Chapter 6
Simulation and Methodology

As outlined in 1.3, our work assesses the performance characteristics of a multi-criteria scheduler that uses seniority as well as priority and load is to make decisions. It does so by using various simulation models to test the scheduling algorithm. Later in this book, the simulator used to conduct these simulations (dSim) is introduced.

Previous chapter described a shortcoming of fair-share scheduling techniques: they achieve long-term fairness at the expense of short-term fairness. Evidence of this includes cases in which system dynamics were altered by small and frequent requests. Further evidence is offered by Sedighi et al. (2014). Current scheduling techniques consider priority and load requirement in making scheduling decisions (Baker and Trietsch 2013), but do not employ a primary variable indicating the time that a task has spent in the system. Tasks are queued—or placed in multiple queues if they have different priority levels—and they are processed in the order in which they arrive. Queues are one-dimensional constructs in which load requirement is the primary independent variable. Priority queues are two-dimensional constructs in which priority is the second independent variable.

Our work introduces an additional independent variable that tracks the time each task has spent in the system. This variable is called "Seniority" (S); it is additive and increases in value as one would expect: a net-positive increase in seniority is calculated for each task after every scheduling cycle, and this new value is used to reorder the standings of the tasks prior to the next scheduling cycle. The task with the highest seniority is scheduled first for a given duple of priority and load set of tasks.

The rest of this chapter describes how seniority is calculated based on the mathematical model introduced in Chap. 5. The experiments conducted to assess the performance characteristics of the scheduler, which uses three independent variables: priority, load, and seniority will be introduced in Chap. 8.

A. Sedighi, M. Smith, *Fair Scheduling in High Performance Computing Environments*, https://doi.org/10.1007/978-3-030-14568-2_6

6.1 Calculating Seniority

Seniority (S), is a dynamic, time-dependent value calculated in part by the time that the task has been in the system relative to the current state of the system and to the other users in the system—i.e. the "time-in-system" (Baker and Trietsch 2013). The weight of seniority is determined by the fairness factor (α). Seniority is an additive value: every time it is calculated, it is added to the seniority value that the task already holds. In essence, the longer a task has been pending (i.e. the larger is its time-in-system), the higher its seniority is. This relationship allows seniority to serve as a surrogate for time-in-system.

Change in seniority, S, is a function of priority and load,

$$\frac{dS}{dt} = \Delta S(t) = \alpha^* P^* L \tag{6.1a}$$

while total seniority depends on time-in-system (t):

$$S_{Total} = \sum_{t=0}^{t=current} \Delta S(t) \tag{6.1b}$$

where P represents the priority of the task, and L represents the load the task will put on the system. Both values are entered by the user when the task is submitted and (for our purposes) both remain constant while the task is in the system. Both values range from 0 to 1: 0 represents the lowest priority and the lowest computational load, and 1 represents the highest priority and the highest computational load. Equation 6.1b illustrates how a task's seniority depends on the time that the task has been in the system. Seniority is calculated in discrete time intervals, but it is additive and continues to increase across the lifetime of the task.

The fairness factor, α, is calculated by taking into account the rate of pending tasks for a user relative to all of the users:

$$\alpha_u^{-1}(t) = \frac{\xi(t)}{\sum_u \xi(t)} \qquad \#\,\text{Rawlsian} \tag{6.2}$$

Equation (5.2) implements Rawls's philosophy of fairness, with $\xi(t)$ representing the pending tasks for a given user at time t and $\sum_u \xi(t)$ representing the pending tasks for all of the users in the system. This equation causes the user with the least number of pending tasks to be moved to the front of the queue. This method resembles the shortest processing time (SPT) scheduling algorithm (Baker and Trietsch 2013), which has been shown to have the lowest makespan when scheduling jobs. In essence, the fairness factor makes an implicit assumption about the future task count of each user, and this assumption is adjusted if it is proven wrong in subsequent scheduling cycles.

For example, if User 1 (u1) has 10 pending tasks and User 2 (u2) has 20 pending tasks, then:

$$\alpha_{u1}^{-1}(t) = \frac{10}{30} = \frac{1}{3} \tag{6.3}$$

$$\alpha_{u1}(t) = 3 \tag{6.4}$$

meaning that for a given task, the fairness multiplier is 3 (Eq. 6.4). User 1's tasks move up 3 spots in the queue, but this does not mean that all of u2's tasks fall behind. As a result of the same calculation, user 2's tasks also move up:

$$\alpha_{u2}(t) = \left(\frac{20}{30}\right)^{-1} = \frac{3}{2} \tag{6.5}$$

In Nash's comparison model, the fairness factor is easier to calculate but more iterative:

$$U_u(t) = \frac{1}{\xi(t)} \qquad \#\,\text{Nash} \tag{6.6}$$

For $u = 1..n$, and:

$$a_{u1,u2}(t) = \frac{U_{u1}(t)}{U_{u2}(t)} \tag{6.7}$$

with $\xi(t)$ representing the number of tasks pending for a given user at time t and $U_u(t)$ representing the utility or satisfaction gained by a given user when an additional one of their tasks is executed. The fairness factor, $\alpha_{u1,\ u2}(t)$, relates u1 over u2: the fairness factor is the ratio of the utilities of two users competing for a resource.

In a Nash-based fairness-factor calculation, the utilities of all of the users are calculated. The fairness factor compares the utilities of two users. If user 1 has 10 pending tasks, when a single task is finished for user 1, user 1 gains a percentage change in utility of 1/10, 0.1, or 10%. Similarly, if user 2 has 30 pending tasks, their percentage gain in utility per task completed is 1/30, 0.03, or 3%.

$$\alpha_{u1,u2}(t) = \frac{0.1}{0.03} = 3.33 \tag{6.8}$$

$$\alpha_{u2,u1}(t) = \frac{0.03}{0.1} = 0.3 \tag{6.9}$$

In a direct competition between two users, Nash's model gives more precedence to the user with the lower number of pending tasks. Equation 6.10 shows the matrix for a 3-user system, and 6.11 shows a matrix for the general case:

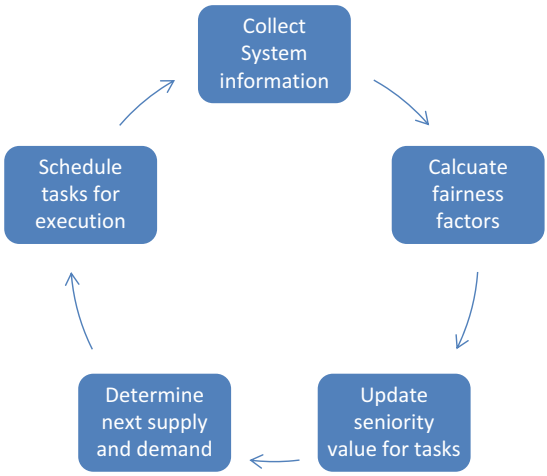

Fig. 6.1 The scheduler's process flow

$$\begin{bmatrix} n/a & \alpha_{u1,u2} & \alpha_{u1,u3} \\ \alpha_{u2,u1} & n/a & \alpha_{u2,u3} \\ \alpha_{u3,u1} & \alpha_{u3,u2} & n/a \end{bmatrix} \tag{6.10}$$

$$\begin{bmatrix} n/a & \cdots & \alpha_{u1,u_n} \\ \vdots & \ddots & \vdots \\ \alpha_{u_n,u1} & \cdots & n/a \end{bmatrix} \tag{6.11}$$

Nash's model is more computationally intensive than other methods of calculating utility, however, as it requires a comparison between all users. The main reason to choose one over the other is computational efficacy.

The scheduler is itself a new operating definition for resource management, as is shown in Fig. 6.1. The scheduler achieves fairness because if it overshoots, incorrectly bumping up a task, the next task to be processed for that user will stay in place or move very slightly towards the front of the queue.

6.1.1 Example of Calculating Seniority

Consider the following scenarios. In the first scenario, two users of a large system each submit one task. Task 1 is submitted by user 1, and task 2 is submitted by user 2 (Table 6.1). While user 1 and user 2 each submit only a single task, the system contains other tasks submitted by other users, which are represented by $\sum_{u} \xi(t)$ (Table 6.2). In the second scenario, user 1` has two pending tasks, while the other conditions are the same as in the first scenario (Tables 6.1 and 6.3).

Table 6.1 Parameters for calculating seniority in a 2-user example

	User 1	User 2	User 1
Number of tasks	1	1	2
Load	{0.5}	{0.5}	{0.5}
Priority	{0.8}	{0.5}	{0.8}
Fairness factor (estimated)	Rawlsian	Rawlsian	Rawlsian

Table 6.2 Calculating seniority in a 2-user example

Params\Time	Definition	t=1	t=2	t=3	t=4	t=5	t=6	t=7	t=8
$\sum_u \xi(t)$	Total pending tasks for all users	100	90	80	70	60	50	40	30
$\xi_{U1}(t)$	Total pending task for user 1	1	1	1	1	1	1	1	1
L_{U1}	Load of task 1	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5
P_{U1}	Priority of task 1	0.8	0.8	0.8	0.8	0.8	0.8	0.8	0.8
α_{U1}	Fairness factor of task 1	100	90	80	70	60	50	40	30
$\Delta S_{U1}(t)$	Change in Seniority for task 1	40	36	32	28	24	20	16	12
$S_{U1\,Total}$	Total Seniority for task 1	**40**	**76**	**108**	**136**	**160**	**180**	**196**	**208**
$\xi_{U2}(t)$	Total pending task for user 2	1	1	1	1	1	1	1	1
L_{U2}	Load of task 2	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5
P_{U2}	Priority of task 2	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5
α_{U2}	Fairness factor of task 2	100	90	80	70	60	50	40	30
$\Delta S_{U2}(t)$	Change in Seniority for task 2	25	23	20	17	15	13	10	7
$S_{U2\,Total}$	Total Seniority for task 2	**25**	**48**	**68**	**85**	**100**	**113**	**123**	**130**

The change in seniority is calculated using Eq. (5.1), and the total accumulated seniority is calculated using Eq. (5.1). In both scenarios, the fairness factor α is designed to implement the Rawlsian definition of fairness, as is suggested by Eq. (5.2).

The total number of tasks pending system wide starts at 100 and decreases every clock cycle. As the time spent in the queue (i.e. the time-in-system) of a task

Table 6.3 Recalculating seniority with an additional task pending for user 1

Params\Time	Definition	t=1	t=2	t=3	t=4	t=5	t=6	t=7	t=8
$\sum_{u} \xi(t)$	Total pending tasks for all users	100	90	80	70	60	50	40	30
$\xi_{U1}'(t)$	Total pending tasks for user 1	2	2	2	2	2	2	2	2
L_{U1}'	Load of *each* task	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5
P_{U1}'	Priority of *each* task	0.8	0.8	0.8	0.8	0.8	0.8	0.8	0.8
α_{U1}'	Fairness factor of *each* task	50	45	40	35	30	25	20	15
$\Delta S_{U1}'(t)$	Change in Seniority for *each* task	20	18	16	14	12	10	8	6
$S_{U1\,Total}'$	Total Seniority for *each* task	**20**	**38**	**54**	**68**	**80**	**90**	**98**	**104**

increases, so does its seniority value. Because it has a higher priority value, user 1's task increases in seniority at a higher rate than does user 2's task.

In scenario 2—which differs from scenario 1 only in that user 1 has 2 pending tasks—the seniority value of user 1 drops by ½ ($S_{U1\ Total} = 104$) to compensate for the increased number of pending tasks, becoming smaller than the seniority value of user 2's task ($S_{U2\ Total} = 130$). This result is shown in Table 6.3.

6.2 Performance Measures

As mentioned in Sect. 1.5, the primary objective of our work was to assess the performance characteristics of a multi-criteria scheduler that utilizes seniority as well as load and priority. Time-in-system was a primary measure in evaluating the scheduler's performance in accommodating various sizes of tasks, classes of workloads, and numbers of users.

Another performance measure, expected utility, which is used to represent user satisfaction, was calculated using the Rawlsian method of fairness. Utility is calculated by comparing the number of completed tasks to the total number of tasks:

$$U_{u_T}(t) = \frac{\xi(t=0) - \xi(t)}{\xi(t=0)} \tag{6.12}$$

where $U_{u_T}(t)$ is the total utility (T) of the user (u) at time t. $\xi(t = 0)$ represents the total number of tasks pending at t = 0—i.e. the total tasks submitted by a given user—and $\xi(t)$ is the current number of pending tasks. A utility plot was used to validate our algorithm.

The algorithm makes scheduling decisions in time intervals, each of which is called a "bucket." Each bucket is filled with incoming tasks. The scheduler makes resource-allocation (i.e. scheduling) decisions for each bucket in the order in which it arrives. Each bucket is then fully processed. For each bucket, the following information is gathered:

- Bucket size
  - The bucket size represents the number of tasks each user submits per scheduling cycle. This value is used to calculate time-in-system for each task and utility of the user.
- Completion time
  - The completion time is used to calculate inter-bucket differences in utility and time-in-system.
- Time-in-System
  - As mentioned previously, a given job can be composed of many tasks that may enter the system at any time. The time-in-system of a given job is a meaningless value because if 2 tasks compose a job, one task may be submitted at the beginning of the day and the other may be submitted at the end of the day. Part of the analysis was devoted to determining the times-in-system of tasks from the various usage profiles we simulated.

In essence, the algorithm time-boxes the utility calculation to ensure that we are within operating parameters.

Bucket duration was also examined. Because each bucket had its own metric, a clear picture of utility emerges. In order to maintain fairness in statistical control, we monitored these values to make sure that the bucket completion time remained in an envelope acceptable to the operators, users, and business. If not, the fairness factor for the user needs to be revisited.

6.3 Experimental Simulation Methodology

The primary goals of the simulation were to validate the new scheduling algorithm and to determine its efficacy in handling various types of workloads. A number of simulators were available, including Optorsim (Bell et al. 2003), the Bricks Grid simulator (Takefusa et al. 2003), GridFlow (Cao et al. 2003), GridSim (Buyya and Murshed 2002), and Alea (Klusáček and Rudová 2010). The majority of these

simulators were designed for job scheduling, however, and thus lacked the characteristics necessary to handle short-running tasks. In addition, we required a simulator that was extendable and able to accommodate dynamic sets of resources, job types, submission profiles, and task durations. We created dSim for this purpose.