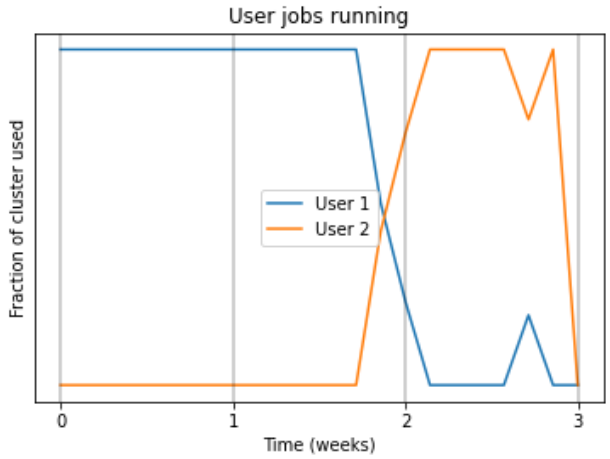

Figure 4: Run 1: Scheduler behaviour: which user's jobs were run over time.

1.2 Run 2

It seems that our weighting of the Fair-share factor is too small relative to the Age factor. In this run, we set `PriorityWeightFairShare = 10000`.

The Age factor plot is exactly the same as before. The Fair-share factor is similar; we will show it later. The driving change here is the total priority, shown below in Figure 5. Now, when User 2's jobs are submitted at the 1-week mark, that user's Fair-share factor of 1, times the increased weight factor of $10,000$, is enough for it to beat the priority accumulated by User 1's long wait. User 2's jobs begin to run immediately. However, when that happens, User 2's Fair-share drops quickly, until its priority drops below that of User 1. Then, both users' jobs are run, which reduces their Fair-shares respectively. Since User 2 is in a steeper portion of the Fair-share curve, that user gets a relatively smaller share of the cluster.

At the 10-day mark, the dynamics change again, because User 1's jobs stop accumulating Age priority, and User 2 starts getting a larger share of the cluster.

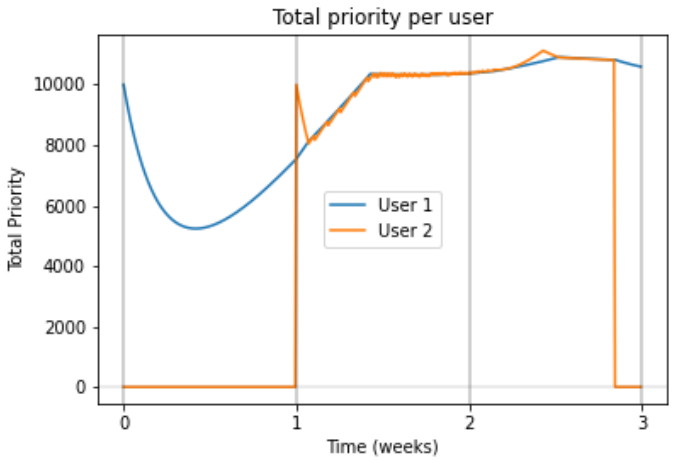

Figure 5: Run 2: Total priority for each user's jobs.

Figure 6 shows the jobs run for each user. As discussed above, the dynamics are surprisingly complicated!

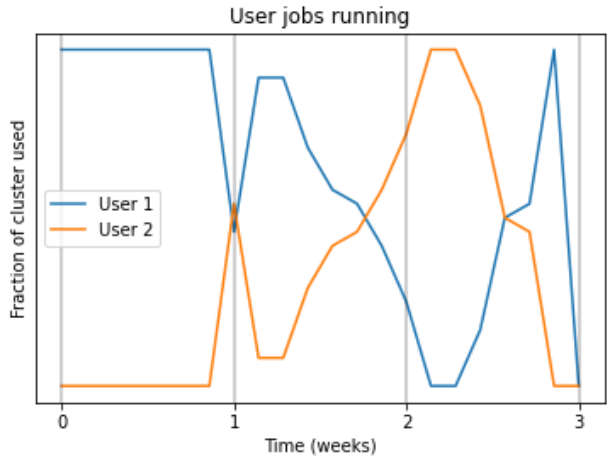

Figure 6: Run 2: Scheduler behaviour: which user's jobs were run over time.

For completeness, Figure 7 shows the Fair-share factor for each user.

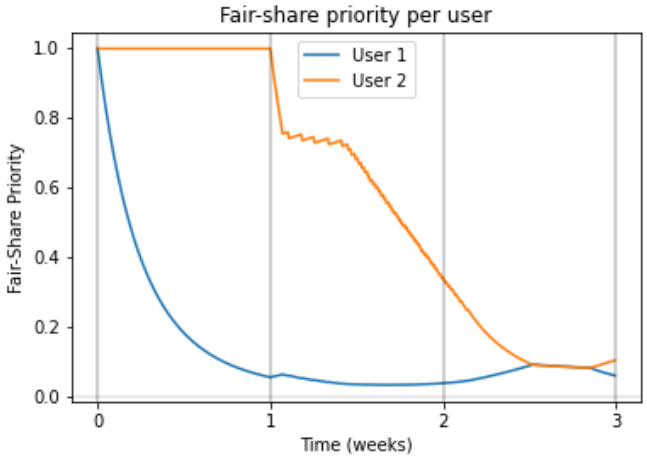

Figure 7: Run 2: Fair-share factor for each user.

2 Conclusions

I think we should increase the `PriorityWeightFairShare` factor to at least `10000`. We *may* want to decrease the `PriorityMaxAge` so that jobs max out their Age priority accumulation sooner. If we do that, we may want to decrease the `PriorityWeightAge` factor so that jobs accumulate the same 1000 priority points per day.