Robustness to Estimation Errors for Size-Aware Scheduling

ROBUSTNESS TO ESTIMATION ERRORS FOR SIZE-AWARE SCHEDULING

BY

RACHEL MAILACH, B. Eng

A THESIS

SUBMITED TO THE DEPARTMENT OF COMPUTING AND SOFTWARE

AND THE SCHOOL OF GRADUATE STUDIES

OF MCMASTER UNIVERSITY IN PARTIAL FULFILMENT OF

REQUIREMENTS

FOR THE DEGREE OF

MASTER OF APPLIED SCIENCE

Master of Applied Science (2016) McMaster University

Software Engineering Hamilton, Ontario, Canada

TITLE: Robustness to Estimation Error for Size-Aware Scheduling

AUTHOR: Rachel Mailach
B.Eng., (Software Engineering and Game Design)
McMaster University, Hamilton, Ontario, Canada

SUPERVISOR: Dr. Douglas G. Down

Abstract

When scheduling size-aware single server systems, Shortest Remaining Processing Time (SRPT) has strong optimality properties - it minimizes the number of jobs at the server and as a consequence, the mean response time. A major caveat of SRPT is that it requires job sizes a priori. This thesis examines a scenario that is likely to occur in practice, where only estimates of job sizes are available. A single server model and a multi-server model using SRPT are compared to a Class-Based policy that is designed to increase robustness to estimation errors. In the single server model we observe from simulations that such robustness is crucial to achieve good performance. In contrast, we observe that a multi-server system is inherently more robust than a single server system. Both policies work well with the estimation errors in the multi-server system.

Acknowledgements

I would like to thank Dr. Douglas G. Down for all his help and support and patience throughout my time as his graduate student. Dr. Down was always approachable and available to listen to my tedious questions, and always reminded me of the importance of research, study and, of course, queues. Without him this thesis would not be possible, and I am forever grateful for his help and for my time as his mentee.

Contents

Abstract **i**

Acknowledgements **ii**

Contents **iv**

List of Figures **v**

1 Introduction **1**

1.1 Problem 5

2 Related Work **9**

2.1 SRPT Optimality 9

2.2 Errors in Estimations 10

2.3 Class-Based SRPT 11

2.4 LCFS versus FCFS 12

3 Estimation Errors for a Single Server System **13**

3.1 Models 13

3.2 Mathematical Analysis of the Class-Based Policy 20

3.3 Analysis 26

4 Estimation Errors for a Multi-Server System **31**

4.1 Models 31

4.2 Analysis 35

5 Conclusions **44**
5.1 Future Work 45

A Single Server Results **47**

B Multi-Server Results **51**

Bibliography **51**

List of Figures

1.1 Large job underestimated by 1% 6
3.2 A comparison of the average number of jobs in the system under varying number of classes 16
3.3 A comparison of the average load per class 18
3.4 A comparison of the average number of jobs using the Class-Based policy, where class r is processed according to LCFS and FCFS . . . 19
3.5 Average number of jobs for a single system where policies combined by load 27
3.6 Catastrophic case - large jobs arrive at time 2M units and 2.005M units 29
4.7 SRPT multi-server model 33
4.8 Class-Based multi-server model 34
4.9 Average wait time under various loads on various numbers of servers . 36
4.10 Multi-server average number of jobs results - policies combined by load 39
4.11 Catastrophic case - large jobs arrive at time 2M units and 2.005M units 41
A.1 Single server average number of jobs results 1 48
A.2 Single server average number of jobs results 2 49
A.3 Single server average number of jobs results 3 50
B.4 Multi-server average number of jobs results 1 52
B.5 Multi-server average number of jobs results 2 53
B.6 Multi-server average number of jobs results 3 54
B.7 Multi-server average number of jobs results 4 55
B.8 Multi-server average number of jobs results 5 56
B.9 Multi-server average number of jobs results 6 57
B.10 Multi-server average number of jobs results 7 58

B.11 Multi-server average number of jobs results 8 59
B.12 Multi-server average number of jobs results 9 60

Chapter 1

Introduction

If processing times are known a priori, scheduling using the Shortest Remaining Processing Time (SRPT) is known to minimize the mean response time for a single server system. However, there are some features that make it potentially difficult to implement. These issues will be discussed in detail and we propose as an alternative, a modified policy that can address some of these objections.

When processing jobs in a system, a typical performance goal is to minimize the mean response time. The response time of a job is the time from arrival to the system, to the time the job departs the system. A fundamental result in queuing theory, Little's Law [1], shows that minimizing the mean response time is equivalent to minimizing the mean number of jobs in the system. One technique for minimizing the mean response time of a system is accomplished by using different scheduling policies. Scheduling policies are rules on how the system should react to jobs that arrive and the order in which to process them. Blind scheduling is scheduling tasks or jobs when job sizes are not known. Some common blind scheduling policies are First-Come-First-Serve (FCFS), where jobs are processed in the order of arrival, and Processor-Sharing (PS),

which processes all jobs at the same time, giving each job some fraction of the server's capacity.

When all job sizes are known, size-aware scheduling policies can be developed and amongst these, SRPT is an optimal policy. By optimal we mean that the number of jobs in the system is minimized at every point in time. The optimality of SRPT does not depend on any assumptions about the distribution of either the interarrival times or processing times. The SRPT policy assigns priority to jobs with the shortest remaining processing time. If a job arrives with a shorter processing time than the remaining processing time of the job that is currently being processed, the job in service is preempted [2]. SRPT is an appropriate policy for a system where dynamic scheduling is required, such as web servers, where job sizes are known on arrival. There is no need to have a predetermined list of jobs and their sizes before run-time.

In the field of job scheduling, a current area of study is working with estimated job sizes. In systems such as web servers, exact job sizes may be known. However, the exact time to process these requests may be unknown. This is due to the inherent latency in sending data over the Internet and sending and receiving acknowledgements, thus processing times must be estimated [3]. Web servers often use virtual machines to process jobs, thus another reason for inexact job sizes is the variance of the performance of virtual resources. When job sizes are estimated, errors are introduced. If the scheduling policy is dependent on the job sizes being known, estimation errors may cause performance issues.

If the exact distribution is known, one could create a compelling argument for using class-based approximations of SRPT. Static thresholds are calculated and ensure perfect classification of jobs. In reality, job distributions are rarely known and are difficult to identify or estimate. For example, with web server traffic, job distributions commonly have a high variance and change over time.

If job sizes are known, jobs can be scheduled in real-time. This allows for a changing distribution over time. The problem of real-time job scheduling tends to be studied using an algorithmic approach. These types of algorithms often provide a solution, but not necessarily an optimal one. These solutions are within a fraction of the optimal solution but are generally difficult to implement. The literature in this area is wide and one example of this is Tao et al. [4]. They use an algorithmic approach to a real-time scheduling problem using parallel servers.

Jelenkovic et al. [5] present a policy that groups arrivals into classes and then serves the classes in order of the sizes of jobs that they contain. This is designed to reduce the complexity of SRPT and yield a more practical policy. This is accomplished by comparing the i^{th} arrival to the previous $r-1$ jobs, where r is the desired number of classes. The jobs are sorted by increasing job size from B_1 to B_r, where B is the job size and the subscript represents the placement of a job in the sorted order. If B_i is less than B_1, job i is placed in class 1. To generalize, job i is placed into class k, where $1 \leq k < r-1$ if $B_{k-1} \leq B_i < B_k$. If $B_i \geq B_r$, job i is placed in class r. Class k is only serviced if classes 1 through $k-1$ are empty. Jelenkovic et al. [5] provide both analytic and simulation results that suggest that their grouping of arrivals into

classes provides a satisfactory approximation of SRPT.

Heavy-tailed distributions such as the Pareto distribution have often been chosen to model job sizes in queuing networks as they provide an accurate representation of reality [6, 7, 8, 9, 10]. Heavy-tailed distributions are cause for concern as they create a more difficult scheduling problem due to the large variance in job sizes. They feature a small number of very large jobs that can often occupy a resource for long periods of time. For this reason, FCFS for example, would be disastrous. In this thesis, the issues caused by job size estimation are only a concern under heavy-tailed distributions and not light-tailed distributions. The large jobs represent a large amount of the load in heavy-tailed distributions. If large jobs are underestimated and not dealt with correctly, they may cause issues.

The distribution we chose to model our processing times is Bounded Pareto. The probability density function $B(L, U, \alpha)$ is described in Equation 1.1:

$$f(x) = \frac{\alpha L^{\alpha} x^{-\alpha-1}}{1 - (\frac{L}{U})^{\alpha}}, \quad L \leq x \leq U \tag{1.1}$$

This distribution has three parameters, α, U, and L. U is the upper bound of the possible job sizes, we use a value of 10^6. L is the lower bound of the possible job sizes, we use a value of 1. α determines the shape of the tail of the distribution. The variance increases exponentially as α decreases as defined in [11]:

$$E[X^2] = \frac{\alpha L^{\alpha}(L^{2-\alpha} - U^{2-\alpha})}{(\alpha - 2)(1 - (\frac{L}{U})^{\alpha})}, \quad \alpha \neq 2$$

Thus, we use three different values for α to diversify the variance of the processing time distribution. We chose to use values of 1.1, 1.3, and 1.5 and all simulation data can be found in the appendices.

1.1 Problem

SRPT has often been labelled as impractical because there are a few caveats to using it as a scheduling policy. The first concern is that it is complicated to implement due to the requirement of a large volume of storage for queued jobs. Each job in SRPT must be stored along with its processing time on arrival. When a job enters or departs the system, all jobs currently processing must update their remaining processing time. This run-time complexity increases linearly as the number of jobs grows. This concern may be mitigated in the future as technology becomes less of a limiting factor.

A second reason for avoiding the use of SRPT is that it may force a large amount of preemption. If this preemption has high overhead this can affect the response time. For the remainder of this thesis, it is assumed that preemption has a low overhead cost and is not a critical problem, however, it is still an issue to consider.

Starvation occurs when jobs are delayed or denied processing time due to having a low priority. In SRPT, because jobs with the shortest remaining processing time are given priority, this can lead to starvation of the large jobs for sufficiently high loads [12]. The starvation of larger jobs leads to dissatisfied users, calling fairness into question. This issue is not a concern when the processing time distribution follows a heavy-tailed distribution. This is because the few large jobs carry the majority of the

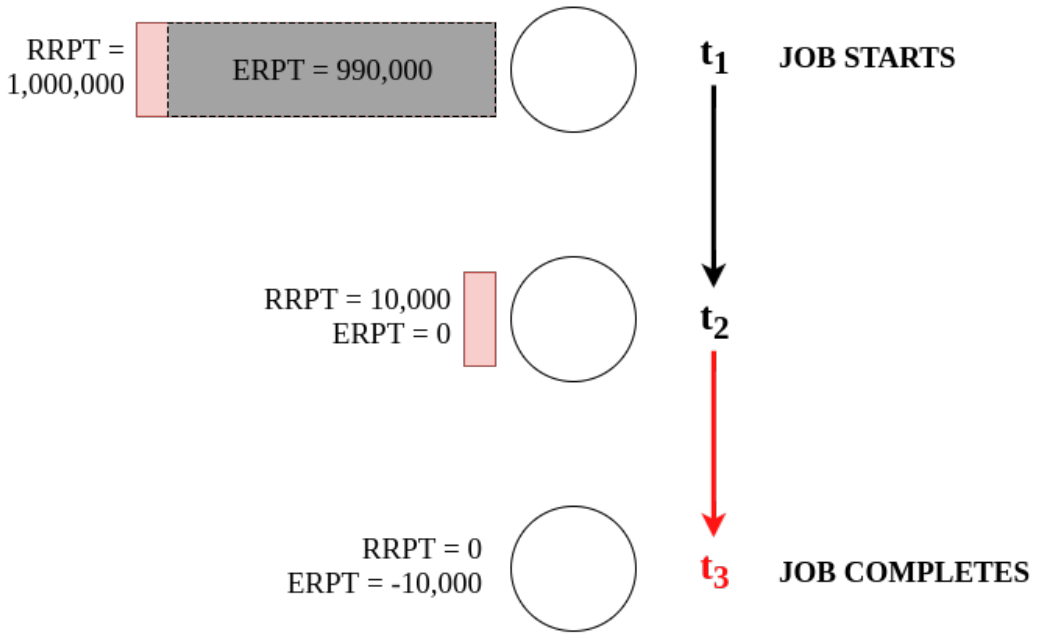


Figure 1.1: Large job underestimated by 1%

system load. The more of the load carried by the large jobs, the less often smaller jobs arrive. This creates idle periods where the large jobs can be processed [13].

The last reason that SRPT is not practical is due to requirements of size-awareness. Job sizes are not always exactly known and may need to be estimated. When estimating job sizes for SRPT, if a job is overestimated, it receives a lower priority and the job may get starved. As mentioned, in this thesis we are concerned with heavy-tailed processing time distributions and as such starvation is not a large concern. The major issues arise when jobs are underestimated.

Figure 1.1 displays an example in which a job enters the system with an actual service time of one million processing time units. Upon arrival, that job is underestimated by 1% and thus has an estimated remaining processing time (ERPT) of 990,000. The

job is processed and may be preempted a number of times during the time interval between t_1 and t_2. At time t_2, the ERPT will equal zero and the job will gain the highest priority within the system until its completion. It will occupy the server and will block any arrivals from processing as no job can arrive with a job size less than zero.

We present one method to counteract this issue, a change in policy. A Class-Based policy based on the work done by [5] is used to contrast SRPT. This new scheduling policy is designed to avoid the issue of underestimated jobs. We then propose two multi-server models that are more robust to estimation errors.

The objective of this thesis is to investigate the practicality of SRPT when there are errors in job size estimation and to determine if the optimality of the SRPT policy is maintained. A practical Class-Based policy is discussed to compare and contrast to SRPT. Simulation tools are constructed to perform the analysis.

We make the following contributions: an augmented policy of [5] is proposed to replace SRPT. An investigation of the impact of estimation errors is done on single server systems and then scaled to multiple server systems.

The organization of the thesis is as follows. Chapter 2 provides a detailed discussion of related work. Chapter 3 presents the models and analysis of the Class-Based and SRPT policies for a single server. Chapter 4 introduces the issues and benefits for a multi-server system under both SRPT and the Class-Based policy. In the final

chapter, Chapter 5, there are conclusions and a discussion of potential future work.

Chapter 2

Related Work

This section explores existing work related to this thesis such as errors in job size estimation, the optimality of SRPT, preemptive Last-Come-First-Serve, and class-based approximations for SRPT.

2.1 SRPT Optimality

In a single server system, the mean response time for SRPT is known to be minimized for general interarrival and processing time distributions [14]. Starvation is one of the main reasons SRPT is not implemented. Work done by Bansal et al. [12] has proven starvation does not typically exist for SRPT under moderate system load. As the load approaches 100%, starvation is more likely to become a problem. They also prove that the mean response time for an M/G/1 system using SRPT is lower than one using Processor Sharing (PS) for all loads less than 90%. SRPT is well studied and well understood for a single server system where job sizes are known exactly, the issues arise during implementation.

SRPT is not an optimal policy for multi-server systems, as shown in [15]. At the time of writing, there is no known optimal policy for scheduling multi-server systems. However, work is being done by those who study scheduling algorithmically. For example, Tao et al. [4] suggest solutions for parallel systems that are within some fraction of the optimal solution.

2.2 Errors in Estimations

Scheduling with errors in job size estimations was first studied by Lue et al. [16]. They examined the performance of SRPT and the Fair Sojourn Protocol (FSP) with jobs of estimated sizes. FSP, described in [17], works by operating the system as if it were running using PS. A list is compiled sorting jobs in order by the time they would complete in PS. The jobs are processed in order following the list. FSP was chosen to be compared with SRPT as it has been proven to outperform PS. In [16] they constructed simulations to show that the performance of SRPT and FSP will exceed PS in terms of mean response time. Their work is limited to M/G/1/m and G/G/n/m queuing models. Job size estimation errors were also discussed in [18]. They analysed the performance of systems with inexact job sizes. The analysis shows that there is a bound on the level of error in job size estimates that the system is able to withstand while maintaining a reasonable mean response time.

There are different ways of dealing with the issues caused by underestimation of job processing times, but only a select few will be discussed in this thesis. The first is

re-estimation of job sizes, periodically or once jobs become late. Late jobs are jobs that have been underestimated and have been processed until their estimated remaining processing time (ERPT) is less than or equal to zero. An alternate approach is to assign late jobs special treatment. Dell'Amico et al. [19] explore policies that are explicitly designed to deal with late jobs. FSP is used as long as jobs are not late. When the jobs become late, they are processed using PS. This policy is then generalized to use Discriminatory Processor Sharing (DPS), which allows increasing priority for the most important jobs.

The approach we opted for, is the use of an entirely different scheduling policy. We chose this approach because we wanted to examine the effect of lateness and how poor the estimates must be before the mean response time becomes too high. We are interested in observing the difference between a policy designed to handle errors and one that is not.

2.3 Class-Based SRPT

An approximate class-based version of SRPT is proposed in [5]. Jobs are classified based on the job sizes of previous arrivals - this will be explained in detail in Section 3.1. The jobs are then routed to m servers where each class has its own queue and its own server. In [5], the simulations use a single server queue with dynamic thresholds. We modelled the single server system the same way. Our work expands on this policy, but modifies it to a multi-server model that has r queues at each server.

2.4 LCFS versus FCFS

Non-preemptive Last-Come-First-Serve (LCFS) is analysed both in [20] and [21], where it is proven that the mean waiting times of FCFS and LCFS are equal for a general distribution. They also prove that the second moment, or variance, of those waiting times differ such that LCFS has a higher variance.

Preemptive LCFS has been studied and analysed in [22] and found to have the same mean response time as PS. It is known that PS has a lower mean response time than First-Come-First-Serve (FCFS) if the variance is higher than for an exponential distribution [22]. In certain situations, where preemption overhead is not a concern, it may be good to consider preemptive LCFS over FCFS.

Chapter 3

Estimation Errors for a Single Server System

This chapter presents the analysis of a single server system. The question of whether the Class-Based policy is more robust than SRPT to processing time estimation error is investigated. Whether or not classes should be served using FCFS or LCFS is addressed.

3.1 Models

Two separate models are discussed in this chapter: an SRPT model and approximate, or Class-Based model. Jobs enter the system following a Poisson Process and are assigned a required processing time that is calculated with respect to a Bounded Pareto distribution. The required processing time is then multiplied by a random percent error value to simulate the estimation errors for approximating job sizes.

SRPT POLICY

When a job enters the system the estimated remaining processing time (ERPT) of the arriving job, Job i, is compared with the current processing job, Job j. If Job i has a smaller ERPT, it preempts Job j and Job j is sent back to the queue. If the arriving Job i has a larger ERPT, it is sent directly to the queue. The queue is sorted in increasing order of ERPT and the job with the current shortest ERPT is always sent to the server. When a job completes and leaves the system, the job next in the queue, with the shortest ERPT, will be sent to the processor until the next event (an arrival or a departure) occurs.

CLASS-BASED POLICY

Algorithm 1 Assign class

```
1: procedure AssignClass(numClasses, job)
2:     while len(prevJobsArray) > (numClasses − 1) do
3:         prevJobsArray.pop(0)
4:     end while
5:     sortedPrevJobsArray = prevJobsArray
6:     sortedPrevJobsArray.append(job)
7:     sortedPrevJobsArray.sort(attribute = ERPT)
8:     i = 1
9:     for j ∈ sortedPrevJobsArray do
10:        if j.name == job.name then
11:            job.class = i
12:            i++
13:        end if
14:    end for
15:    prevJobsArray.append(job)
16: end procedure
```

Assigning Classes

As described in [5], this model is an approximation of the SRPT policy and the class assignment process is called "Comparison Splitting". The class assignment algorithm is given in Algorithm 1. The desired number of classes, r, is chosen to divide the jobs between. When a job enters the system, the assigned processing time is compared with the remaining processing times of the previous $r-1$ jobs. The list of jobs is sorted from 1 through r. The location of the current job in the sorted list will become its assigned class. This is explained further in the example demonstrated in Table 3.1. Job 14 has just arrived to the system, meaning Jobs 1 through 13 have already arrived and been assigned a class before joining the queue. Job 14 is then added to the Previous Jobs table. The table is then sorted by ERPT and Job 14 is 3rd largest in size, as illustrated in Table 3.2. Thus, Job 14 will be assigned to class 3. It is important to note that during the class assignment process, no jobs are re-assigned to classes. It may be more accurate to say that this policy is an approximation to Shortest Processing Time (SPT) than an approximation to SRPT, as jobs do not dynamically change classes even after a reduction in ERPT.

Table 3.1: List of jobs in order of arrival

Previous Jobs	
Job Name	**ERPT**
Job 10	1.4
Job 11	6.0
Job 12	0.3
Job 13	2.0
Job 14	**1.9**

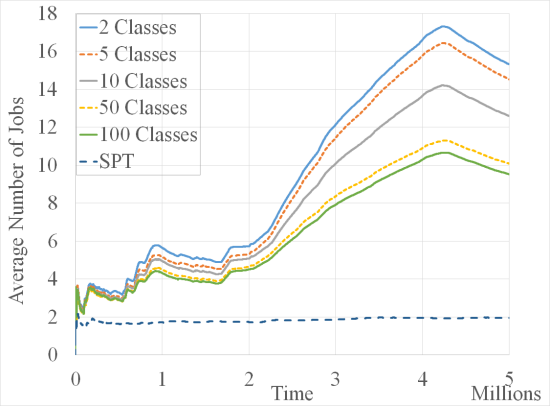
Table 3.2: List of jobs in order of ERPT

Sorted Previous Jobs		
Order (Class)	**Job Name**	**ERPT**
1	Job 12	0.3
2	Job 10	1.4
3	**Job 14**	**1.9**
4	Job 13	2.0
5	Job 11	6.0

Each class at the server has its own queue. After an arriving job is assigned its class, the job is then sent to the corresponding queue. Queues are processed in order of priority, thus all jobs in the class 1 queue must be processed first, then all jobs in the class 2 queue and so on, with class r having the lowest priority. If a job is being processed from one of the lower priority classes and a job arrives that has been assigned to a higher priority class, the job currently being processed is preempted. When a job is preempted, its accumulated processing is not lost. Jobs within each class are processed in order of First-Come First-Serve (FCFS). Processing in order of FCFS allows less data to be stored and updated for each job, thus simplifying the policy.

Parameters

α=1.1, L=1, $U = 10^6$, Simulation Length=5M, Percent Error=0%, Load=95%



(a) Class-Based

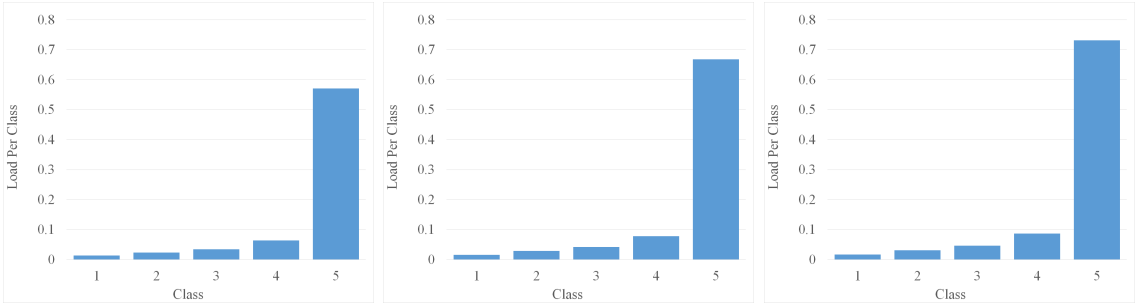
Figure 3.2: A comparison of the average number of jobs in the system under varying number of classes

In a system with exact job sizes, as the number of classes approaches infinity, we approach SPT. Figure 3.2 shows the Class-Based policy with an increasing number of servers and no estimation errors. As the number of classes increase, the relative performance improvement decreases. The downside to having a large number of classes is that it would require more space in order to store the previous $r-1$ jobs necessary for comparison during classification. Recall if the number of classes is infinite, the policy is equivalent to SPT. Another downside to maintaining many classes is that if the processing time distribution changes, it would take longer for the system to adapt. For our system we choose to use 10 classes as we found it to be sufficiently close to SPT while keeping all the benefits mentioned for the Class-Based policy.

After experimentation with this policy, jobs tended to remain queued in class r for long periods of time. This is because it has the lowest priority of the classes, which means queues 1 through $r-1$ would have to be empty in order to begin processing the jobs in class r. This topic is frequently discussed when studying SRPT and is called starvation of large jobs. This occurs when large jobs are frequently preempted or pushed to the back of the queue by smaller jobs who have been assigned higher priority. The class assignment algorithm tends to assign significant load to the last class; this issue is not addressed by [5]. A question arises as to whether increasing r, i.e., adding more classes, would rectify this problem. If there are equal loads in each class, adding more classes would be helpful to reduce the load for each class. This does not occur in simulation, as the last class tends to carry a significant load no matter the size of r. Figure 3.3 illustrates this issue for systems with 5 classes, and 20 classes. This figure shows the percentage of the total load carried by the jobs

Parameters

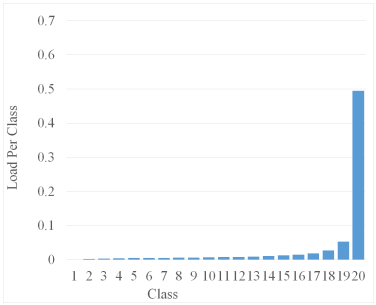
α=1.1, L=1, $U = 10^6$, Simulation Length=5M, Percent Error=0%



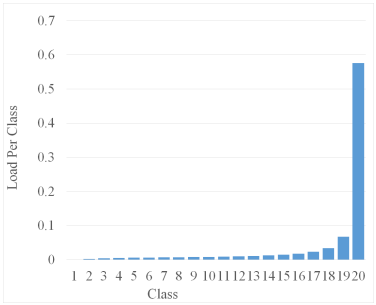
(a) Class-Based, Number of Classes=5, Load=70%

(b) Class-Based, Number of Classes=5, Load=85%

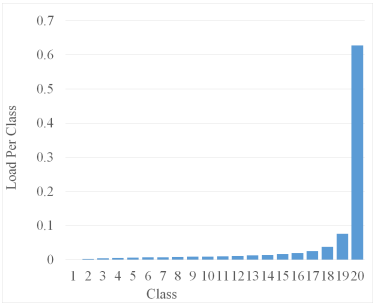
(c) Class-Based, Number of Classes=5, Load=95%



(d) Class-Based, Number of Classes=20, Load=70%



(e) Class-Based, Number of Classes=20, Load=85%



(f) Class-Based, Number of Classes=20, Load=95%

Figure 3.3: A comparison of the average load per class

of each class. It is clear in all cases that the load in the last class remains high no matter how many classes are added.

In order to address the issue of starvation, the policy was adapted so that the lowest priority class is processed using Last-Come First-Serve (LCFS). LCFS will work better in comparison to FCFS for class r because of the distribution of jobs in the last class. Consider a system where a very large job arrives, is assigned to the lowest priority class and is immediately followed by a burst of smaller jobs. The arriving

Parameters

α=1.1, L=1, $U = 10^6$, Simulation Length=5M, Percent Error=[-50%,0%], Load=95%, 10 classes

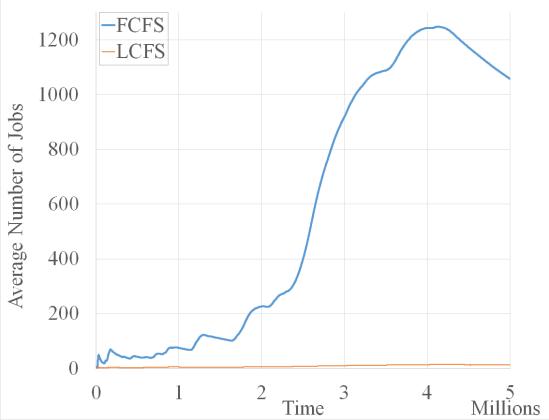


Figure 3.4: A comparison of the average number of jobs using the Class-Based policy, where class r is processed according to LCFS and FCFS

jobs will be spread through the different classes, some of which are assigned to class r. If class r is processed using FCFS, the smaller jobs would be blocked until the completion of the large job. Conversely, if class r is processed using LCFS, the same situation may occur in reverse. A large job might arrive immediately after a burst of smaller jobs where smaller jobs are in the queue. The large job will block these smaller jobs. However, this situation is much less likely to occur because the processing time distribution is heavy-tailed and large jobs are sparse. The likelihood that a large job will arrive before the smaller jobs have completed is low. This reasoning is supported when looking at a side-by-side comparison of LCFS and FCFS for class r in Figure 3.4.

There are two main disadvantages of LCFS compared to FCFS, the first being that LCFS requires much more preemption as it preempts the job currently processing for

every arriving job. Performance may increase with the use of LCFS for every class, or for some sufficient number of classes. There must be a a balance between the preemption overhead and the advantages in performance gained through the use of LCFS. The second disadvantage to LCFS is that it only performs well in a system with high variance for the job size distribution. In our system, only class r has high enough variance to warrant using LCFS over FCFS. Section 3.2 provides more insight into this topic.

3.2 Mathematical Analysis of the Class-Based Policy

Based on the method and notation described in [23], R_i is the remaining processing time and B_i is the service time for jobs of class i. In Equation 3.2, proven by Kleinrock in [24], it is clear that $E[R_i]$ is dependent on $E[B_i]$ and the variance of B_i, ${\sigma_i}^2$. This will be important later as the condition for when LCFS is preferable over FCFS is dependent on the variance of B_i.

$$E[R_i] = \frac{E[B_i]}{2} + \frac{\sigma_i^2}{2E[B_i]} \tag{3.2}$$

If classes 1 through r are processed using FCFS the mean response time, $E[S_i]$ in Equation 3.3, for class i would be calculated by summing the work in the system upon arrival and preemption of the jobs in service as any jobs arrive of higher priority after processing has commenced [23]. ρ_j is the utilization of class j, or the arrival rate to class j, λ_j, multiplied by the mean service time, $E[B_j]$. $E[L_j^q]$ is the expected

number of jobs queued for class j and is equal to the expected wait time for jobs in class j, $E[W_j]$ multiplied by λ_j.

$$E[S_i] = \frac{\sum_{j=1}^{i} \rho_j E[R_j]}{(1 - \sum_{j=1}^{i} \rho_j)(1 - \sum_{j=1}^{i-1} \rho_j)} + \frac{E[B_i]}{1 - \sum_{j=1}^{i-1} \rho_j}, \quad i = 1 \ldots r \tag{3.3}$$

We begin our analysis with the response time of the last class, class r, if it is processed using FCFS.

$$E[S_r^{FCFS}] = \frac{\sum_{j=1}^{r} \rho_j E[R_j]}{(1 - \sum_{j=1}^{r} \rho_j)(1 - \sum_{j=1}^{r-1} \rho_j)} + \frac{E[B_r]}{1 - \sum_{j=1}^{r-1} \rho_j}$$

Next, we consider the case where class r is processed using LCFS. The response time of the last class, using LCFS, is split into two segments. The first segment is the wait time upon arrival to the system, $E[T_1]$. This would include all jobs that are of higher priority that need to be processed before class r and all the jobs of class r that arrive

during the waiting time of processing the higher priority jobs.

$$
\begin{aligned}
E[T_1] &= \sum_{j=1}^{r-1} E[L_j^q]E[B_j] + \sum_{j=1}^{r} \lambda_j E[B_j]E[T_1] + \sum_{j=1}^{r-1} \rho_j E[R_j] \\
&= \frac{\sum_{j=1}^{r-1} E[L_j^q]E[B_j] + \sum_{j=1}^{r-1} \rho_j E[R_j]}{1 - \sum_{j=1}^{r} \lambda_j E[B_j]} \\
&= \frac{\sum_{j=1}^{r-1} \lambda_j E[W_j]E[B_j] + \sum_{j=1}^{r-1} \rho_j E[R_j]}{1 - \sum_{j=1}^{r} \lambda_j E[B_j]} \\
&= \frac{\sum_{j=1}^{r-1} \rho_j E[W_j] + \sum_{j=1}^{r-1} \rho_j E[R_j]}{1 - \sum_{j=1}^{r} \rho_j}
\end{aligned}
$$

The second segment, $E[T_2]$, is the time elapsed since the job began processing, which is comprised of the processing time of the arrival and the time used to process all of the jobs that preempt the arrival.

$$
\begin{aligned}
E[T_2] &= E[T_2] \sum_{j=1}^{r} \lambda_j E[B_j] + E[B_r] \\
&= E[T_2] \sum_{j=1}^{r} \rho_j + E[B_r] \\
&= \frac{E[B_r]}{1 - \sum_{j=1}^{r} \rho_j}
\end{aligned}
$$

The summation of the two segments is as follows.

$$
\begin{aligned}
E[S_r^{LCFS}] &= E[T_1] + E[T_2] \\
&= \frac{E[B_r] + \sum_{j=1}^{r-1} \rho_j E[W_j] + \sum_{j=1}^{r-1} \rho_j E[R_j]}{1 - \sum_{j=1}^{r} \rho_j} \\
&= \frac{E[B_r] + \sum_{j=1}^{r-1} \left(\rho_j \left(\frac{\sum_{k=1}^{j} \rho_k E[R_k]}{(1 - \sum_{k=1}^{j} \rho_k)(1 - \sum_{k=1}^{j-1} \rho_k)} \right) \right) + \sum_{j=1}^{r-1} \rho_j E[R_j]}{1 - \sum_{j=1}^{r} \rho_j} \\
&= \frac{E[B_r] + \sum_{j=1}^{r-1} \left(\rho_j^2 \left(\frac{\sum_{k=1}^{j} E[R_k]}{(1 - \sum_{k=1}^{j} \rho_k)(1 - \sum_{k=1}^{j-1} \rho_k)} \right) \right) + \sum_{j=1}^{r-1} \rho_j E[R_j]}{1 - \sum_{j=1}^{r} \rho_j}
\end{aligned}
$$

The difference between the response times for FCFS and LCFS is described with the

following equation.

$$
\begin{aligned}
&E[S_r^{FCFS}] - E[S_r^{LCFS}] \\
&= \left[\frac{\sum_{j=1}^{r} \rho_j E[R_j]}{(1 - \sum_{j=1}^{r} \rho_j)(1 - \sum_{j=1}^{r-1} \rho_j)} + \frac{E[B_r]}{1 - \sum_{j=1}^{r-1} \rho_j} \right] \\
&\quad - \left[\frac{E[B_r]}{1 - \sum_{j=1}^{r} \rho_j} + \frac{1}{1 - \sum_{j=1}^{r} \rho_j} \sum_{j=1}^{r-1} \left(\rho_j^2 \left(\frac{\sum_{k=1}^{j} E[R_k]}{(1 - \sum_{k=1}^{j} \rho_k)(1 - \sum_{k=1}^{j-1} \rho_k)} \right) \right) + \frac{\sum_{j=1}^{r-1} \rho_j E[R_j]}{1 - \sum_{j=1}^{r} \rho_j} \right] \\
&= \left[\frac{\sum_{j=1}^{r} \rho_j E[R_j] + E[B_r](1 - \sum_{j=1}^{r} \rho_j)}{(1 - \sum_{j=1}^{r} \rho_j)(1 - \sum_{j=1}^{r-1} \rho_j)} \right] \\
&\quad - \left[\frac{E[B_r]}{1 - \sum_{j=1}^{r} \rho_j} + \frac{1}{1 - \sum_{j=1}^{r} \rho_j} \sum_{j=1}^{r-1} \left(\rho_j^2 \left(\frac{\sum_{k=1}^{j} E[R_k]}{(1 - \sum_{k=1}^{j} \rho_k)(1 - \sum_{k=1}^{j-1} \rho_k)} \right) \right) + \frac{\sum_{j=1}^{r-1} \rho_j E[R_j]}{1 - \sum_{j=1}^{r} \rho_j} \right] \\
&= \frac{\sum_{j=1}^{r} \rho_j E[R_j] + E[B_r](1 - \sum_{j=1}^{r} \rho_j) - E[B_r] \sum_{j=1}^{r-1} \rho_j E[R_j](1 - \sum_{j=1}^{r-1} \rho_j)}{(1 - \sum_{j=1}^{r} \rho_j)(1 - \sum_{j=1}^{r-1} \rho_j)} \\
&\quad - \frac{1}{1 - \sum_{j=1}^{r} \rho_j} \sum_{j=1}^{r-1} \left(\rho_j^2 \left(\frac{\sum_{k=1}^{j} E[R_k]}{(1 - \sum_{k=1}^{j} \rho_k)(1 - \sum_{k=1}^{j-1} \rho_k)} \right) \right)
\end{aligned}
$$

LCFS is preferred to FCFS for class r if and only if the following condition holds.

$$
\begin{aligned}
& E[S_r^{LCFS}] \leq E[S_r^{FCFS}] \\
\iff \quad & \frac{E[B_r]}{1-\sum_{j=1}^{r}\rho_j} + \frac{1}{1-\sum_{j=1}^{r}\rho_j}\sum_{j=1}^{r-1}\left(\rho_j^2\left(\frac{\sum_{k=1}^{j}E[R_k]}{(1-\sum_{k=1}^{j}\rho_k)(1-\sum_{k=1}^{j-1}\rho_k)}\right)\right) \leq \frac{\sum_{j=1}^{r}\rho_j E[R_j]}{(1-\sum_{j=1}^{r}\rho_j)(1-\sum_{j=1}^{r-1}\rho_j)} + \frac{E[B_r]}{1-\sum_{j=1}^{r-1}\rho_j} \\
\iff \quad & \frac{1}{1-\sum_{j=1}^{r}\rho_j}\sum_{j=1}^{r-1}\left(\rho_j^2\left(\frac{\sum_{k=1}^{j}E[R_k]}{(1-\sum_{k=1}^{j}\rho_k)(1-\sum_{k=1}^{j-1}\rho_k)}\right)\right) \leq \frac{\sum_{j=1}^{r}\rho_j E[R_j]}{(1-\sum_{j=1}^{r}\rho_j)(1-\sum_{j=1}^{r-1}\rho_j)} + \frac{E[B_r]}{1-\sum_{j=1}^{r-1}\rho_j} - \frac{E[B_r]}{1-\sum_{j=1}^{r}\rho_j} \\
\iff \quad & \frac{1}{1-\sum_{j=1}^{r}\rho_j}\sum_{j=1}^{r-1}\left(\rho_j^2\left(\frac{\sum_{k=1}^{j}E[R_k]}{(1-\sum_{k=1}^{j}\rho_k)(1-\sum_{k=1}^{j-1}\rho_k)}\right)\right) \leq \frac{\sum_{j=1}^{r}\rho_j E[R_j]}{(1-\sum_{j=1}^{r}\rho_j)(1-\sum_{j=1}^{r-1}\rho_j)} - \frac{E[B_r]\rho_r}{(1-\sum_{j=1}^{r}\rho_j)(1-\sum_{j=1}^{r-1}\rho_j)} \\
\iff \quad & \cancel{\frac{1}{1-\sum_{j=1}^{r}\rho_j}}\sum_{j=1}^{r-1}\left(\rho_j^2\left(\frac{\sum_{k=1}^{j}E[R_k]}{(1-\sum_{k=1}^{j}\rho_k)(1-\sum_{k=1}^{j-1}\rho_k)}\right)\right) \leq \frac{\sum_{j=1}^{r}\rho_j E[R_j]-E[B_r]\rho_r}{\cancel{(1-\sum_{j=1}^{r}\rho_j)}(1-\sum_{j=1}^{r-1}\rho_j)} \\
\iff \quad & \sum_{j=1}^{r-1}\left(\rho_j^2\left(\frac{\sum_{k=1}^{j}E[R_k]}{(1-\sum_{k=1}^{j}\rho_k)(1-\sum_{k=1}^{j-1}\rho_k)}\right)\right) \leq \frac{\sum_{j=1}^{r}\rho_j E[R_j]-E[B_r]\rho_r}{(1-\sum_{j=1}^{r-1}\rho_j)} \\
\iff \quad & \left(1-\sum_{j=1}^{r-1}\rho_j\right)\sum_{j=1}^{r-1}\left(\rho_j^2\left(\frac{\sum_{k=1}^{j}E[R_k]}{(1-\sum_{k=1}^{j}\rho_k)(1-\sum_{k=1}^{j-1}\rho_k)}\right)\right) \leq \sum_{j=1}^{r}\rho_j E[R_j] - E[B_r]\rho_r \\
\iff \quad & \rho_r(E[R_r]-E[B_r]) \geq \left(1-\sum_{j=1}^{r-1}\rho_j\right)\sum_{j=1}^{r-1}\left(\rho_j^2\left(\frac{\sum_{k=1}^{j}E[R_k]}{(1-\sum_{k=1}^{j}\rho_k)(1-\sum_{k=1}^{j-1}\rho_k)}\right)\right) - \sum_{j=1}^{r-1}\rho_j E[R_j]
\end{aligned}
\tag{3.4}
$$

Inequality 3.4 describes when LCFS is preferred over FCFS for class r. It is a complicated expression that we have not been able to simplify further. If the left hand side, which is dependent on the processing time variance of class r, is greater than the right hand side, then LCFS is preferred. As the variance increases, the more desirable LCFS becomes for that class. This expression can be extended to determine exactly how many classes one would want to use LCFS for. To determine if class $r-1$ should use LCFS we are able to use the same formula where $r-1$ is substituted for r. In

the situations we have explored in this thesis the r^{th} class is the only class that has a high enough variance to warrant using LCFS.

3.3 Analysis

For this and subsequent simulations, simulation code was written using the Python 3.x language and is available for download at [25] and [26]. The simulations were each run with the SRPT policy and the Class-Based SRPT policy. Policies were run with the same seed value and other parameters such as processing distribution so the arriving jobs would be identical. All graphs in this paper use a sample rate of 1 per every 100 events.

In this section the two policies, SRPT and Class-Based, are compared to see which policy reacts better to estimation errors. All single server simulation results are available in Appendix A.

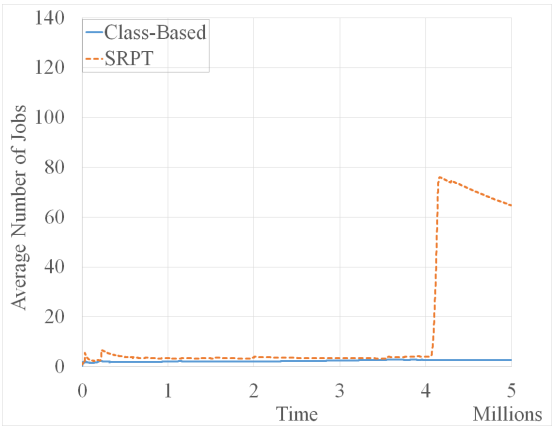
Figure 3.5 shows the comparison between the SRPT policy and the Class-Based policy for three different loads. In all three graphs there are large jumps in the average number of jobs when using the SRPT policy, but the jumps are almost non-existent when using the Class-Based policy.

The large vertical gains correspond to extremely large jobs that are underestimated. When these jobs reach zero ERPT, they are considered to be the smallest in the system, and because of this, are given the highest priority. They block jobs that have true sizes that are much smaller than the true remaining processing time of the large

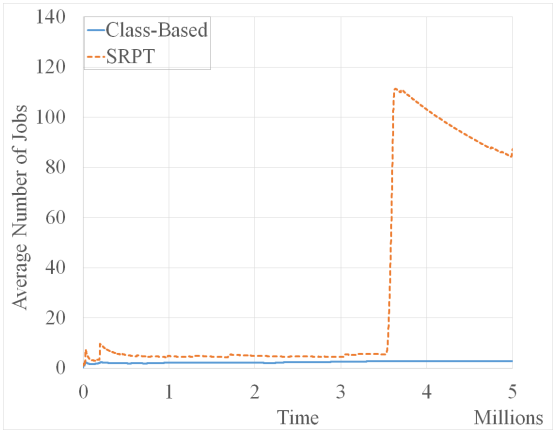
jobs. As these jobs are being processed, a large number of jobs enter the system and are forced to wait.

Parameters

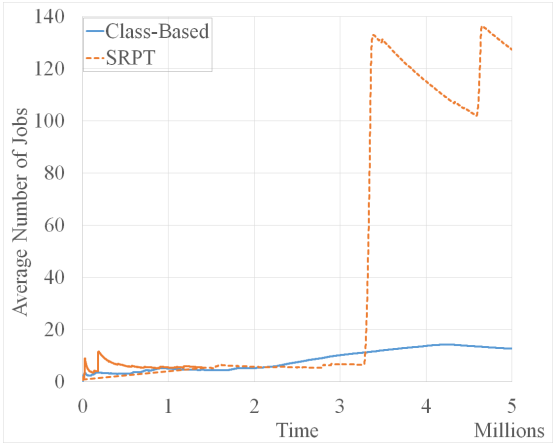
α=1.1, L=1, $U = 10^6$, Simulation Length=5M, Percent Error=[-50%,0%]



(a) SRPT + Class-Based
Load = 70%



(b) SRPT + Class-Based
Load = 85%



(c) SRPT + Class-Based
Load = 95%

Figure 3.5: Average number of jobs for a single system where policies combined by load

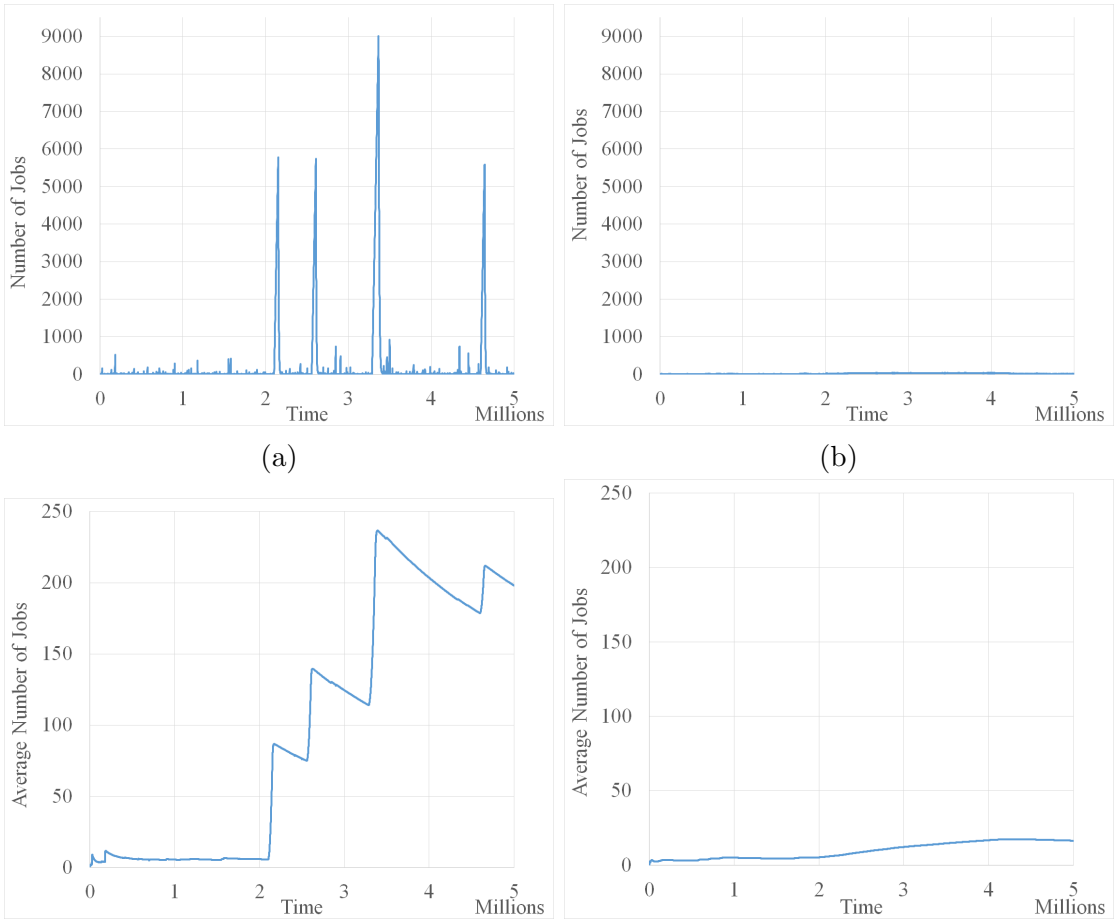
The Class-Based policy does much better than SRPT because the large jobs that are underestimated are still sent to class r. Refer back to the example illustrated in Figure 1.1, where a large job of size 1,000,000 units is underestimated by 1%. If the average job size in the system is on the order of 10, it is reasonable to assume that a job of size 990,000 would be the largest job in the system. Since the Class-Based policy processes in order of priority by class, a job in class r will not block jobs in queues 1 through $r-1$. Recall, jobs are never re-classified. This effect is why the Class-Based policy is much more robust to estimation errors than SRPT.

In order to test the robustness of the systems, a more extreme case is considered. This situation, as seen in Figure 3.6, illustrates how both systems deal with large jobs, an order smaller than the upper limit of processing times, when they are underestimated by 50%. With processing times of 10^5 units, they are estimated to be of size 5×10^4. One of the large jobs is injected into the system at 2M units, and the second at 2.005M units. This would describe a situation where two large jobs arrive to the system closely together. Though it is an unlikely scenario, it could be problematic. However, as described below, it is not as problematic as one might think.

The system reacts to this case the same way it does to the typical cases in Figure 3.5. This is because a catastrophic situation in the single server SRPT is the basic case where a single large job arrives and is underestimated and it occupies the server until it completes. The second large job does not have its ERPT decreased until the first job completes, so the close arrival of two large jobs does not compound the issue of underestimating the size of large jobs. The Class-Based policy will classify large jobs

Parameters

α=1.1, L=1, U=10^6, Simulation Length=5M, Percent Error=[-50%,0%], Load = 95%



(a) (b)

(c) SRPT (d) Class-Based 10 classes

Figure 3.6: Catastrophic case - large jobs arrive at time 2M units and 2.005M units

into the lowest priority class and they will not block any jobs in higher priority classes.

In general, the Class-Based policy is more robust to estimation error than the SRPT policy in a single server system. These results are promising if there are concerns about the caveats mentioned above that arise when implementing SRPT.

Chapter 4

Estimation Errors for a Multi-Server System

This chapter presents the analysis of SRPT and the Class-Based policy in multi-server systems. The primary motivation for looking into multi-server systems is to explore the effect of estimation error as the system scales up. We question the benefits of using the Class-Based policy in the place of SRPT in a multi-server system.

4.1 Models

There are two models described in this chapter, both of which include a set of parallel servers. The first model applies a multi-server version of the SRPT policy, and the second applies the multi-server version of the Class-Based policy that was discussed in Section 3.1.

SRPT POLICY

In the single-server system, the SRPT policy was used to compare the Class-Based policy to the optimal policy. As discussed in the Introduction, SRPT is not optimal in a multi-server system. While not optimal, SRPT does appear to perform reasonably well for a multi-server system. At the time of writing, there are no known optimal policies and if one does exist it is most probably very complex. Our suggested policy, the Class-Based policy, is compared to SRPT, as no optimal benchmark exists to compare it to.

As depicted in Figure 4.7, jobs arrive to the system following an arbitrary arrival process. There is a global queue sorted by increasing order of ERPT. Arriving jobs are immediately sent to the Router which finds the job with the largest ERPT currently processing in one of the m servers, call this job Job j. The arriving job's ERPT is compared to Job j's ERPT. If the arriving job has a smaller ERPT, it preempts Job j and Job j is sent back to the queue. If the arriving job has a larger ERPT, it is sent directly to the queue.

CLASS-BASED POLICY

There are a few key differences between the two multi-server models, the SRPT model in Figure 4.7, and the Class-Based model in Figure 4.8. The first difference between the two models is that the Class-Based model has a component called Assign Class. On arrival, the jobs are assigned a class, which acts as a priority level. Jobs are classified using the same technique as the single server model. Jobs are then routed to the servers based on their assigned class using the Round Robin (RR) method described

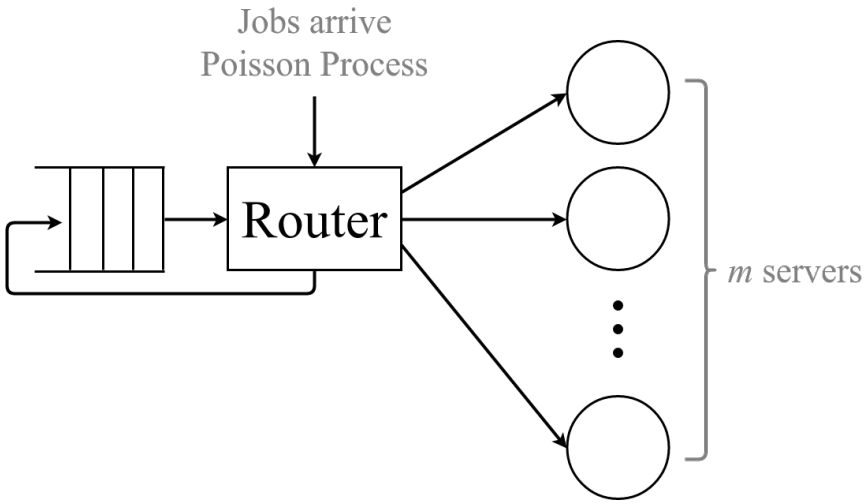


Figure 4.7: SRPT multi-server model

in [27] to balance the loads between the servers. RR was chosen because it distributes the jobs of each class so that each server has an approximately equal number of jobs from each class. It also minimizes the variance between interarrival times at each server from each class. RR does not require any state information from the servers. This keeps the routing operation fairly simple to implement.

The second key difference between the two models is that in the SRPT model, a global queue is used for all waiting jobs. Conversely, in the Class-Based model, there are r queues at each server; where r is the number of classes. Each server in the Class-Based model implements the preemptive Class-Based policy independently.

Routing Policy

Jobs are routed by class to each server queue using a RR policy. After Job i has been assigned to class k, there is a look up table which is used to determine the server that the previous job of class k was routed to. If the last job of class k was sent to Server

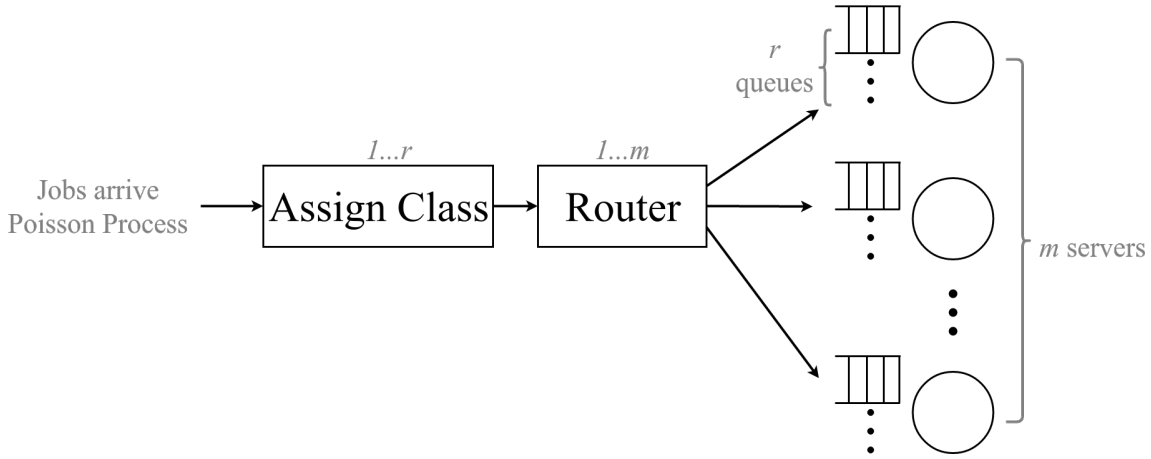


Figure 4.8: Class-Based multi-server model

n, Job i is then sent to Server $1 + n \mod numServers$. The table is then updated with the server that Job i is routed to. An example is illustrated in Figure 4.3 where Job 14 has arrived and been assigned to class 3. The next step is to look up on the Previously Routed table where class 3 had been last routed to. In this example, class 3 was last routed to Server 1, so the job is routed to the next server, Server 2, and the table is updated. This way each server should be sent an approximately equal number of jobs from each class. Other routing options are discussed in Section 5.1.

Table 4.3: Class-Based multi-server routing example Using Round Robin

Previously Routed	
Class	**Server Routed To**
1	2
2	2
3	**1**
4	3
5	2

Table 4.4: Update server for future use

Previously Routed	
Class	**Server Routed To**
1	2
2	2
3	**2**
4	3
5	2

Number of Classes

The choice of number of classes for this policy has been discussed in Section 3.3. The same principle applies in the multi-server policy as the single server policy is implemented independently m times.

4.2 Analysis

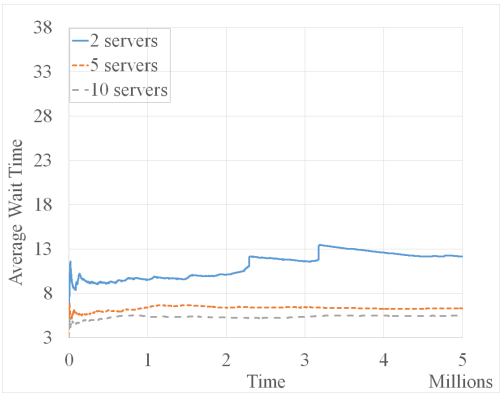
In this section, the two policies are compared to determine if the Class-Based policy may be an intelligent choice for scheduling a multi-server size-aware system. All multi-server simulation results are available in Appendix B.

Figure 4.9 depicts how each policy behaves under different loads with increasing numbers of servers, using $\alpha = 1.1$. Recall the closer α is to 1, the higher the variance in the job sizes. With the three different loads (4.9a, 4.9c, and 4.9e), the SRPT policy experiences some issues with underestimated jobs in the two server systems. The Class-Based policy (4.9b, 4.9d, and 4.9f) has a similar average wait time under all three loads.

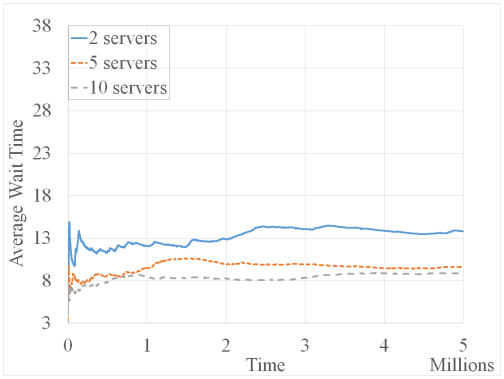
In all cases except the two class SRPT system under a load of 85%, the Class-Based policy seems to have a slightly higher average wait time than the SRPT policy. One reason this may occur has been discussed in 3.1. To reiterate, because of the way the classes are assigned to arriving jobs, it is possible that some jobs are poorly classified. If a large burst of similar sized jobs arrives, some of those jobs may be placed in lower priority classes as a result of being slightly larger than the other arrivals. Another reason may be because the utilization of the servers is not as efficient in the

Parameters

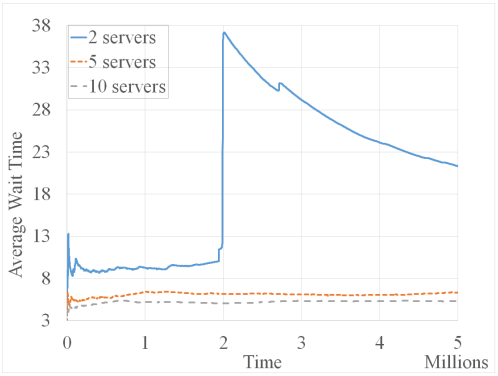
α=1.1, L=1, $U = 10^6$, Simulation Length=5M, Percent Error=[-50%,0%]



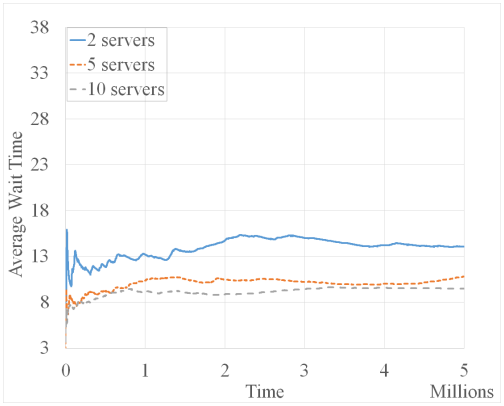
(a) SRPT
Load = 70%



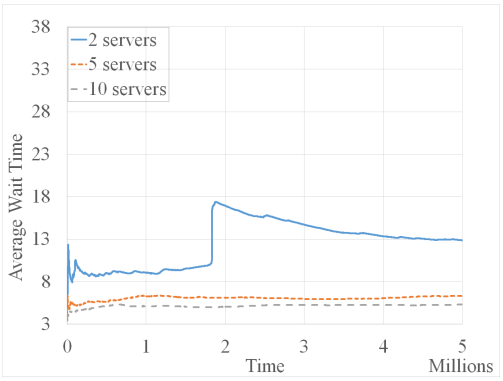
(b) Class-Based
Load = 70%, 10 classes



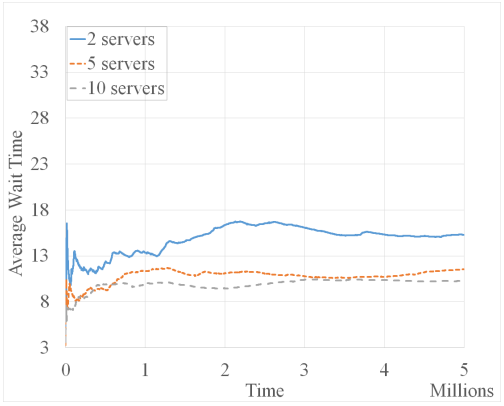
(c) SRPT
Load = 85%



(d) Class-Based
Load = 85%, 10 classes



(e) SRPT
Load = 95%



(f) Class-Based
Load = 95%, 10 classes

Figure 4.9: Average wait time under various loads on various numbers of servers

Class-Based policy. In the SRPT model, if there are any jobs in the queue, it means all servers are busy. Recall, in the Class-Based policy jobs are never re-classified or re-routed. A situation could arise where one server has a large queue and one or more servers are idle because they have completed all the jobs routed to them.

The two server system shown in Figure 4.9c is a special case in this set of graphs. The arrival streams to each of these graphs varies depending on load and number of servers, so no two arrival streams are identical. What is visible here is a situation where a large job that has been underestimated is occupying a server. In this case there are only two servers and there is a significant increase in the average wait time. In the two server case, it is fairly likely that this situation could arise. We investigate this topic in a little more detail below.

When inspecting the SRPT policy, there is very little difference to observe from the varying number of servers once there are five or more servers under all loads. The average wait time is decreasing with the increasing number of servers. As more servers are added to either model, there is a decrease in the effect that the number of servers has on the average wait time in the system.

When scaling the system from one server to multiple servers, there are two main aspects that must be addressed: robustness to estimation errors and maintaining busy servers. The Class-Based policy is inherently more robust to estimation errors than SRPT as any large jobs that are underestimated are still classified into class r, as previously discussed.

When comparing the two policies in Figure 4.9, as the number of servers goes up, the average wait time for SRPT is lower than Class-Based. A reason for this is the inefficient usage of servers in the Class-Based policy. SRPT uses a global queue so there are no idle servers when the queue contains jobs. In the Class-Based policy, there are some servers that have queues that are non-empty while there are idle servers in the system.

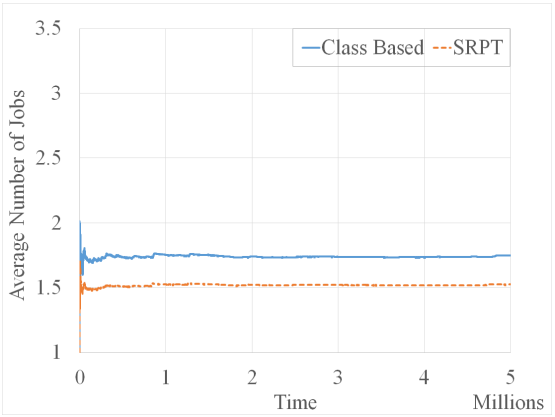
As the number of servers in the system grows, the effect of the idle servers from the Class-Based policy become more of a concern than the benefits we gain for robustness to estimation errors. If the system is has many servers, SRPT is preferable.

Figure 4.10 contrasts the two policies against one another using an α of 1.5 with two servers. In this case, the SRPT policy does slightly better than the Class-Based policy. This is because when α is 1.5 there are less very large jobs, resulting in less potential problems for the system. The main advantage of using the Class-Based policy is its ability to deal well with underestimated large jobs. In this situation, the Class-Based policy is at a disadvantage and the system still maintains a low average number of jobs.

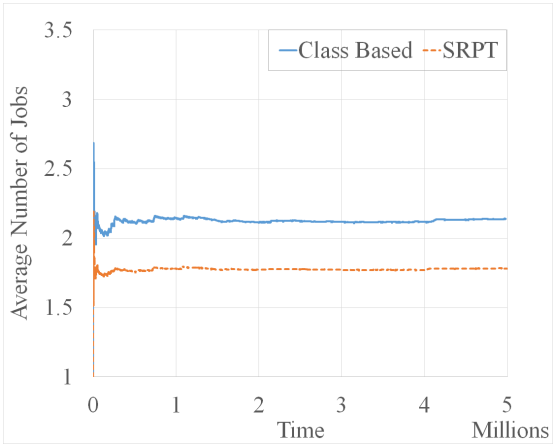
In order to test the robustness of the systems, a more extreme case is considered. This artificial situation, as seen in Figure 4.11, demonstrates how both systems deal with large jobs. The large jobs have processing times of 10^5 and are underestimated by 50% so they are estimated to be of size 5×10^4. One of the large jobs is injected into the system at 2M units, and the second at 2.005M units. This describes a situation where two large jobs arrive to the system closely together. This may result in a catastrophe

Parameters

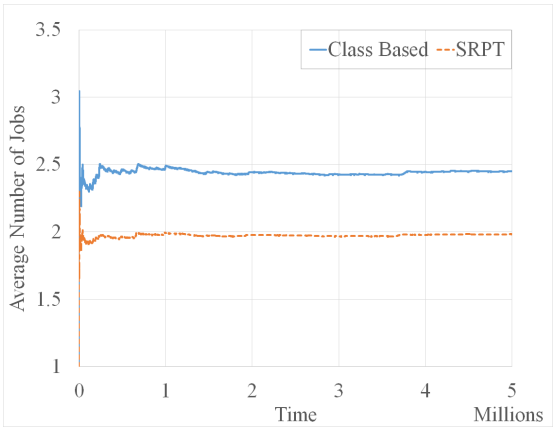
α=1.5, L=1, $U = 10^6$, Simulation Length=5M, Percent Error=[-50%,0%], 2 servers



(a) SRPT + Class-Based load = 70%



(b) SRPT + Class-Based load = 85%



(c) SRPT + Class-Based load = 95%

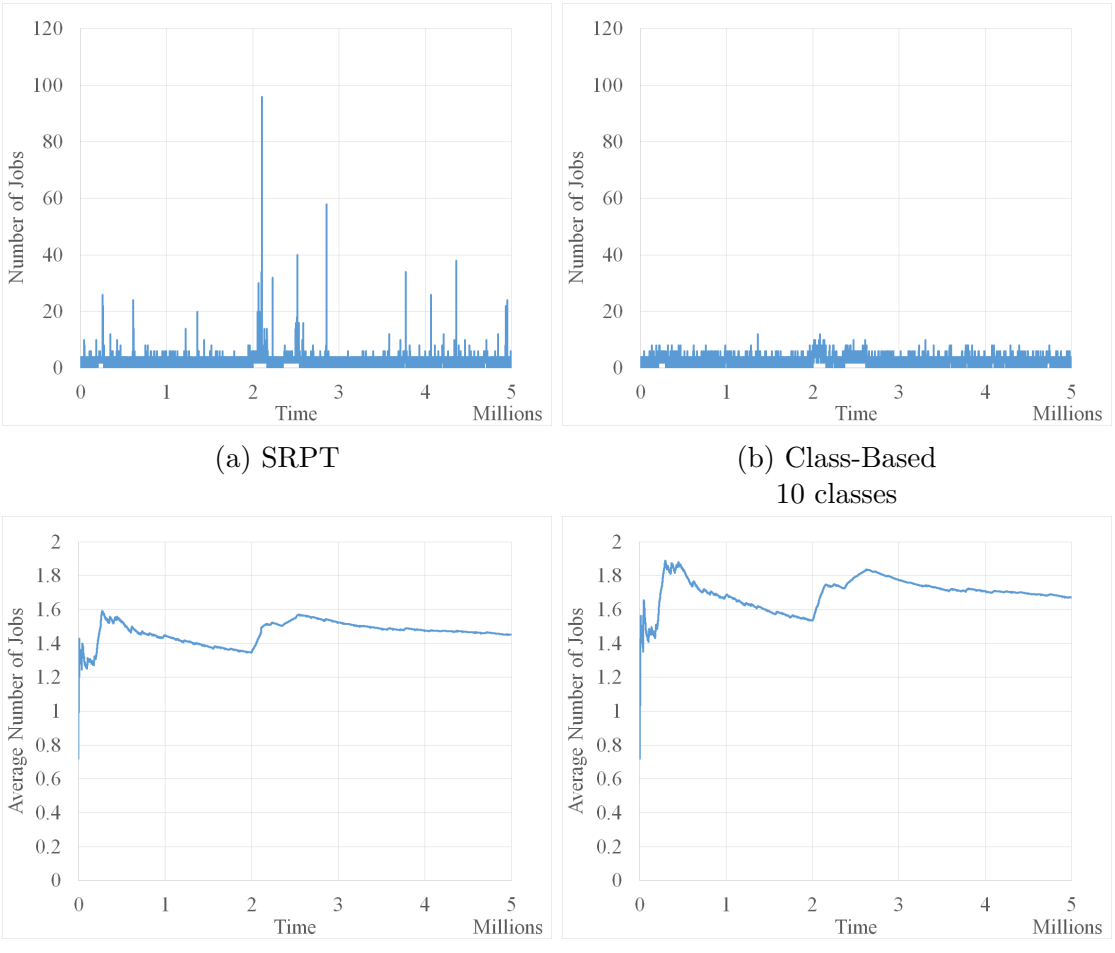
Figure 4.10: Multi-server average number of jobs results - policies combined by load

as when both jobs are still in the system and have an estimated remaining processing time less than zero, they would occupy both servers, and hence block many small jobs.

Intuition would dictate that the Class-Based policy should do a better job at handling the underestimated jobs as was seen in the single server case. A large underestimated job is still considered a large job and classified as such. Thus the large jobs are only served when all higher priority classes have been served and are now empty. Because the classes of the jobs do not get upgraded, the underestimated jobs can only block jobs of their own class. Conversely, the SRPT policy ensures that once the job has an estimated remaining processing time less than or equal to zero, that job has the highest priority in the system and will keep a server occupied until the job completes. Thus, Figure 4.11d is counter-intuitive as it does not have a lower average number of jobs than SRPT. This is because of the high load in the system. A high load means that there are more arrivals and because of the heavy-tailed distribution there will be a large number of small arrivals. With a load of 95%, there are many small arrivals in between the first large job and the second large job. This means that the second job may not actually occupy a server before the first job completes. So this situation is not as problematic as might have initially been expected. Thus, it is hard to envision a catastrophic case with a high load. Interestingly, this effect does become problematic when there is a low load. With a low load, few enough small jobs arrive that the second job occupies a server at the same time that the first job is occupying a server, forcing all small jobs that arrive afterwards to wait. On the positive side, this case is even less likely to occur because in order to cause an issue, it would require enough small jobs to arrive and wait for the large jobs, which is doubtful to occur naturally

Parameters

α=1.1, L=1, U=10^6, Simulation Length=5M, Percent Error=[-50%,0%], Load=95%, 2 servers



(a) SRPT

(b) Class-Based 10 classes

(c) SRPT

(d) Class-Based 10 classes

Figure 4.11: Catastrophic case - large jobs arrive at time 2M units and 2.005M units

due to the low load.

It was mentioned that this catastrophic case is artificially designed and is extremely unlikely to occur naturally. One might question the value of looking into this rare case. As this case is so unlikely to happen, it is hard to see in a normal situation. We manufactured the situation because if it did occur naturally, there could have been issues that we would have been unaware of. We can now be convinced that even in for this scenario, the system performs reasonably well. As the number of servers in the system grows, this becomes less and less of an issue as it is harder and harder to construct a catastrophic case. In order to recreate this catastrophic case with a large number of servers, the number of arriving underestimated large jobs must also be scaled up.

Consider a system with m servers, when a large job is underestimated it occupies a server for the duration of its service time. The system then uses the remaining $m-1$ servers to deal with the rest of the load. If there is a system with a high load, and there are x underestimated large jobs that are occupying x servers, then $m-x$ servers are free to handle the remaining jobs. The system only becomes unbalanced if the $m-x$ servers can not handle the remaining small, and medium sized jobs. In this scenario, the large jobs carry the load, not the average sized jobs. There may be large time gaps between arrivals. For example, if there are four servers and 50% load and two of the servers are occupied, there is a very low probability that another large job arrives before the two jobs occupying the servers complete. The system would appear to have a lower load as the remaining load would not consist of extremely large jobs.

As the load grows and the number of servers grows, longer simulations are required to observe a catastrophic case. Thus as the number of servers grow, they are better able to compensate for the estimation errors.

When designing a system with estimated job processing times, it is advisable to use two or more servers as there is a large increase in robustness when moving from a single server model to a multi-server model. When using SRPT, the performance concerns of estimation errors are mitigated for multi-server systems. It may be an intelligent choice to implement the Class-Based policy instead of the SRPT policy for the same reasons mentioned in Section 1.1. The reasons being ease of implementation, reduced preemption, smaller storage requirement, and lesser sensitivity to estimation errors. The Class-Based policy is only advisable for a small multi-server systems as the effects of estimation errors are mitigated by the number of servers in the system in which case, SRPT would be a better choice. The Class-Based policy introduces server idleness, whereas SRPT does not.

Chapter 5

Conclusions

When considering the implementation of SRPT, which is the optimal policy for scheduling on single server systems, there are a few well known issues. The first is that it requires a large volume of storage to maintain the data of the queued jobs. It also tends to have a large amount of preemption which may severely affect performance. When looking at real-life scenarios such as web servers, processing time estimations may not be exact, thus, there will be some errors. These errors may cause response times to grow at an alarming rate and must be considered when attempting to use the single-server optimal policy. When moving to a multi-server model, we have demonstrated the increase in robustness to estimation error.

We simulated both single and multi-server systems with SRPT and a Class-Based policy. We then analysed those results and discussed reasons for the results. From this research, we can draw certain conclusions. The first is that a multi-server system will always be more robust to estimation errors than a single server system. This is because if there are issues, there are always additional servers for the remaining load.

The second conclusion that we can draw is that the Class-Based policy proposed is a good option for making the SRPT policy more practical. In the single-server case we see a drastic increase in robustness when using the Class-Based policy over SRPT. In the multi-server case, both policies preform relatively well. The Class-Based policy does fall short of SRPT in performance as the number of servers increase.

Using SRPT with only one server could be disastrous with estimation errors and this thesis provides an alternative policy when using inexact job sizes. Our work provides a practical method to implement a policy that has near-optimal behaviour without some of the caveats of SRPT. It also provides an examination of single and multiple server systems and how they each react to job-size estimation errors.

5.1 Future Work

In the future, we would like to work on a better classification scheme of jobs. As mentioned above, our assignment algorithm can classify small jobs to the lowest class and those jobs can be blocked by a large arrival. To rectify this issue, one method that could be investigated in future work is designing a policy that can learn the distribution over time. This would allow for setting better thresholds as long as the distribution is relatively static. If the distribution is frequently changing, the thresholds might have issues adapting to these changes. Another addition to the class assignment algorithm could be to set a lower bound to the threshold of class r. This would mean that if Job j was given a low priority, it would not end up in the last class and be blocked by any extremely large jobs. This addition does create a question on

where to send Job j.

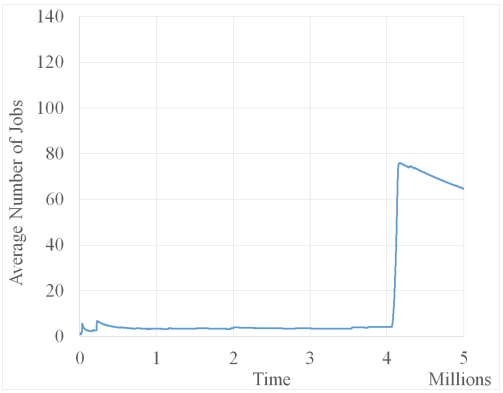
A second topic to investigate is the routing of jobs to the servers. Round Robin (RR) could be used with the condition that no servers are idle. This creates a question on how to update the Previously Routed table in order to maintain balanced classes. In this scenario, even if a server is idle it may be desirable to avoid sending large jobs to it.

Appendix A

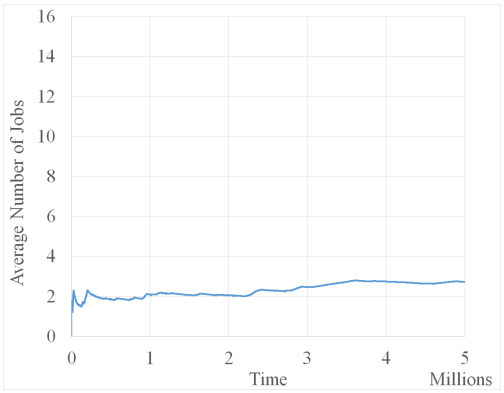
Single Server Results

Parameters

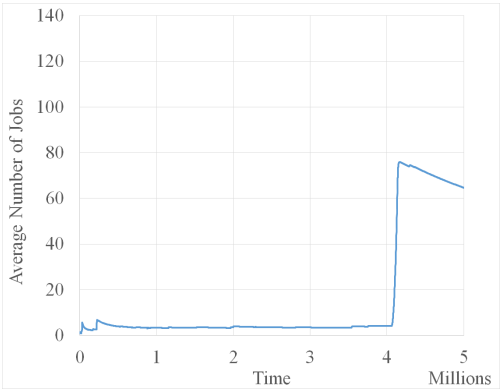
α=1.1, L=1, $U = 10^6$, Simulation Length=5M, Percent Error=[-50%,0%]



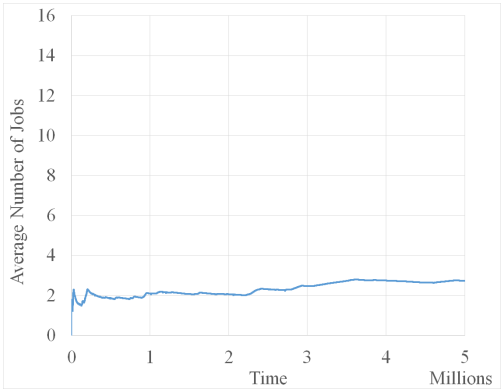
(a) SRPT
Load = 70%



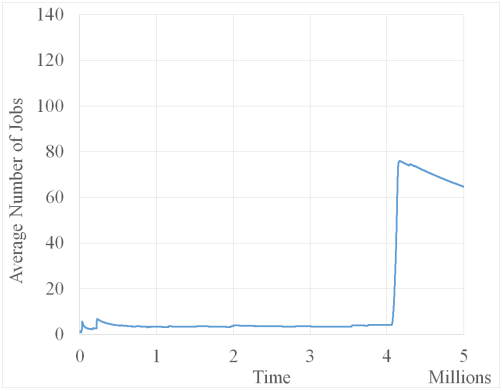
(b) Class-Based
Load = 70%, 10 classes



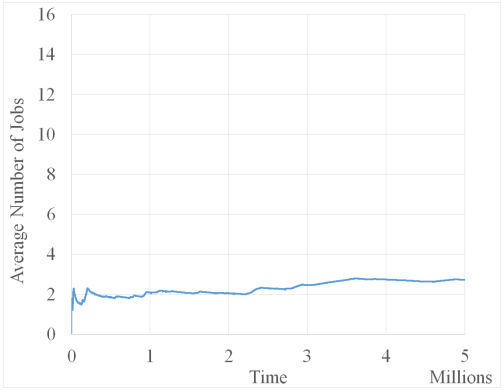
(c) SRPT
Load = 85%



(d) Class-Based
Load = 85%, 10 classes



(e) SRPT
Load = 95%

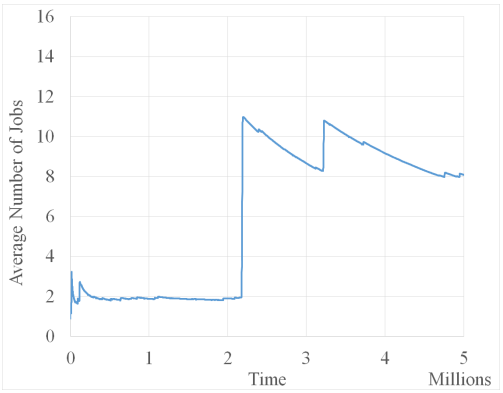


(f) Class-Based
Load = 95%, 10 classes

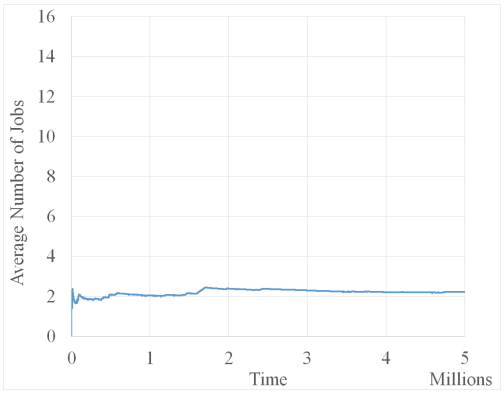
Figure A.1: Single server average number of jobs results 1

Parameters

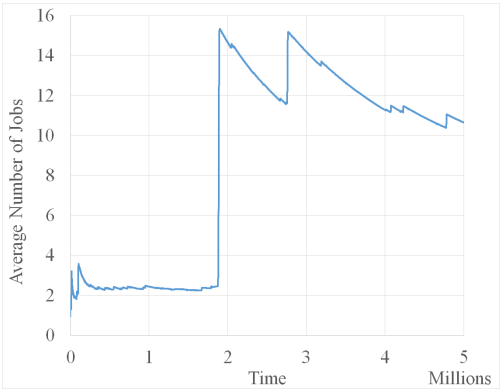
α=1.3, L=1, $U = 10^6$, Simulation Length=5M, Percent Error=[-50%,0%]



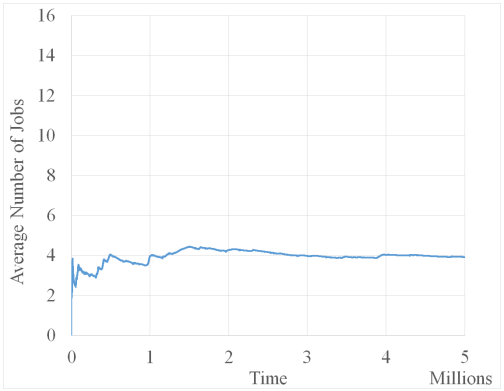
(a) SRPT
Load = 70%



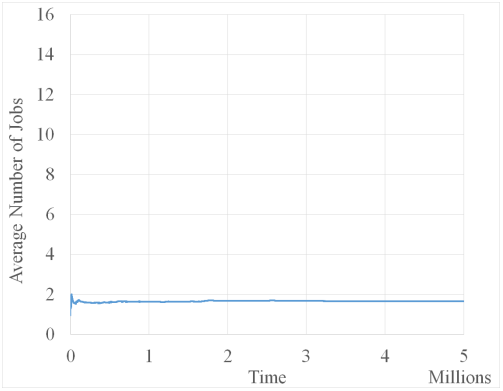
(b) Class-Based
Load = 70%, 10 classes



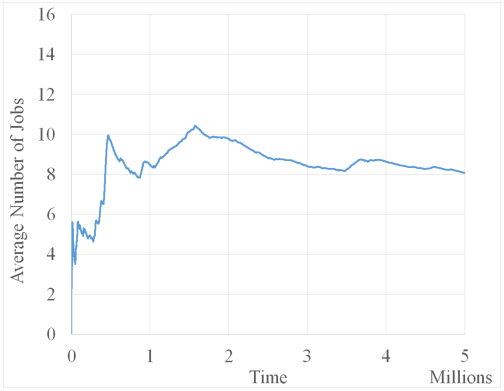
(c) SRPT
load = 85%



(d) Class-Based
Load = 85%, 10 classes



(e) SRPT
Load = 95%

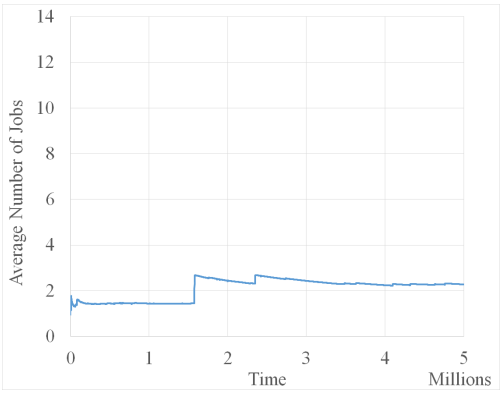


(f) Class-Based
Load = 95%,10 classes

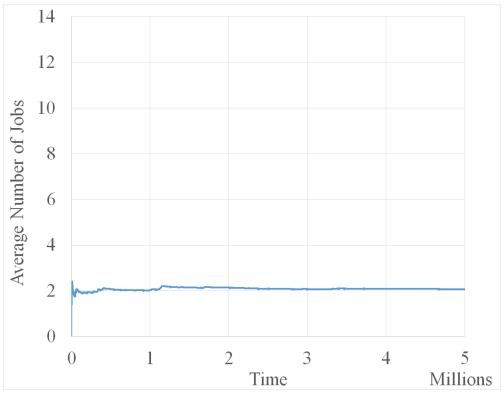
Figure A.2: Single server average number of jobs results 2

Parameters

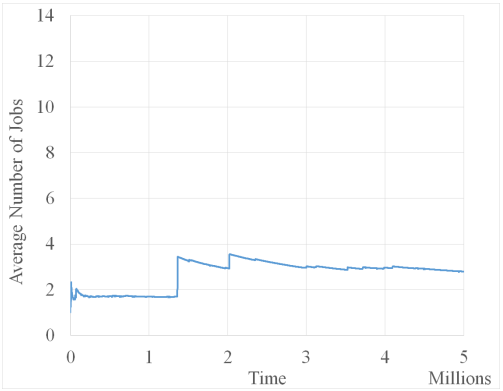
α=1.5, L=1, $U = 10^6$, Simulation Length=5M, Percent Error=[-50%,0%]



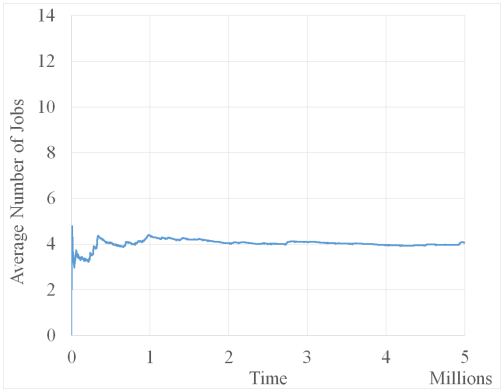
(a) SRPT
Load = 70%



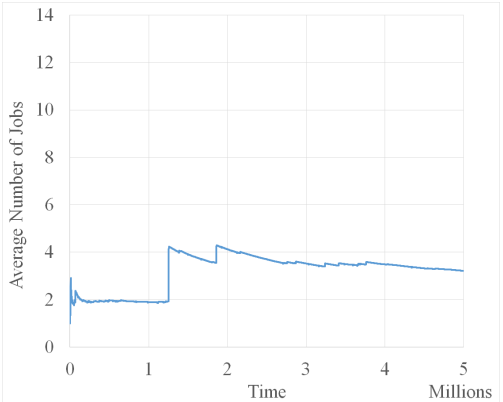
(b) Class-Based
Load = 70%, 10 classes



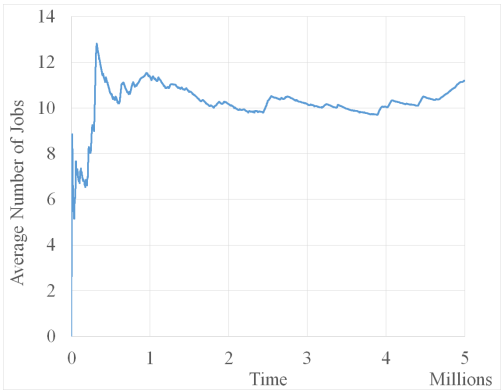
(c) SRPT
Load = 85%



(d) Class-Based
Load = 85%, 10 classes



(e) SRPT
Load = 95%



(f) Class-Based
Load = 95%, 10 classes

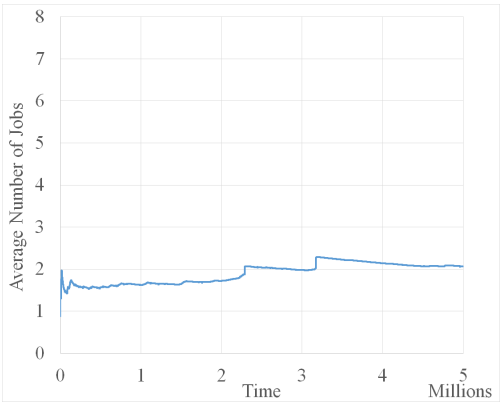
Figure A.3: Single server average number of jobs results 3

Appendix B

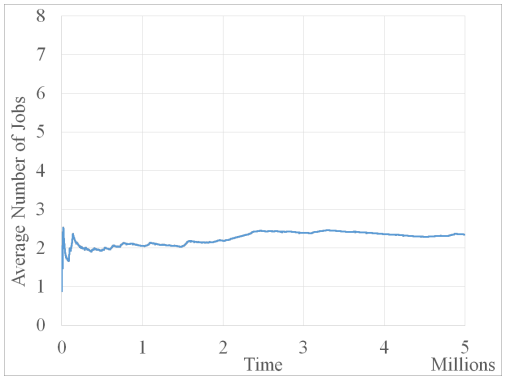
Multi-Server Results

Parameters

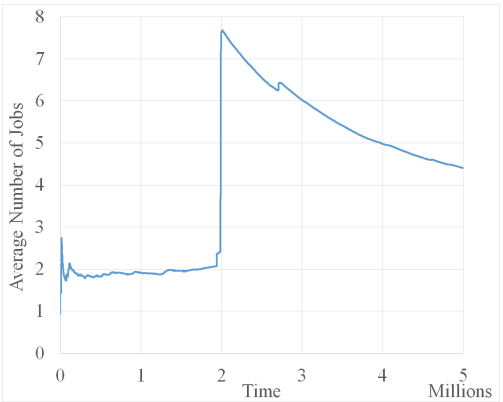
α=1.1, L=1, $U = 10^6$, Sim Length=5M, Percent Error=[-50%,0%], 2 servers



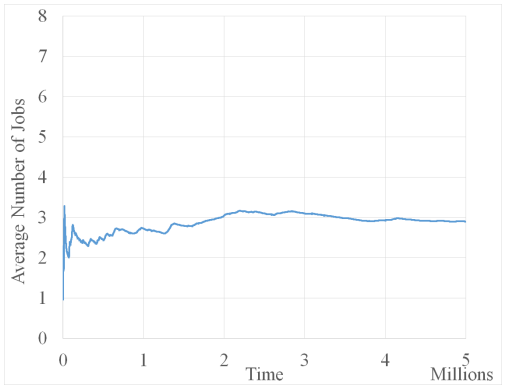
(a) SRPT
Load=70%



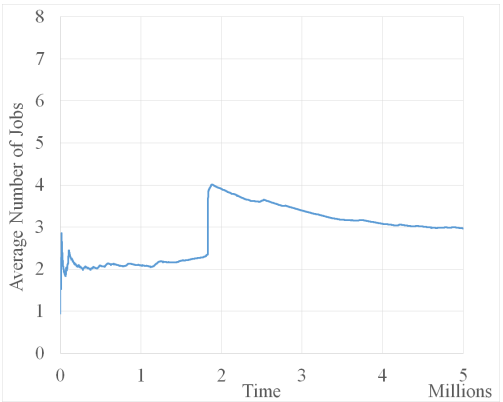
(b) Class-Based
Load=70%, 10 classes



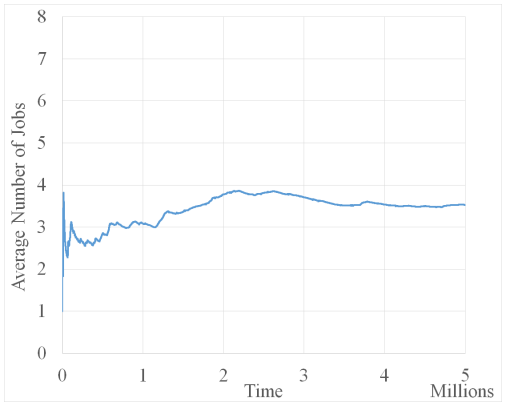
(c) SRPT
Load=85%



(d) Class-Based
Load=85%, 10 classes



(e) SRPT
Load=95%

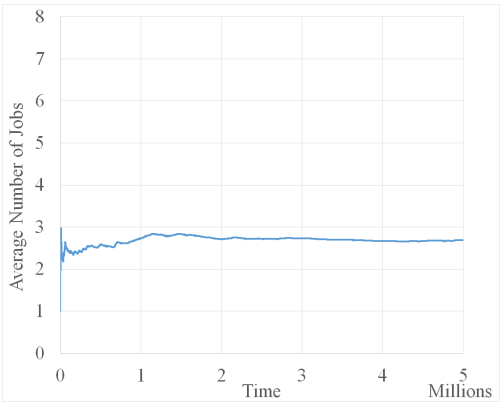


(f) Class-Based
Load=95%, 10 classes

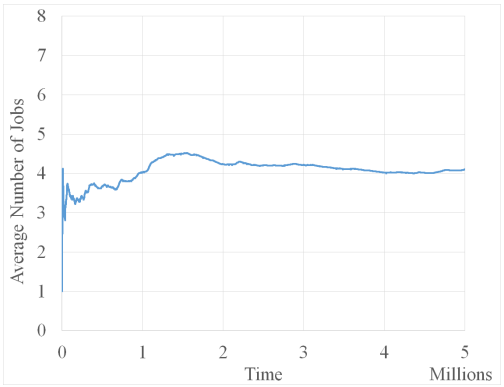
Figure B.4: Multi-server average number of jobs results 1

Parameters

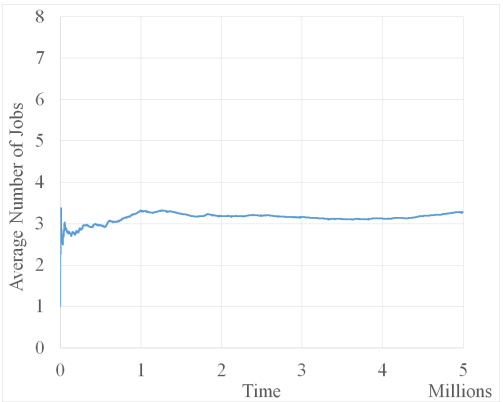
α=1.1, L=1, $U = 10^6$, Sim Length=5M, Percent Error=[-50%,0%], 5 servers



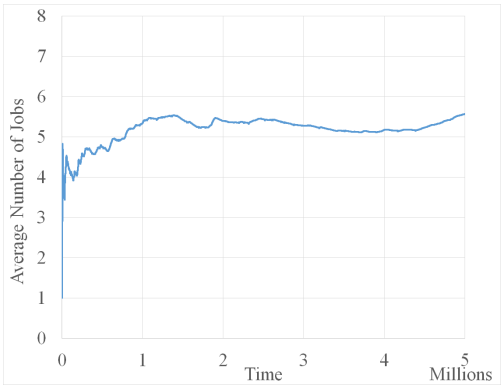
(a) SRPT
Load=70%



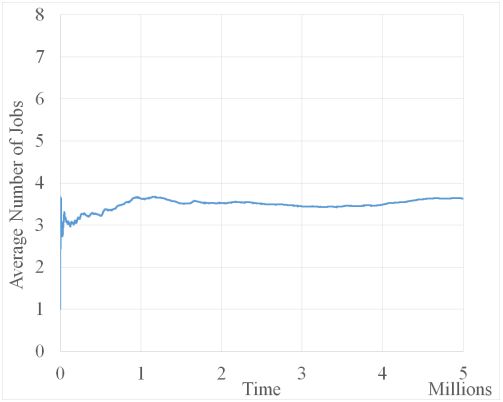
(b) Class-Based
Load=70%, 10 classes



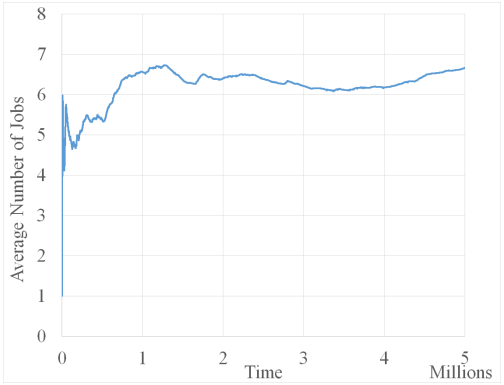
(c) SRPT
Load=85%



(d) Class-Based
Load=85%, 10 classes



(e) SRPT
Load=95%

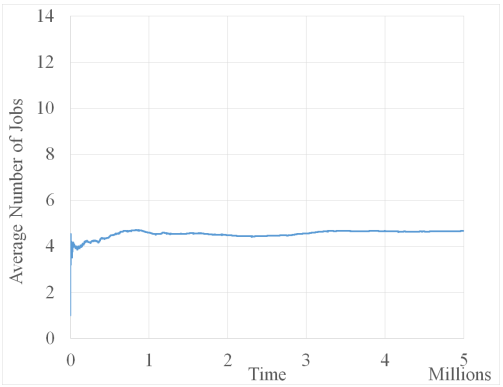


(f) Class-Based
Load=95%, 10 classes

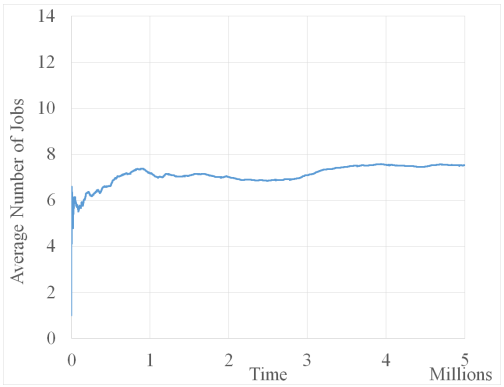
Figure B.5: Multi-server average number of jobs results 2

Parameters

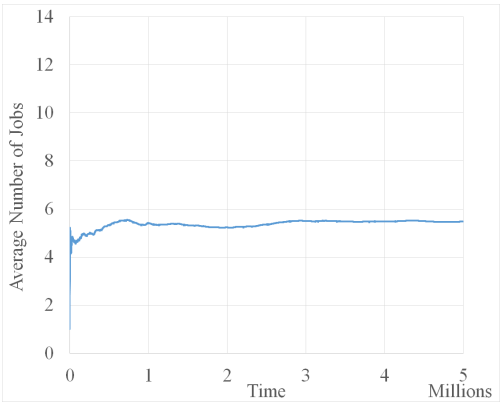
α=1.1, L=1, $U = 10^6$, Sim Length=5M, Percent Error=[-50%,0%], 10 servers



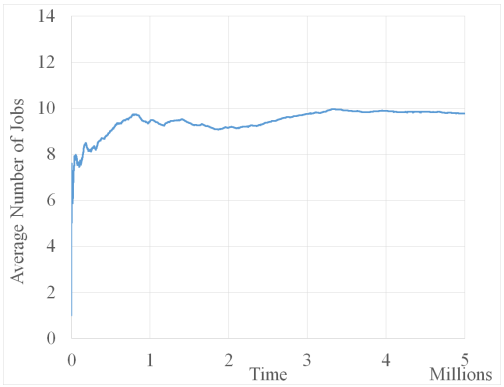
(a) SRPT
Load=70%



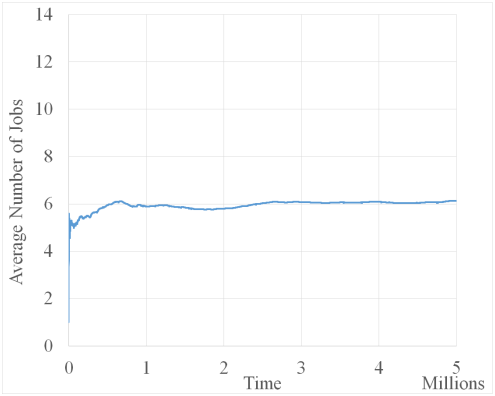
(b) Class-Based
Load=70%, 10 classes



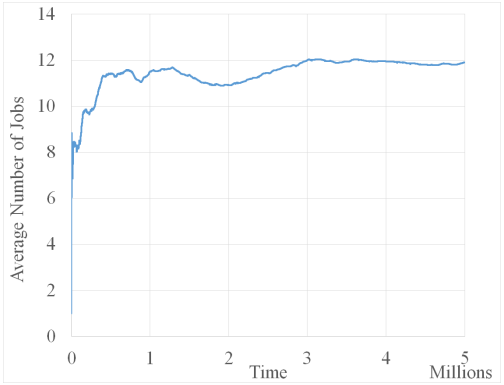
(c) SRPT
Load=85%



(d) Class-Based
Load=85%, 10 classes



(e) SRPT
Load=95%

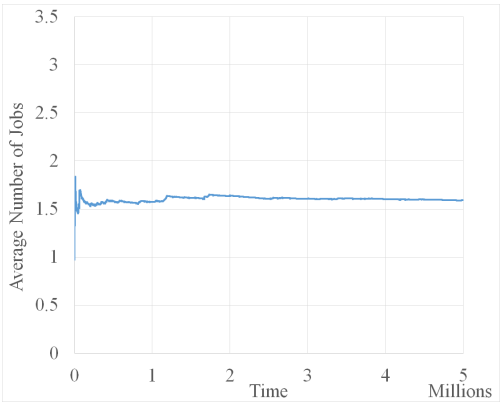


(f) Class-Based
Load=95%, 10 classes

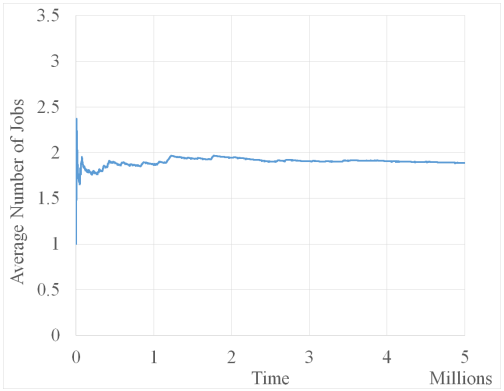
Figure B.6: Multi-server average number of jobs results 3

Parameters

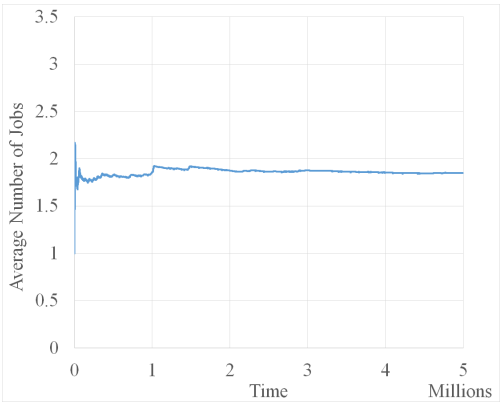
α=1.3, L=1, $U = 10^6$, Sim Length=5M, Percent Error=[-50%,0%], 2 servers



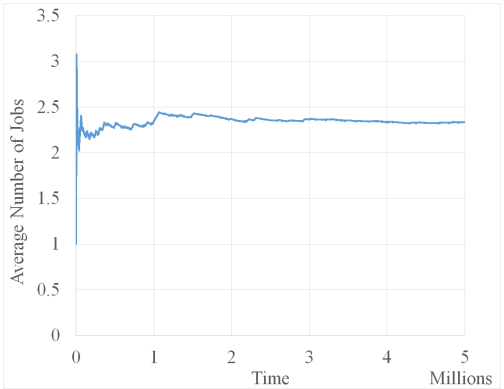
(a) SRPT
Load=70%



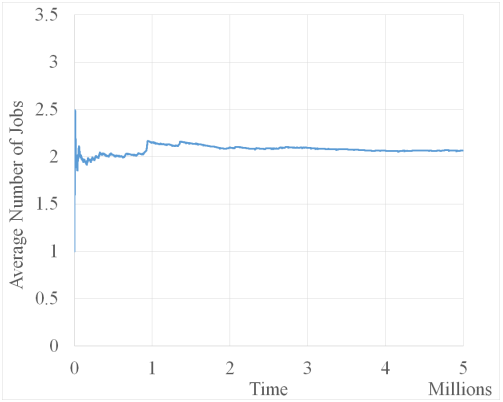
(b) Class-Based
Load=70%, 10 classes



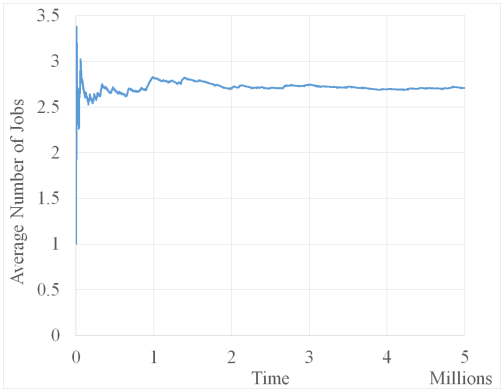
(c) SRPT
Load=85%



(d) Class-Based
Load=85%, 10 classes



(e) SRPT
Load=95%

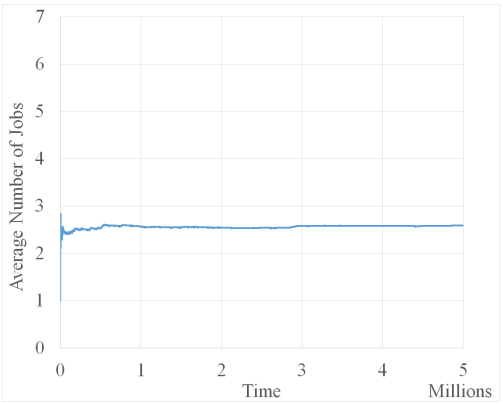


(f) Class-Based
Load=95%, 10 classes

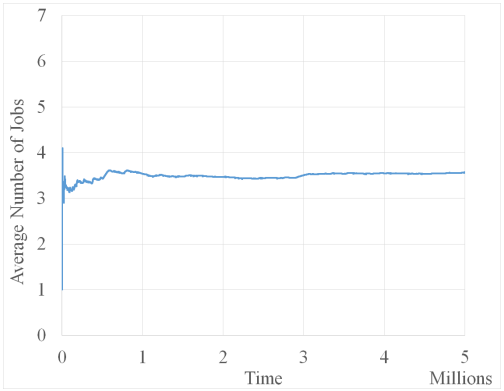
Figure B.7: Multi-server average number of jobs results 4

Parameters

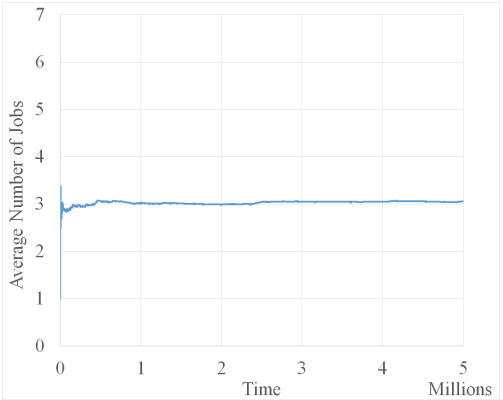
α=1.3, L=1, $U = 10^6$, Sim Length=5M, Percent Error=[-50%,0%], 5 servers



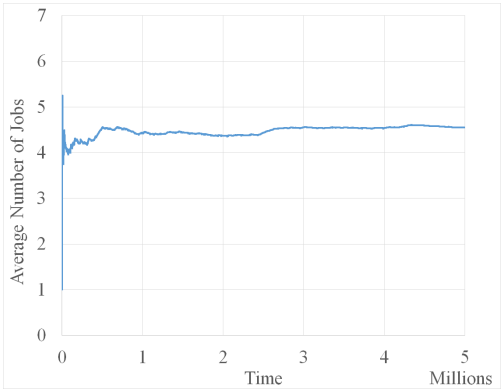
(a) SRPT
Load=70%



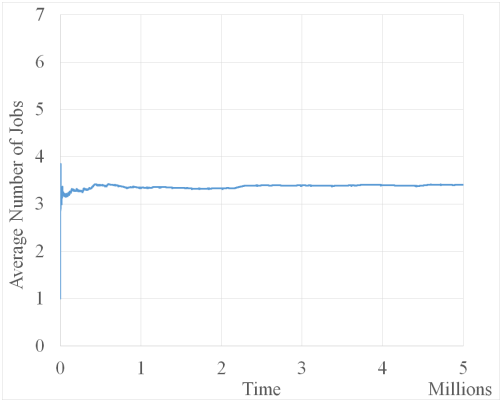
(b) Class-Based
Load=70%, 10 classes



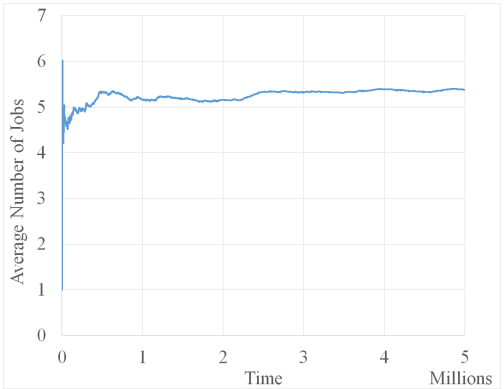
(c) SRPT
Load=85%



(d) Class-Based
Load=85%, 10 classes



(e) SRPT
Load=95%

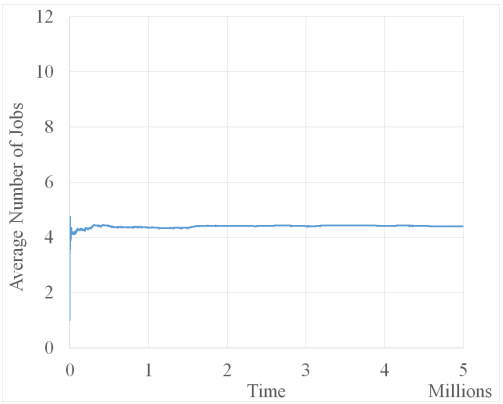


(f) Class-Based
Load=95%, 10 classes

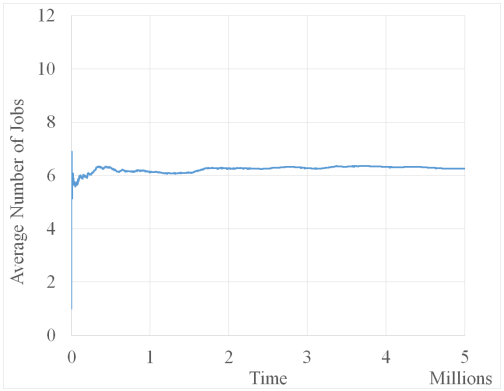
Figure B.8: Multi-server average number of jobs results 5

Parameters

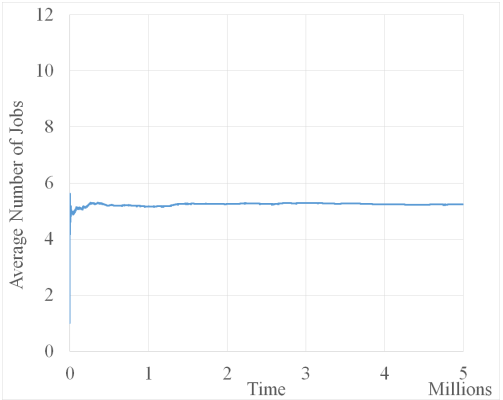
α=1.3, L=1, $U = 10^6$, Sim Length=5M, Percent Error=[-50%,0%], 10 servers



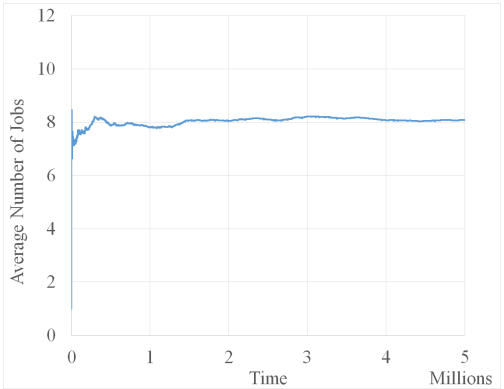
(a) SRPT
Load=70%



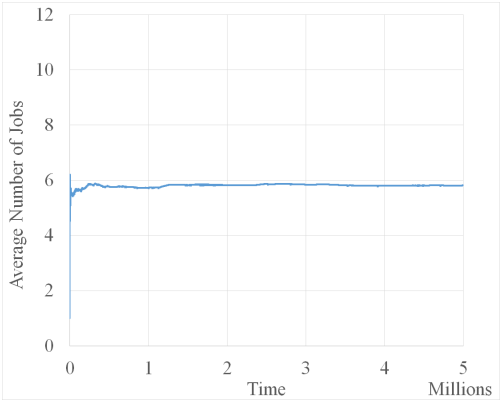
(b) Class-Based
Load=70%, 10 classes



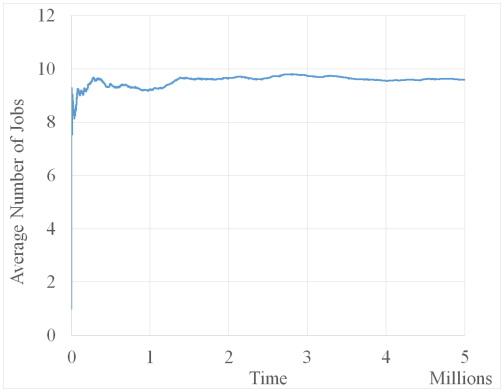
(c) SRPT
Load=85%



(d) Class-Based
Load=85%, 10 classes



(e) SRPT
Load=95%

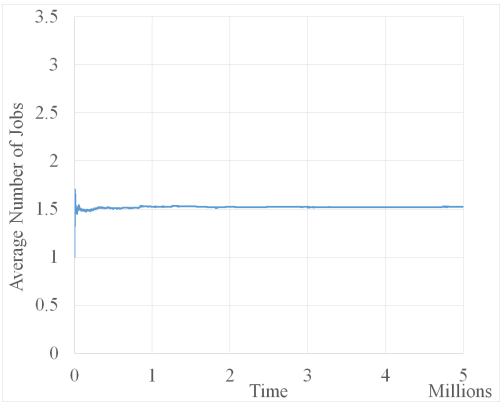


(f) Class-Based
Load=95%, 10 classes

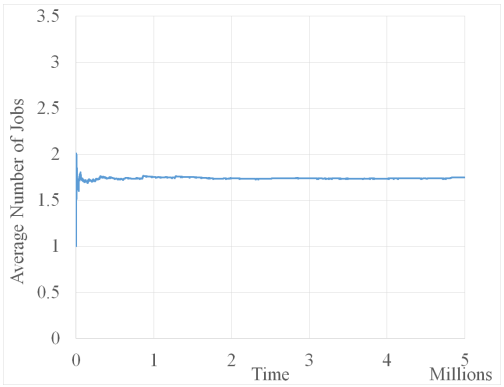
Figure B.9: Multi-server average number of jobs results 6

Parameters

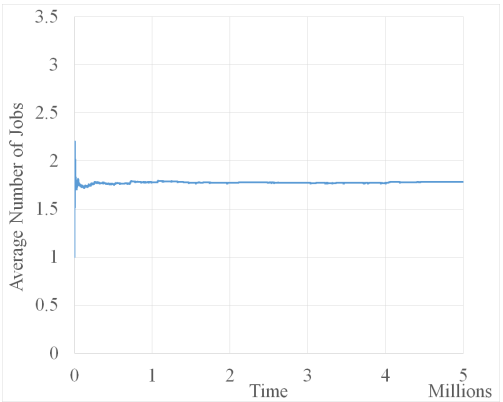
α=1.5, L=1, $U = 10^6$, Sim Length=5M, Percent Error=[-50%,0%], 2 servers



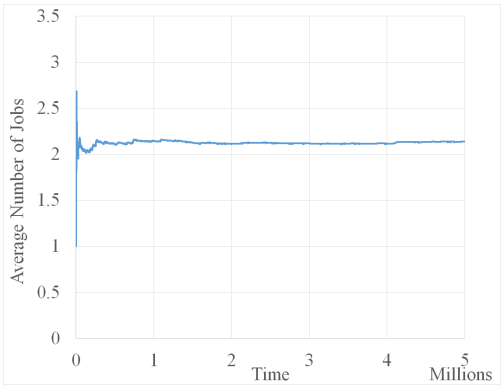
(a) SRPT
Load=70%



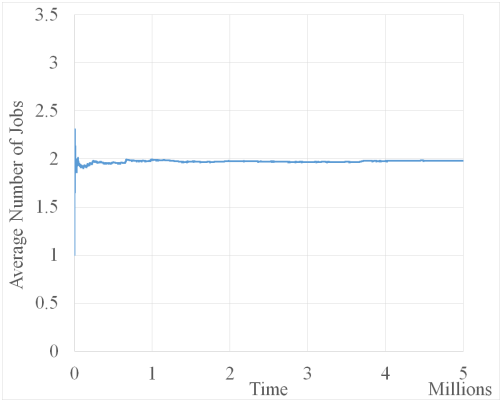
(b) Class-Based
Load=70%, 10 classes



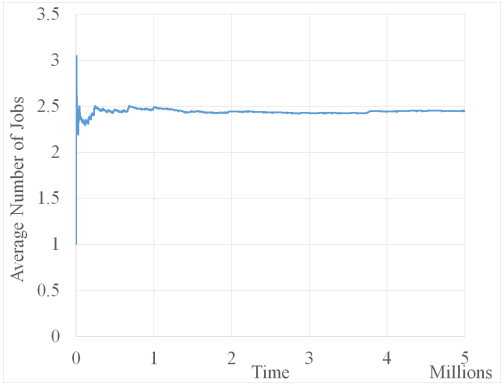
(c) SRPT
Load=85%



(d) Class-Based
Load=85%, 10 classes



(e) SRPT
Load=95%

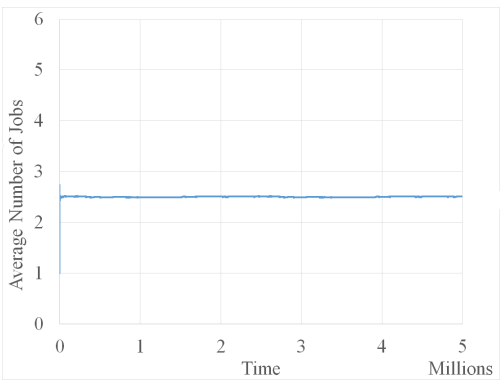


(f) Class-Based
Load=95%, 10 classes

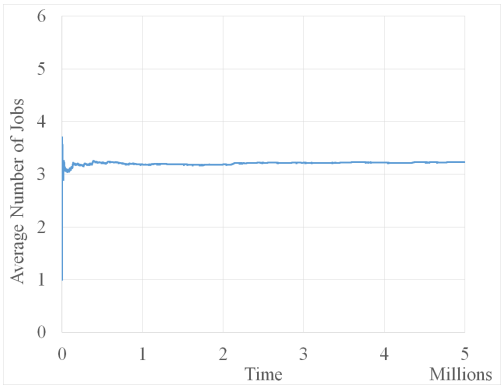
Figure B.10: Multi-server average number of jobs results 7

Parameters

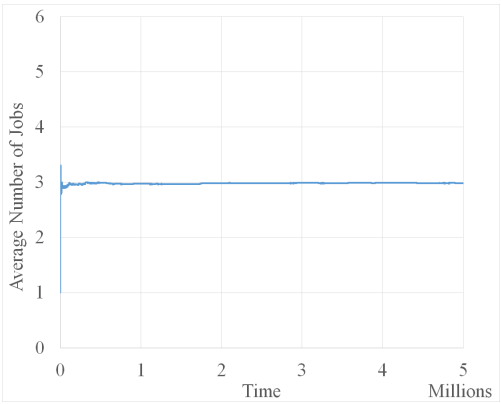
α=1.5, L=1, $U = 10^6$, Sim Length=5M, Percent Error=[-50%,0%], 5 servers



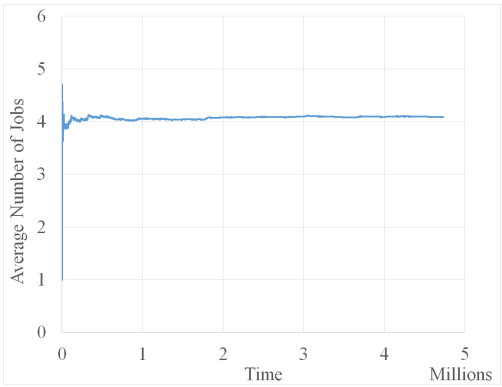
(a) SRPT
Load=70%



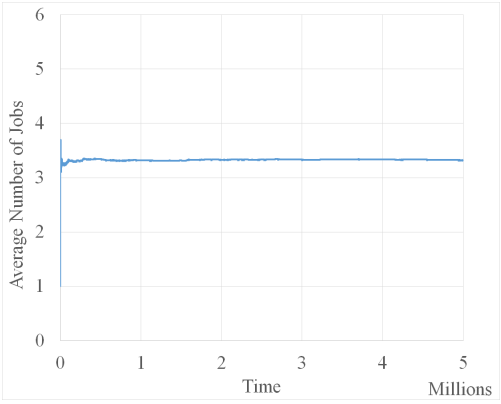
(b) Class-Based
Load=70%, 10 classes



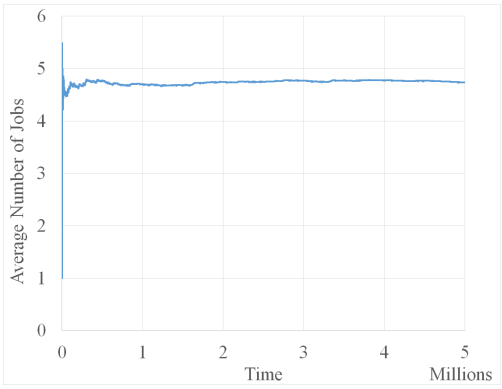
(c) SRPT
Load=85%



(d) Class-Based
Load=85%, 10 classes



(e) SRPT
Load=95%

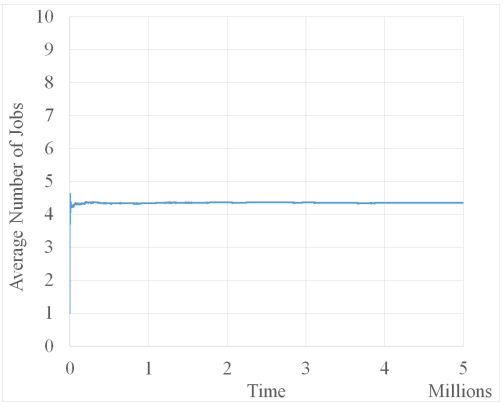


(f) Class-Based
Load=95%, 10 classes

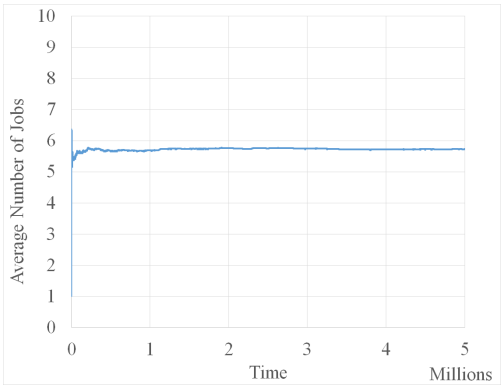
Figure B.11: Multi-server average number of jobs results 8

Parameters

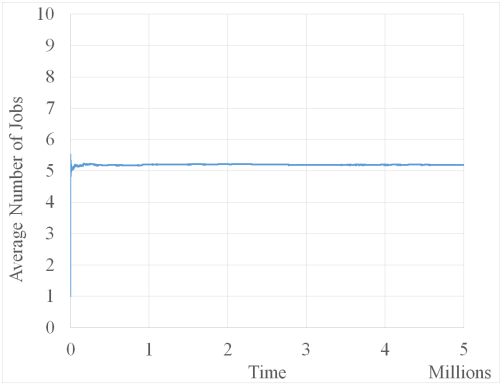
α=1.5, L=1, $U = 10^6$, Sim Length=5M, Percent Error=[-50%,0%], 10 servers



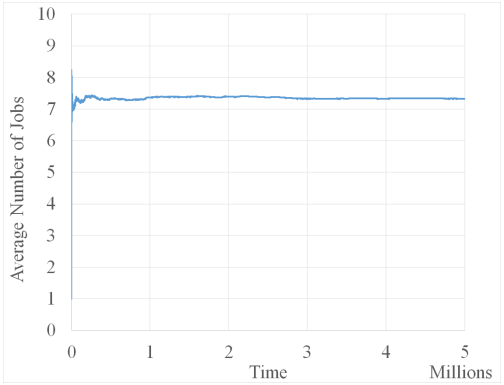
(a) SRPT
Load=70%



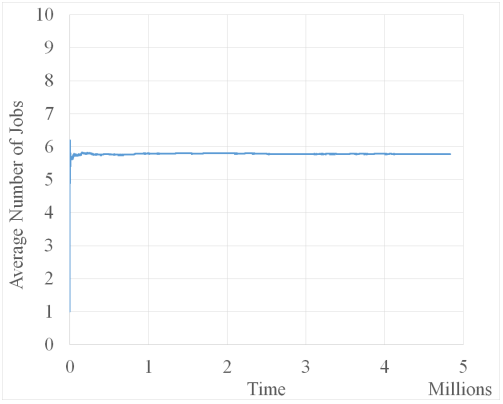
(b) Class-Based
Load=70%, 10 classes



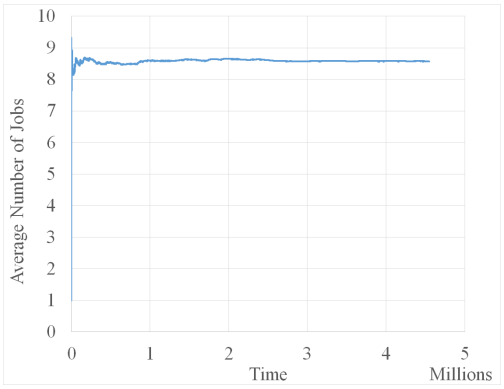
(c) SRPT
Load=85%



(d) Class-Based
Load=85%, 10 classes



(e) SRPT
Load=95%



(f) Class-Based
Load=95%, 10 classes

Figure B.12: Multi-server average number of jobs results 9

Bibliography

[1] J. Little, "A proof for the queuing formula: L = λW," *Operations Research*, vol. 9, no. 3, pp. 383–387, 1961.

[2] L. Wang, J. Chen, and W. Jie, *Quantitative quality of service for grid computing: Applications for Heterogeneity, Large-Scale Distribution, and Dynamic Environments*. Hershey, New York: Information Science Reference, 2009.

[3] B. Schroeder and M. Harchol-Balter, "Web servers under overload: How scheduling can help," *ACM Transactions on Internet Technology (TOIT)*, vol. 6, no. 1, pp. 20–52, 2006.

[4] J. Tao, "A better online algorithm for the parallel machine scheduling to minimize the total weighted completion time," *Computers & Operations Research 43*, pp. 215–224, 2014.

[5] P. Jelenkovic, X. Kang, and J. Tan, "Adaptive and scalable comparison scheduling," *Proceedings of the 2007 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems - SIGMETRICS '07*, 2007.

[6] M. E. Crovella and A. Bestavros, "Self-similarity in world wide web traffic evidence and possible causes," *Proceedings of the 1996 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems*, pp. 160–169, 1996.

[7] M. Harchol-Balter, "The effect of heavy-tailed job size distributions on computer system design," *Proceedings of ASA-IMS Conference on Applications of Heavy Tailed Distributions in Economics, Engineering and Statistics*, 1999.

[8] M. E. Crovella, M. S. Taqqu, and A. Bestavros in *A Practical Guide to Heavy Tails*, ch. "Heavy-tailed probability distributions in the world wide web", pp. 3–25, Cambridge, MA, USA: Birkhauser Boston Inc., 1998.

[9] M. Harchol-Balter and A. Downey, "Exploiting process lifetime distributions for dynamic load balancing," *ACM Transactions on Computer Systems (TOCS)*, pp. 253–285, 1997.

[10] V. Paxson and S. Floyd, "Wide-area traffic: The failure of poisson modeling," *IEEE/ACM Transactions on Networking (ToN)*, pp. 226–244, 1995.

[11] M. E. Crovella, M. Harchol-Balter, and C. D. Murta, "Task assignment in a distributed system: Improving performance by unbalancing load," *Boston University Computer Science Department*, 1997.

[12] N. Bansal and M. Harchol-Balter, "Analysis of SRPT scheduling: Investigating unfairness," *Proceedings of ACM SIGMETRICS 2001 Conference on Measurement and Modeling of Computer Systems*, pp. 279–290, 2001.

[13] M. Harchol-Balter, B. Schroeder, N. Bansal, and M. Agrawal, "Size-based scheduling to improve web performance," *ACM Transactions on Computer Systems (TOCS)*, vol. 21, no. 2, pp. 207–233, 2003.

[14] L. Schrage, "A proof of the optimality of the shortest remaining processing time discipline," *Operations Research*, vol. 16, no. 3, pp. 687–690, 1968.

[15] S. Leonardi and D. Raz, "Approximating total flow time on parallel machines," *STOC '97 Proceedings of the Twenty-Ninth Annual ACM Symposium on Theory of Computing*, pp. 110–119, 1997.

[16] D. Lu, H. Sheng, and P. Dinda, "Size-based scheduling policies with inaccurate scheduling information," *IEEE 12th International Symposium on Modelling, Analysis and Simulation of Computer and Telecommunication Systems*, 2004.

[17] E. J. Friedman and S. G. Henderson, "Fairness and efficiency in web server protocols," *ACM SIGMETRICS Performance Evaluation Review*, vol. 31, no. 1, p. 229237., 2003.

[18] A. Wierman and M. Nuyens, "Scheduling despite inexact job-size information," *ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems*, pp. 25–36, 2008.

[19] M. Dell'Amico, D. Carra, M. Pastorelli, and P. Michiardi, "Revisiting size-based scheduling with estimated job sizes," *IEEE 22nd International Symposium on Modelling, Analysis and Simulation of Computer and Telecommunication Systems*, 2014.

[20] R. W. Conway, W. L. Maxwell, and L. W. Miller, *Theory of Scheduling*. Addison-Wesley, 1967.

[21] L. Kleinrock, *Queueing Systems*, vol. II: Computer Applications. Wiley-Interscience, 1976.

[22] M. Harchol-Balter, *Performance Modeling and Design of Computer Systems*. 32 Avenue of the Americas, New York NY 10013-2473, USA: Cambridge University Press, 2013.

[23] I. Adan and J. Resing, "Queueing systems." `http://www.win.tue.nl/~iadan/queueing.pdf`, 2015.

[24] L. Kleinrock, *Queueing Systems*, vol. I: Theory. Wiley-Interscience, 1975.

[25] R. Mailach, "Discrete event simulation - single-server SRPT and class-based policies." `https://github.com/rsmailach/SRPT`, 2015-2016.

[26] R. Mailach, "Discrete event simulation - multi-server SRPT and class-based policies." `https://github.com/rsmailach/MultiServerSRPT`, 2015-2016.

[27] D. G. Down and R. Wu, "Multi-layered round robin routing for parallel servers," *Queueing Systems: Theory and Applications*, vol. 53, no. 4, pp. 177–188, 2006.