Chapter 9
Overview of Results

Chapter 8 outlined 17 cases simulated using dSim. Both the traditional, fair-share scheduling algorithm and the Rawlsian Fair scheduling algorithm developed for our work were used in these simulations. Table 9.1 outlines the different classes of simulations and their purposes.

9.1 Results Recap

The results reveal that the Rawlsian Fair scheduling algorithm both significantly decreased the times-in-system of the users with the lowest relative numbers of tasks and decreased the time-in-system variance for every user. This section offers an overview of the delays incurred by the users in the simulations. Data analysis is conducted per class of requests, as is outlined in Sect. 1.2. Data analysis for the Least Benefited User (LBU) will be considered as defined in Rawls (2001).

The Rawlsian Fair scheduling algorithm uses seniority as well as the traditional parameters of load and priority. Seniority is updated by the scheduler according to the changes that occur as tasks enter and leave the system. Seniority is included to decrease the time-in-system experienced by users with low numbers of tasks. In fair-share scheduling, the delay is normalized across the users. This effect is revealed in our results. In Rawlsian Fair scheduling, however, seniority moves tasks to the front of the queue: the tasks in the queue are ordered first by seniority and then by arrival time, so that of the tasks that share the highest seniority value, the first one in the queue is executed first. As mentioned previously, the Rawlsian Fair scheduler uses buckets to calculate seniority. Seniority is calculated based on the current state of the bucket (i.e. the number of tasks in the bucket), and all of the tasks in a given bucket are finished before the scheduler moves on to the next bucket (Fig. 9.1).

The Placement Counter (PC) dictates the target bucket location: PC = 1 indicates that the tasks are destined for Bucket 1, PC = 2 indicates that the tasks are destined for Bucket 2, and so on. Bucket History (BH) size is another variable. Seniority is

A. Sedighi, M. Smith, *Fair Scheduling in High Performance Computing Environments*, https://doi.org/10.1007/978-3-030-14568-2_9

Table 9.1 Summary of simulation types and expected outcome

Workload classification	Relevance
Class A	Intermittent workloads occur when users submit jobs throughout the day. These simulations describe situations in which complementary users of different load compete for resources. It considers how performance is affected as the number of users increase.
Class B	These simulations describe situations in which steady workloads compete for resources with intermittent workloads.
Class C	These simulations describe situations in which one user is competing for resources for a one-time burst submission against larger workloads.
Class D	These simulations describe situations in which small but steady workloads compete with larger workloads.

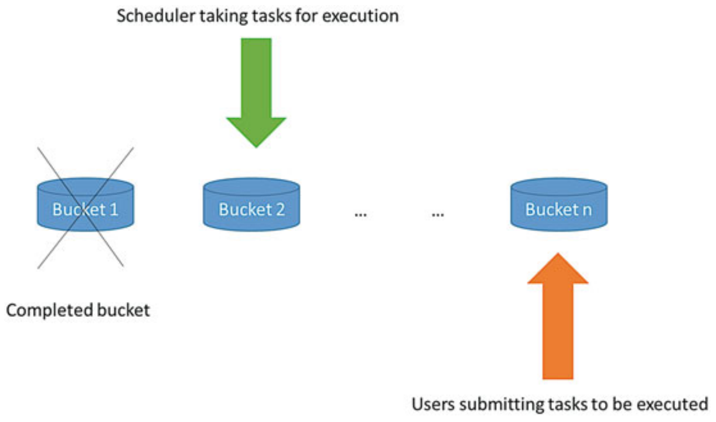
Scheduler taking tasks for execution
Bucket 1
Bucket 2
...
...
Bucket n
Completed bucket
Users submitting tasks to be executed

Fig. 9.1 Illustration of the role of buckets in placing (via the Placement Counter) and executing tasks

calculated across a number of buckets denoted by the BH value. BH = 1 (Fig. 9.1) means that seniority is calculated based on the parameters of the current bucket, i.e. the oldest bucket waiting to be drained.

Once all of the tasks in a given bucket have been scheduled for execution, the bucket is deleted and the scheduler moves on to the next bucket. In each of the simulations, the number of processing nodes (or CPUs) was held constant at 100, meaning that 100 tasks could be processed simultaneously. Each task takes 1000 ms to complete, so if 1000 tasks were submitted and the system was able to process 100 at a time, it would take 10 s to complete all of the tasks in a real-world scenario.

In BH = 2 scenarios (Fig. 9.2), the tasks are placed in their respective buckets, but seniority is calculated based on the parameters of two buckets. In other words, the total number of pending tasks and the total number of submitted tasks are based on the values spanning two buckets.

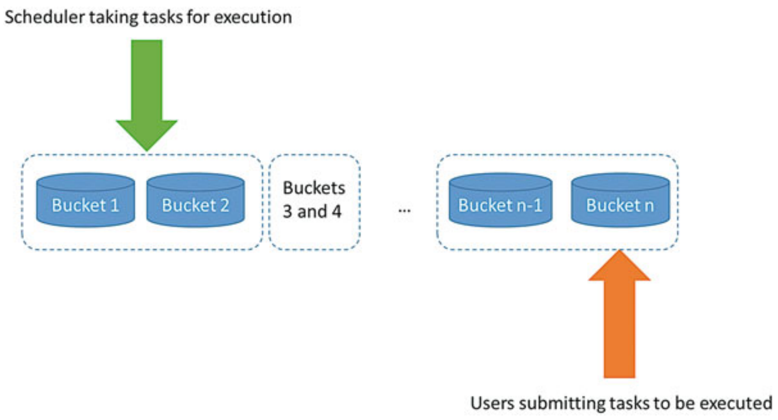
Scheduler taking tasks for execution
Bucket 1
Bucket 2
Buckets 3 and 4
...
Bucket n-1
Bucket n
Users submitting tasks to be executed

Fig. 9.2 Scheduling scenario with BH = 2

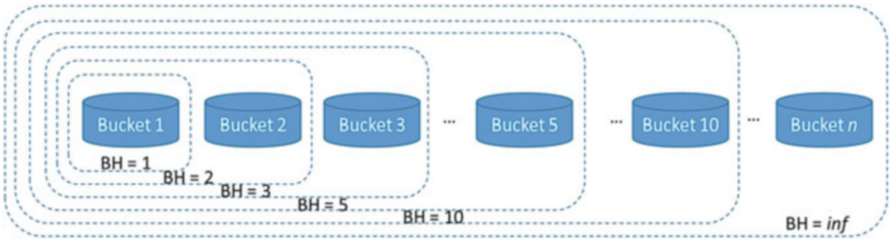

Fig. 9.3 Calculating seniority based on BH value

Simulations were run for different workloads and six different BH sizes:

$$BH = \{1, 2, 3, 5, 10, inf\} \tag{9.1}$$

$Bh = inf$ is a special case in which seniority is calculated by taking into consideration the entire submission history of a given user (Fig. 9.3). This scenario consider a system never forgets. While smaller BH values can suffer from local maxima problems (see the results of simulations 3 and 4), local maxima problems are resolved when $Bh = inf$. A side effect, however, is that disproportionately large jobs tend to suffer.

Although the aim of the Rawlsian Fair scheduler is to improve the fairness of the system, as the FUD conjecture suggests, one of the other parameters will suffer as a result of this change.

Our results revealed that although the Rawlsian Fair scheduler improved fairness of the system, i.e. smaller users experienced smaller times-in-system—the overall dynamicity of the system decreased. To calculate seniority, the incoming tasks were placed in buckets (or queues) (see Sect. 6.2) in the order in which they arrived. In cases in which multiple users had the same seniority level (e.g. in simulation 6), the

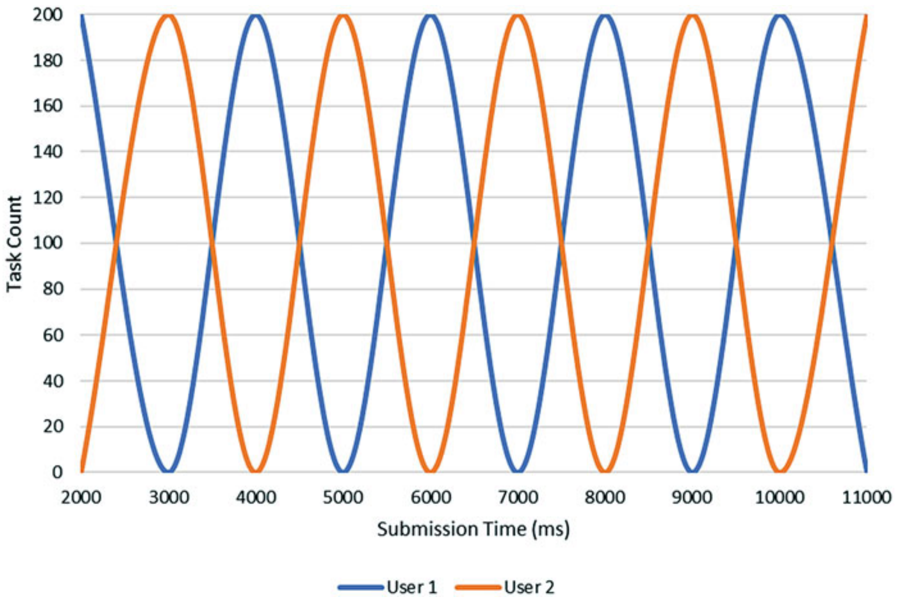

Fig. 9.4 Task submission pattern for simulation 1

tasks were executed in a first-come-first-served fashion. The same applies for users with the same number of overall tasks, as one would expect. In cases in which two or more users submitted the same number of tasks at the same time, the tasks were executed in the order in which they arrived in the queue.

9.1.1 Simulation 1 Analysis and Results

Simulation 1 was a "smoke simulation" to make sure that our system worked as expected end-to-end. Two users submitted 1000 tasks each at 200 tasks/PC (Fig. 9.4).

Each user submitted their next task after the other user had submitted theirs, creating a sine–cosine submission pattern. We expected that the delays incurred for each task would be similar for Rawlsian Fair with BH = *inf* and fair-share (Figs. 9.5, 9.6).

First, we compare the simulation results for fair-share and Rawlsian Fair with a bucket history of infinity (*inf*). Rawlsian Fair's stepwise task-delay line is the result of the bucketing feature for task submission. Tasks are sorted into buckets and executed in the order in which they arrive. There were 100 worker nodes, so user 1's first 100 tasks were sorted into bucket 1, and so on.

The task delays obtained with each scheduling method were complimentary because there were only two users: an increase in the delay experienced by one user meant a decrease in the delay experienced by the other (Fig. 9.7).

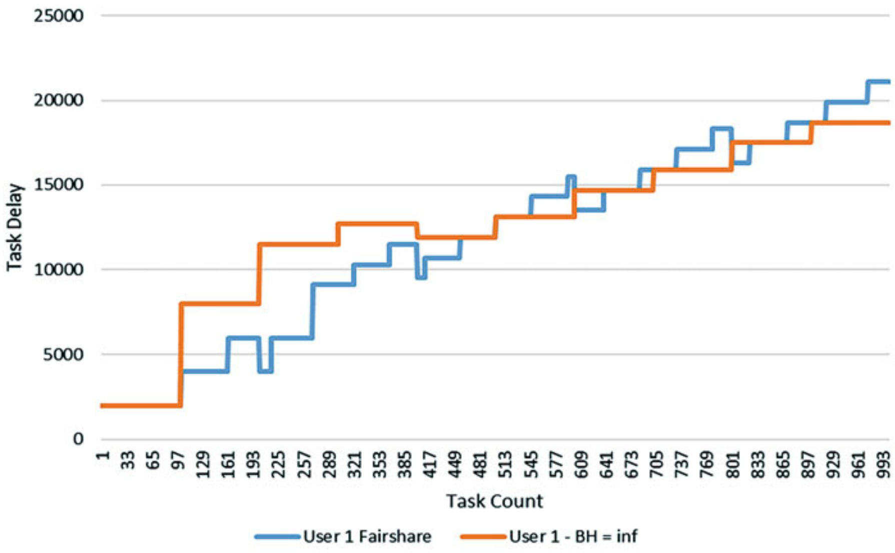

Fig. 9.5 User 1's task delay in simulation 1 with BH = *inf*

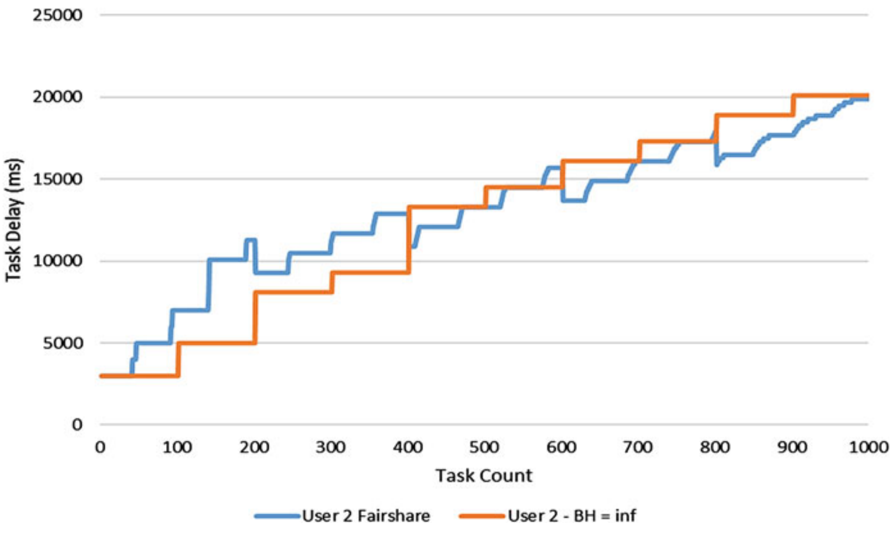

Fig. 9.6 User 2's task delay in simulation 1 with BH = *inf*

The goal in this simulation was to evaluate how close Rawslian Fair scheduling gets to traditional fair-share when two users share the same job-submission profile. The results for all of the BH sizes in simulation 1 are also presented (Figs. 9.8, 9.9), and they show a similar delay pattern.

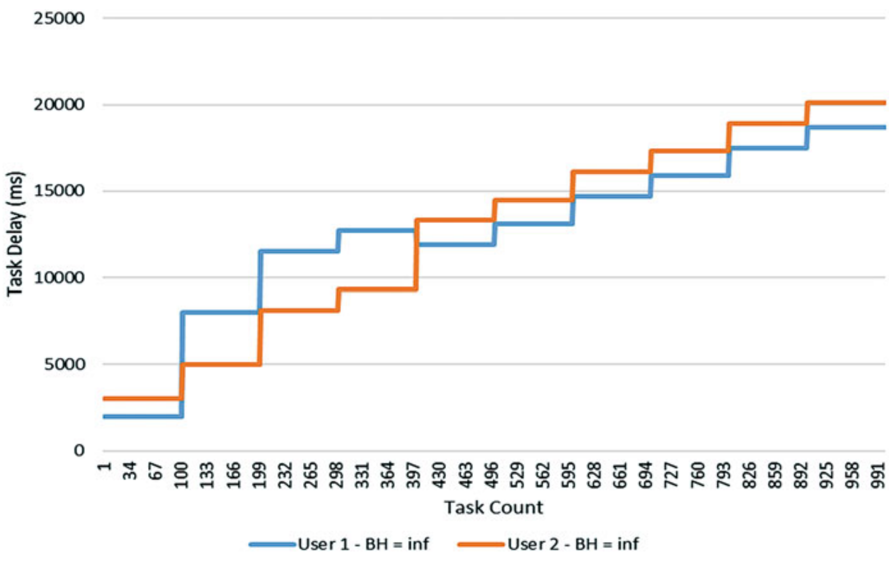

Fig. 9.7 Task delay comparison of both users for simulation 1

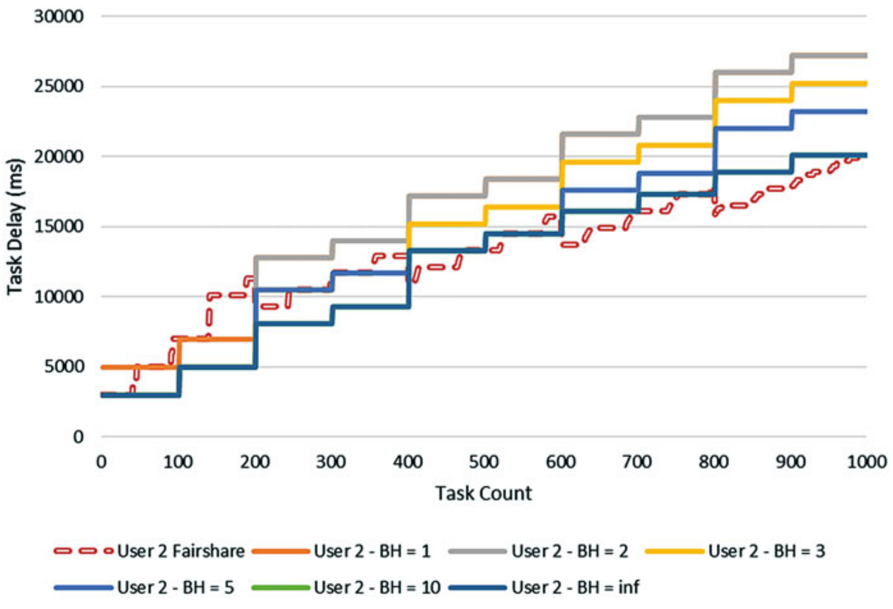

Fig. 9.8 User 2's task delay for all BH sizes in simulation 1

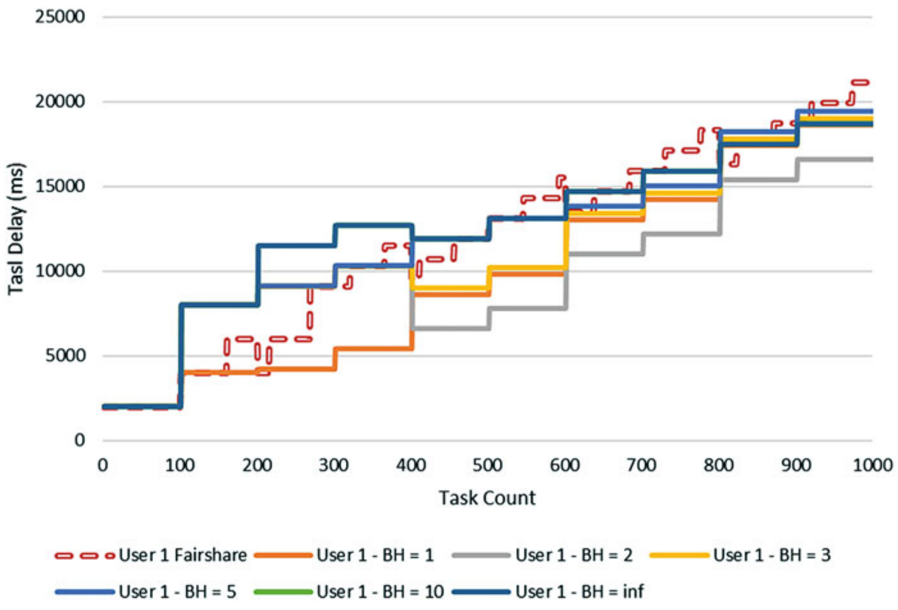

Fig. 9.9 User 1's task delay for all BH sizes in simulation 1

9.2 Lessons Learned

In the calculation of seniority, incoming tasks were timeboxed into "buckets" (see Sect. 6.2) denoted by the placement counter (PC) variable. The granularity of the placement counter had a profound effect on the performance levels of the less senior tasks. While the PC granularity had to be set at 1000 ms (1 s) (equal to the task-execution size) for seniority to be calculated, this had the side effect of delaying less senior tasks by their position within each bucket. This delay resulted from the action of the First-Come-First-Serve (FCFS) algorithm, which was used to take equal-seniority tasks off the queue. Even though the tasks were submitted at the "same time," even a fraction of a second can affect the arrival time. dSim is aware of communication delays and uses them in setting the arrival time of each task.

Regardless of the PC value, however, Rawlsian Fair scheduler performed equal to or better than fair-share overall for Class A and Class B workloads. A comparison of the results of simulations 5 and 9 further demonstrates this point. While the number of tasks was the same in both simulations, simulation 9 had more users. As the number of users increased, so did the effectiveness of Rawlsian Fair.

Rawlsian Fair performed universally better with Class C and Class D workloads. This is apparent when simulation 5 is compared to simulations 13 and 17. The tasks submitted by a single, the large user for simulation 5, are distributed among 5 users in simulation 9, 11 users in simulation 13, and 20 users in simulation 17. The outcome of each simulation is the same for the smallest user, however, demonstrating the effect of Rawlsian Fair's use of seniority.

The number of tasks distributed among the users is held constant across each of the simulations, including simulations 10 and above, and the effect of Rawlsian Fair becomes increasingly apparent as the number of users increases. Seniority consistently benefits disproportionately small users, but because Rawlsian Fair picks smaller users from sets of larger users, this effect becomes more pronounced as the number of users increases.

9.3 Simulation Results and Analysis

The following chapters review the results of the various simulation runs. We generated over 600,000 data points,[1] and we illustrate our key findings. Given the sheer number of data points, every data point cannot be independently considered.

Some of the simulation results are grouped into pairs because while the number of tasks is the same for the members of each pair, one member of each pair demonstrates the effect of Rawlsian Fair on the smallest user, and the other member demonstrates the effect of seniority as the number of users increases.

[1] 17 tests, 7 different scheduling algorithms, and 5000+ tasks per run