Scheduling Jobs with Estimation Errors for Multi-Server Systems

Rachel Mailach
Department of Computing and Software
McMaster University
Hamilton, Ontario
Email: mailacrs@mcmaster.ca

Douglas G. Down
Department of Computing and Software
McMaster University
Hamilton, Ontario
Email: downd@mcmaster.ca

***Abstract*—When scheduling single server systems, Shortest Remaining Processing Time (SRPT) minimizes the number of jobs in the system at every point in time. However, a major limitation of SRPT is that it requires job processing times *a priori*. In practice, it is likely that only estimates of job processing times are available. This paper proposes a policy that schedules jobs with estimated job processing times. The proposed Modified Comparison Splitting Scheduling (MCSS) policy is compared to SRPT when scheduling both single and multi-server systems. In the single server system we observe from simulations that the proposed scheduling policy provides robustness that is crucial for achieving good performance. In contrast, in a multi-server system we observe that robustness to estimation errors is not dependent on the scheduling policy. However, as the number of servers grows, SRPT becomes preferable.**

I. Introduction

If processing times are known *a priori*, scheduling using the Shortest Remaining Processing Time (SRPT) policy is proven to be optimal by Schrage [1] in 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 job interarrival times or processing times. SRPT is an appropriate scheduling policy for systems where dynamic scheduling is required, such as web servers, where job processing times are known on arrival. There is no need to have a predetermined list of jobs and their processing times before run-time. Web servers are quite fitting for this domain and we will continue to use them to illustrate some of the issues that may arise in a practical setting.

Even though SRPT is optimal when scheduling for a single server system, it may be worth noting some of its disadvantages. The first drawback, noted by Bansal and Harchol-Balter [2], is that it may force a large amount of preemption. If this preemption has high overhead cost this can affect the system response time. Response time for a single job is the time between the arrival of a job to its departure. For the remainder of this paper, it will be assumed that preemption has a low overhead cost and is not a critical problem, however, it is still an issue to consider.

The second concern in using SRPT is that it may be complicated to implement. It requires a large volume of storage as each job must be stored along with its processing time. If maintaining data for every job is a problem, one could use a class-based approximation of SRPT. This is the primary motivation for Jelenkovic et al. [3] when designing the Comparison Splitting scheduling policy, where the complexity of implementation is a critical concern. They are interested in finding a threshold-based policy that approximates SRPT. Static thresholds are those where values are calculated and set in advance and do not change over time. Conversely, dynamic thresholds are adaptive and change over time based on the job size distribution. In order to design a class-based scheduling policy with static thresholds, one must know the job processing time distribution in order to calculate threshold bounds. 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; hence dynamic thresholds are a better choice for job size classification.

The assumption that job processing times are known exactly is problematic in general. Thus, a current area of study is working with estimated job processing times. In web server systems, exact job sizes may be known, however, the exact time to process these requests may be unknown. As noted by Schroeder and Harchol-Balter [4], this is due in part to the inherent latency when sending data over a network and while sending and receiving acknowledgments, therefore processing times must be estimated. Web servers often use virtual machines to process jobs, thus another reason that job processing times must be estimated is the variance of the performance of virtual resources. When job processing times are estimated, errors are introduced. If the scheduling policy is dependent on the job processing times being known, estimation errors may cause performance issues.

Figure 1 displays an example in which a job enters the system with an actual processing time of one million time units. Here, the updated real remaining processing (RRPT) is maintained for illustration. Upon arrival, the job is underestimated by 1% and thus has an estimated remaining

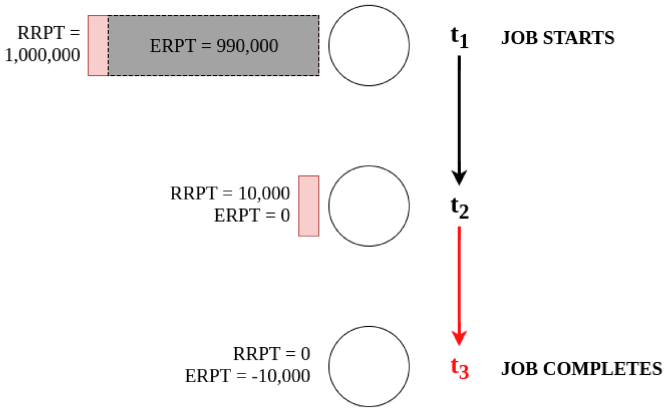


Fig. 1: Example of a large job underestimated by 1%

processing time (ERPT) of $990,000$. Under SRPT, 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.

When the processing time of a job is underestimated, under SRPT the job may block arriving jobs. If the ERPT of a job becomes zero, the job will gain the highest priority within the system until that job completes. It will occupy the server and will block any arrivals from processing, since no job can arrive with a job processing time or ERPT less than zero. This effect is exacerbated as the processing time of the job grows.

Scheduling with errors in job processing time estimates was first studied by Lu et al. [5]. They examined the performance of SRPT and the Fair Sojourn Protocol (FSP) for jobs with estimated processing times. FSP, described by Friedman and Henderson [6], works by simulating a system as if it were running using Processor Sharing (PS). A list is compiled sorting jobs in order by the time necessary to complete under PS. The jobs are processed according to how they are ordered in the list. FSP was chosen to be compared with SRPT in work by Lu et al. [5] as they have both demonstrated, by simulations, to outperform PS in terms of mean response time. Job processing time estimation errors were also discussed by Wierman and Nuyens [7]. Their analysis determined a bound on the level of error in job processing time estimates that a 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, we describe two in this paper. The first method is re-estimation of job processing times, periodically, or once a job becomes late. Late jobs are jobs that have been underestimated and have been processed until their ERPT is less than or equal to zero. An alternate approach is to assign late jobs special treatment. Dell'Amico et al. [8] 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 scheduling policy is then generalized to use Discriminatory Processor Sharing (DPS), which allows increasing priority for the most important jobs.

We present an alternative method for scheduling with estimated job processing times, a change in scheduling policy. This approach was chosen so we could examine the effect of lateness and how poor job processing time estimates may become before the mean response time becomes too high. We are interested in observing the difference between a scheduling policy designed to handle errors and one that is not. A class-based scheduling policy based on the work done by Jelenkovic et al. [3] is used to contrast SRPT. This scheduling policy is designed to avoid the issue of underestimated jobs. While an approximation of SRPT will not perform as well as SRPT in an ideal setting, its performance degrades more gradually in the face of estimation errors.

SRPT is not an optimal scheduling policy for multi-server systems, as shown by Leonardi and Raz [9]. At the time of writing, there is no known optimal scheduling policy for scheduling multi-server systems.

We make the following contributions: first, we study the Comparison Splitting scheduling policy proposed by Jelenkovic et al. [3], and observe that it may not approximate SRPT well for high variance job processing time distributions. This is due to much of the overall load being contained in the class with the largest jobs, so we suggest that changing the scheduling policy from First-Come, First-Served (FCFS) to Last-Come, First-Served (LCFS) for this class can result in significant performance improvement. Secondly, for single server systems, we demonstrate that while the performance of SRPT is highly sensitive to estimation errors, the Modified Comparison Splitting Scheduling (MCSS) policy has significantly lower sensitivity to such errors. Finally, we provide results that suggest that multi-server systems are inherently more robust to estimation errors, regardless of whether SRPT or the MCSS policy is employed.

The organization of the paper is as follows: Section II presents the description and analysis of the MCSS and SRPT policies for a single server. Section III introduces the issues and benefits for a multi-server system under both SRPT and the MCSS policies. In Section IV we present conclusions and in Section V, there is a discussion of potential future work.

II. Single Server Systems

Two separate policies are discussed in this section: SRPT and the approximate, or Modified Comparison Splitting Scheduling (MCSS) policy. Each scheduling policy is applied

in a single server system where jobs arrive following a Poisson Process and are assigned a required processing time that is calculated with respect to a Bounded Pareto distribution (details of the Bounded Pareto distribution will be given below). The required processing time is then multiplied by a random percent error value to simulate the estimation errors for approximating job processing times.

A. Policies

SRPT Scheduling Policy — The queue is sorted in increasing order of Estimated Remaining Processing Time (ERPT) and the job with the current shortest ERPT is always processing. ERPT is calculated from taking the initial processing time estimate of the job and subtracting from it the elapsed processing time. The initial estimates are never updated. When a job completes and leaves the system, the job next in the queue, with the shortest ERPT, will be sent to the server until the next event, a job arrival or departure, occurs.

Algorithm 1 Assign class

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

Modified Comparison Splitting Scheduling Policy — As described by Jelenkovic et al. [3], this scheduling policy is designed to approximate the SRPT scheduling policy — the class assignment process is called Comparison Splitting. The class assignment algorithm is given in Algorithm 1. The jobs are divided between the desired number of classes, r, using dynamic thresholds. When a job enters the system, its estimated processing time is compared with the estimated processing times of the previous $r-1$ jobs. The list of jobs is sorted from 1 through r by ERPT. The location of the current job in the sorted list will become its assigned class. The thresholds are dynamic as the class a job might be assigned to is dependent on the jobs that arrive before it. Two jobs of the same size may arrive to the system at two different points in time and be assigned to different classes. It is important to note that during the class assignment process, no jobs have their assigned classes updated. It may be more accurate to say that this scheduling policy is an approximation to Shortest Processing Time (SPT) than an approximation to SRPT, as jobs never change classes, even after a reduction in ERPT.

Each class at the server has its own queue. The classes are processed in order of priority, with class 1 having the highest priority and class r the lowest. The priorities are preemptive. When a job is preempted, its accumulated processing is not lost. Jobs within each class are processed in order of First-Come, First-Served (FCFS). Processing in FCFS order allows less data to be maintained and updated for each job, thus simplifying the scheduling policy.

High variance distributions have often been chosen to model job processing times as they provide an accurate representation of reality [10], [11], [12], [13], [14]. We focus on such distributions for the reason that the issues caused by job processing time estimates are only a concern when using high variance job size distributions, such as those that exhibit heavy-tailed behavior. These issues appear as a result of large jobs that are underestimated and not dealt with correctly. We have observed that such problems do not arise for lower variance distributions [15].

The details of the processing time distribution used in our simulations, Bounded Pareto, are as follows. The probability density function $B(L, U, \alpha)$ is described in Equation (1).

$$f(x) = \begin{cases} \dfrac{\alpha L^{\alpha} x^{-\alpha-1}}{1-(\frac{L}{U})^{\alpha}}, & L \le x \le U \\ 0, & otherwise \end{cases} \tag{1}$$

This distribution has three parameters, α, U, and L. U is the upper bound of the possible job processing times, we use a value of 10^6. L is the lower bound of the possible job processing times, we use a value of 1. The parameter α determines the shape of the tail of the distribution. The second moment, and thus for this distribution the variance increases as α decreases as defined by Crovella et al. [16]:

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

We used three different values for α to diversify the variance of the processing time distribution: 1.1, 1.3, and 1.5. In this paper we are focusing on high variance distributions, so only the results for $\alpha = 1.1$ are presented. The results for the lower variance simulations can be found in [15].

In a system with no estimation errors, as the number of classes approaches infinity, we approach SPT. Figure 2 shows the Comparison Splitting scheduling policy with an increasing number of servers and no processing time estimation errors. As the number of classes increases, 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. Another downside to maintaining many classes is that if the processing time distribution changes, it

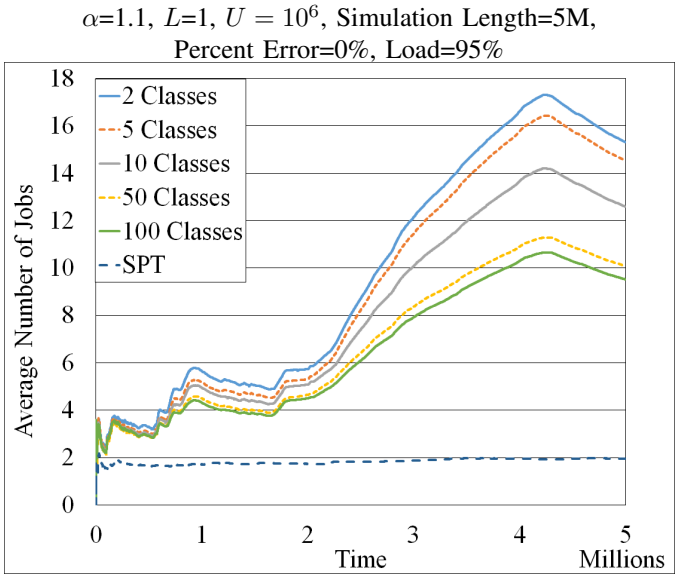


Fig. 2: A comparison of the average number of jobs in the system under varying number of classes using the Comparison Splitting scheduling policy

would take longer for the system to adapt. For our simulations we choose to use 10 classes as it maintains a reasonable amount of prior processing time data without requiring more storage.

As displayed in Figure 2, the gap between the Comparison Splitting Scheduling policy and SPT arises due to jobs that are queued in class r for long periods of time. This is because class r 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. The class assignment algorithm tends to assign significant load to the last class; this issue is not addressed by Jelenkovic et al. [3]. Class r has the largest range of processing times within its threshold. This is a problem when using FCFS to process the class r queue as FCFS is known to perform poorly if the variance is high. 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. The classes do not have equal load in simulation, as illustrated in Figure 3. This figure shows the percentage of the total load carried by the jobs of each class. It is clear that the last class tends to carry a significant load no matter the size of r.

In our experiments, classes 1 through $r-1$ tended to have low variance, class r had high variance. The scheduling policy must take variance in processing times into account in order to optimize mean response time. The Comparison Splitting scheduling policy was adapted so that the lowest priority class is processed using Last-Come, First-Served (LCFS) and will be henceforth be called the MCSS policy. This reasoning is supported when looking at a side-by-side comparison of LCFS and FCFS for a typical case of class r in Figure 4.

Preemptive LCFS has been studied and analysed by Harchol-Balter et al. [17] and found to have the same mean response time as PS. PS and LCFS are both size-blind policies and thus do not know the sizes of arriving jobs. They also show that PS has a lower mean response time than FCFS if the variance of the service time distribution is higher than the variance would be for an exponential distribution.

There are two main disadvantages of LCFS compared to FCFS, the first being that LCFS requires 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 if there is 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 processing time distribution. In our system, only class r has high enough variance to warrant using LCFS over FCFS. Section II-B provides more insight into this topic.

B. Condition for Preferring LCFS

Consistent with the method and notation described by Adan and Resing [18], define R_i to be the remaining processing time and B_i the service time for a job of class i. The variance of B_i is represented by σ_i^2.

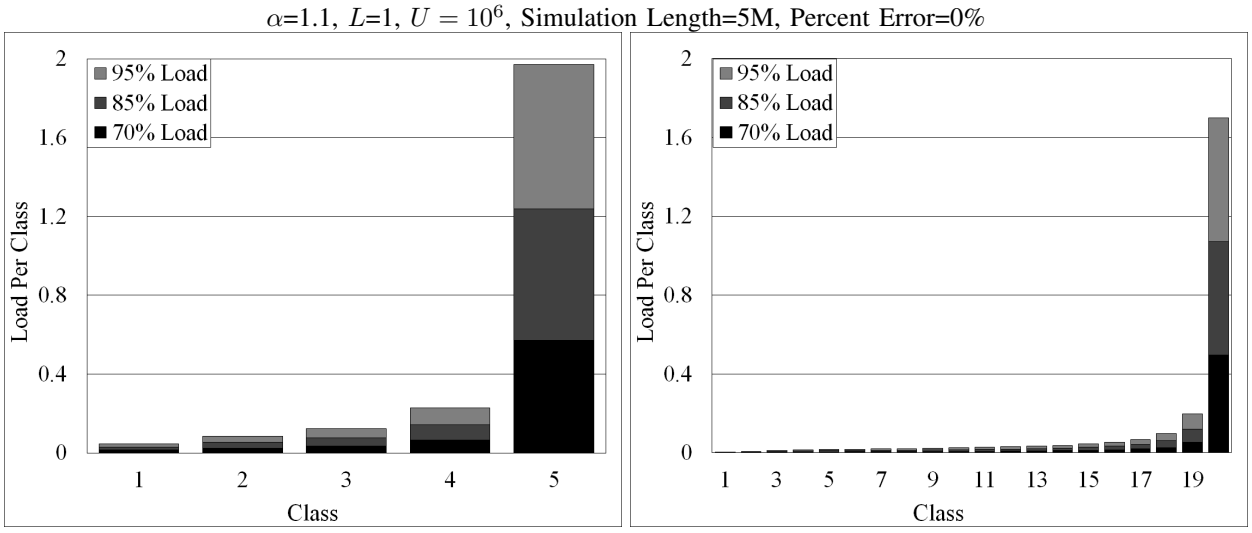
$$E[R_i] = \frac{E[B_i]}{2} + \frac{\sigma_i^2}{2E[B_i]} \tag{2}$$

In Equation (2), it is clear that $E[R_i]$ is dependent on $E[B_i]$ and σ_i^2. This is important as the condition for when LCFS is preferable over FCFS in Inequality (3) below, is dependent on the variance of B_i. The utilization of class j is denoted by ρ_j calculated by multiplying the arrival rate to class j, λ_j, by the mean service time, $E[B_j]$.

LCFS is preferred over FCFS for class r if and only if:

$$\begin{aligned}\rho_r(E[R_r] - E[B_r]) \\ \geq \left(1 - \sum_{j=1}^{r-1}\rho_j\right)\sum_{j=1}^{r-1}\left(\rho_j^2\left(\frac{\sum_{k=1}^{j}E[R_k]}{(1-\sum_{k=1}^{j}\rho_k)(1-\sum_{k=1}^{j-1}\rho_k)}\right)\right) \\ - \sum_{j=1}^{r-1}\rho_j E[R_j].\end{aligned} \tag{3}$$

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



(a) Number of Classes=5

(b) Number of Classes=20

Fig. 3: A comparison of the average load per class using the Comparison Splitting scheduling policy

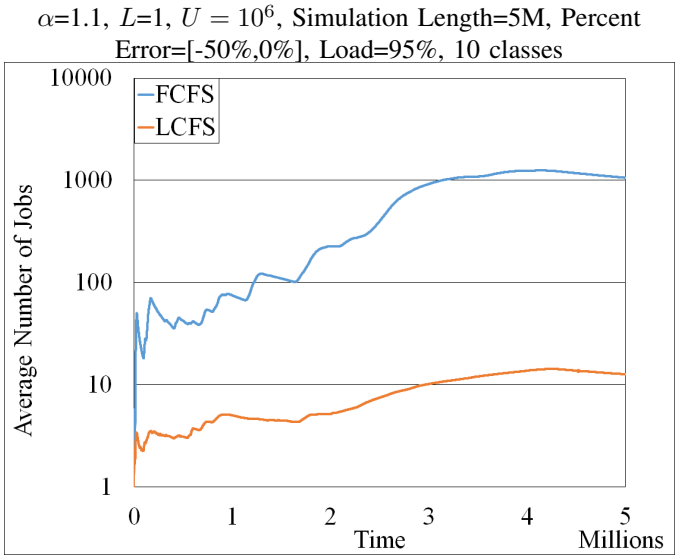


Fig. 4: A comparison of the average number of jobs using the Comparison Splitting scheduling policy, where class r is processed according to LCFS and FCFS

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, the r^{th} class is the only class that has a high enough variance to warrant using LCFS. A detailed proof and additional discussion can be found in [15].

C. Results and Discussion

For this and subsequent simulations, simulation code was written using the Python 3 language and is available for download at [19] and [20]. The simulations were each run using the SRPT and the MCSS policies. Common random numbers were used for the simulations.

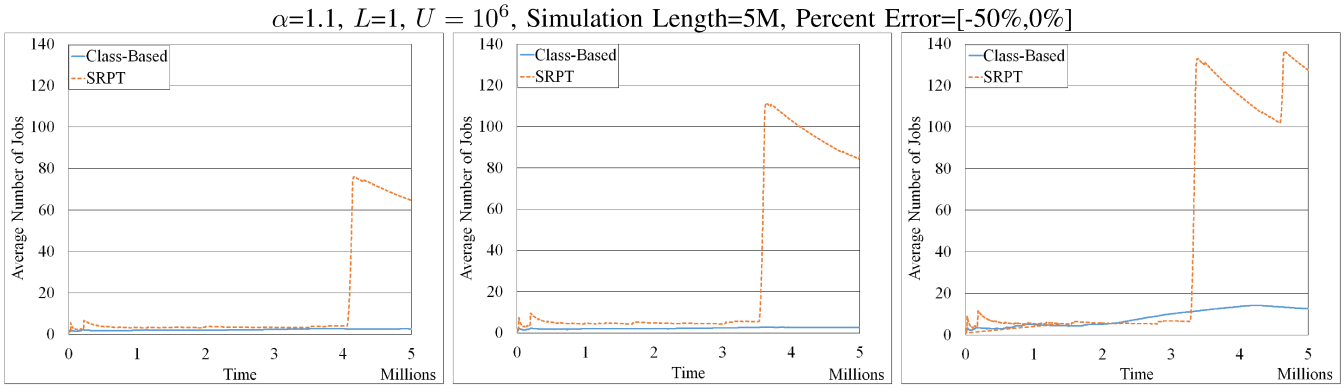
Figure 5 shows the comparison between the SRPT scheduling policy and the MCSS policy for three different loads. In all three graphs there are large jumps in the average number of jobs when using the SRPT scheduling policy, but the jumps are almost non-existent when using the MCSS policy.

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, they are given the highest priority. They block jobs that have true processing times 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.

The MCSS policy performs much better than SRPT because the large jobs that are underestimated are still sent to the lowest priority class, class r. Since the MCSS 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 MCSS policy is much more robust to estimation errors than SRPT.

III. Multi-Server Systems

This section presents the analysis of SRPT and the MCSS 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 is scaled up. In the course of doing this examination, we found that multi-server systems are naturally more robust to estimation errors.



(a) Load = 70% (b) Load = 85% (c) Load = 95%

Fig. 5: Average number of jobs for a single server system under SRPT and MCSS policies

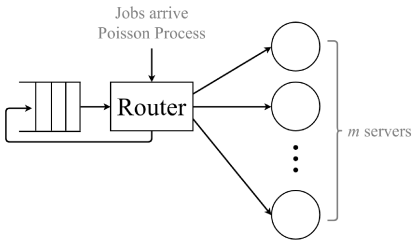


Fig. 6: SRPT multi-server model

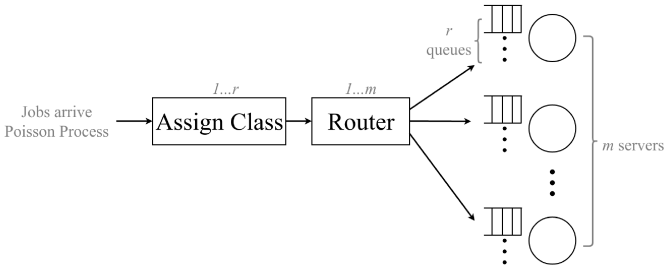


Fig. 7: MCSS multi-server model

A. *Policies*

SRPT Scheduling Policy — In the single server system, the MCSS policy could be compared to the optimal scheduling policy, SRPT. 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. Our suggested scheduling policy, the MCSS policy, is thus compared to SRPT, as no optimal benchmark exists to compare it to.

As depicted in Figure 6, jobs arrive to a system following an arbitrary arrival process. The m jobs with the smallest remaining ERPTs are served.

Modified Comparison Splitting Scheduling Policy — There are a few key differences between the two multi-server models, the SRPT model in Figure 6, and the MCSS model in Figure 7. The first difference between the two models is that the MCSS 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 by Down and Wu [21] to balance the load between the servers. Independent RR policies are implemented for each class. RR was chosen because it distributes the jobs 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 MCSS model, there are r queues at each server, where r is the number of classes. Each server in the model where the MCSS policy is used implements the single server MCSS policy independently.

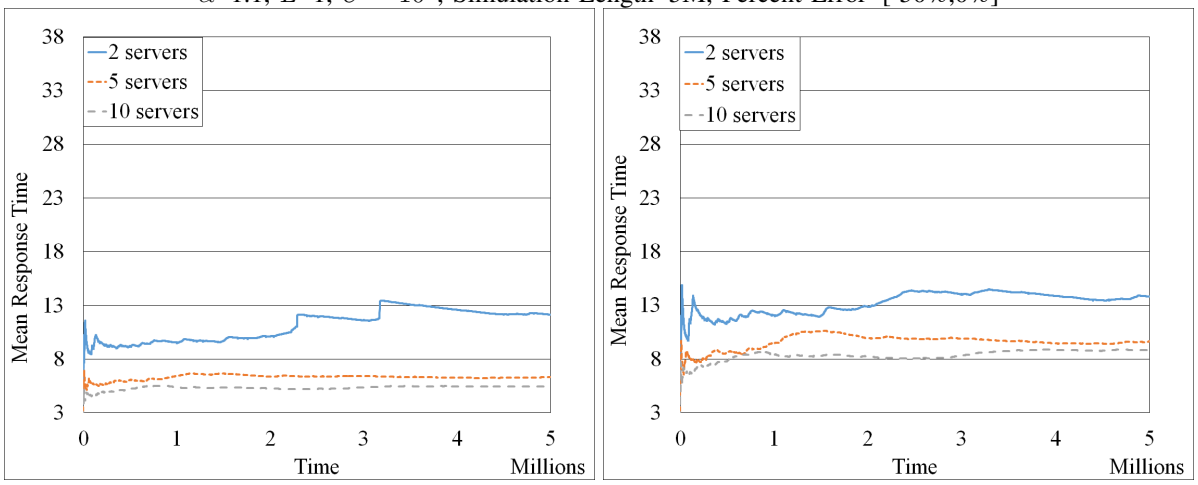
Number of Classes
The choice of number of classes for this scheduling policy has been discussed in Section II-C. The same principle applies in the multi-server scheduling policy as the single server scheduling policy is implemented independently m times.

B. *Results and Discussion*

In this section, the two policies are compared to determine if the MCSS policy may be an intelligent choice for scheduling a multi-server size-aware system.

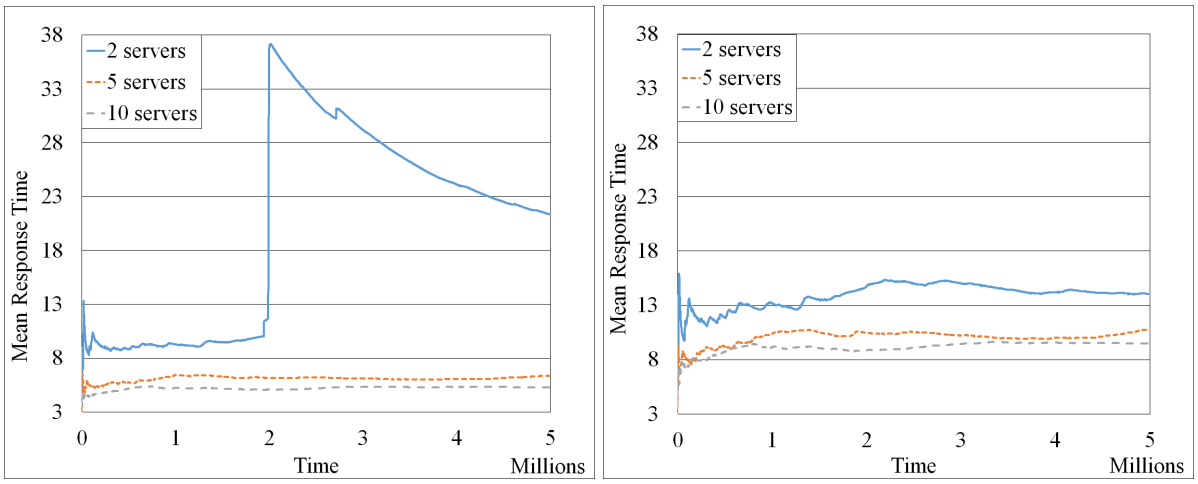
Figure 8 depicts how each scheduling policy behaves under different loads with increasing numbers of servers, using α equal to 1.1. Recall the closer α is to 1, the higher the variance in the job processing times. With the three different loads (8a, 8c, and 8e), the SRPT scheduling policy experiences some issues with underestimated jobs in the

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



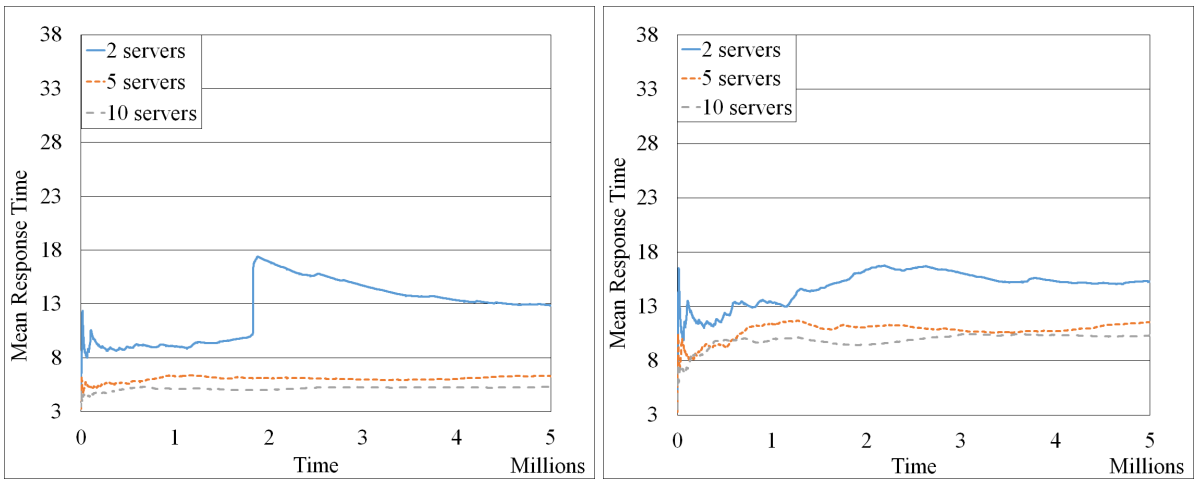
(a) SRPT
Load = 70%

(b) MCSS
Load = 70%



(c) SRPT
Load = 85%

(d) MCSS
Load = 85%



(e) SRPT
Load = 95%

(f) MCSS
Load = 95%

Fig. 8: Mean response time under various loads on various numbers of servers

two-server systems. The MCSS policy (8b, 8d, and 8f) has similar mean response times under all three loads.

In all cases except the two-class SRPT system under a load of 85%, the MCSS policy seems to have a slightly higher mean response time than the SRPT scheduling policy. One reason this may occur is because some jobs may be 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 for the difference in performance between the two policies may be because over short time periods some servers may be underutilized when using the MCSS policy. When using the SRPT scheduling policy, if there are any jobs in the queue, it means all servers are busy. Recall, in the MCSS 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 8c is a special case. The arrival streams for each of these graphs vary 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 mean response time. In the two-server case, it is fairly likely that this situation could arise. We investigate this issue in a little more detail below.

When inspecting the SRPT scheduling 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 mean response time decreases 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 mean response 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. For a single server system, we have seen that the MCSS policy is inherently more robust to estimation errors in performance than SRPT as any large jobs that are underestimated are still classified into class r. However, the situation changes for a multi-server system.

In 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 incoming 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 appears temporarily unstable if the $m-x$ servers can not handle the incoming load. For a fixed load, as m increases, it becomes more unlikely that x will ever be large enough to destabilize the system. While this effect seems quite intuitive, a more detailed analysis is desirable, but appears somewhat difficult to perform. This effect is also present in related work on stability and response time asymptotics for multi-server queues, see the work of Scheller-Wolf and Sigman [22] and Foss and Korshunov [23], respectively.

When comparing the two policies in Figure 8, as the number of servers increases, the mean response time under SRPT is lower than under the MCSS policy. A reason for this is the inefficient usage of servers in the MCSS policy. SRPT uses a global queue so there are no idle servers when the queue contains jobs. In the MCSS policy, there are some servers that have queues that are non-empty while there are idle servers in the system. There is a trade-off between the number of servers, and robustness to estimation errors. As the number of servers in the system grows, the effect of the idle servers from the MCSS policy becomes more of a concern than the benefits gained for robustness to estimation errors. Therefore, as the number of servers increases, SRPT outperforms the MCSS policy as the poor utilization of servers increases response time more than the delays caused by estimation errors.

When designing a system with estimated job processing times, it is thus advisable to use two or more servers if possible, as there is a large increase in robustness when moving from a single server system to a multi-server system. This robustness allows one to pick between using SRPT or a class-based scheduling policy without much difference in performance. As the number of servers used increases, the more desirable SRPT is for the system. If the architecture of the system only allows for one (or perhaps two) servers, one should consider a class-based scheduling policy.

IV. Conclusion

When considering the implementation of SRPT, 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 estimates may not be exact, thus, there will be some errors. These errors may cause response times to grow unacceptably large and must be considered when attempting to use the single server optimal scheduling policy. When moving to a multi-server system, we have demonstrated the increase in robustness to estimation errors.

We studied both single and multi-server systems with SRPT and a class-based scheduling policy. We were able to draw certain conclusions. The first is that a multi-server system appears to always be more robust to estimation errors than a single server system. The second conclusion that we can draw

is that the Modified Comparison Splitting Scheduling (MCSS) policy performs well without some of the disadvantages of SRPT. In the single server case we see a drastic increase in robustness when using the MCSS policy over SRPT. In the multi-server case, both policies preform relatively well but as the number of servers increases, the MCSS policy becomes less desirable.

V. Future Work

In the future, we would like to work on an improved classification scheme for jobs. As mentioned above, our assignment algorithm can classify small jobs to the lowest priority class and those jobs can be blocked by a large arrival. To rectify this issue, one method that could be investigated is designing a scheduling policy that can learn the job size distribution over time. This would allow for setting more precise thresholds as long as the distribution is relatively static. If the job size 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 on the threshold of class r. This would mean that no jobs smaller than this threshold value would end up in the last class and be potentially blocked by any extremely large jobs. At the moment, it is not entirely clear how to go about choosing this threshold.

As mentioned above, when there are an infinite number of classes in the MCSS policy (i.e. each job is assigned to its own class), it approaches Shortest Processing Time (SPT) which is itself an approximation of SRPT. It would be beneficial to compare the robustness of SPT to SRPT as it has many of the same advantages as the MCSS policy.

A third topic to investigate is the routing of jobs to servers. The main issue that arises when routing with Round Robin (RR) is that it allows for some servers to be idle, while others have a queue. One possible modification would be to only use RR if no servers are idle. Idle servers could ping the router and would be sent the next arriving job. It would be worthwhile to explore the PULL algorithm used by Stolyar [24], where the servers pull jobs from a pool. This algorithm works well when the system is scaled up, as the wait time becomes negligible

Acknowledgment

This research was supported by the Natural Sciences and Engineering Research Council of Canada.

References

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

[2] 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.

[3] 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.

[4] 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.

[5] 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.

[6] E. J. Friedman and S. G. Henderson, "Fairness and efficiency in web server protocols," *ACM SIGMETRICS Performance Evaluation Review*, vol. 31, no. 1, pp. 229–237, 2003.

[7] 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.

[8] 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.

[9] 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.

[10] 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.

[11] 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.

[12] 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.

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

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

[15] R. Mailach, "Robustness to estimation errors for size-aware scheduling," Master's thesis, McMaster University, 2016.

[16] 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.

[17] M. Harchol-Balter, *Performance Modeling and Design of Computer Systems*. New York, NY, USA: Cambridge University Press, 2013.

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

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

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

[21] 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.

[22] A. Scheller-Wolf and K. Sigman, "Delay moments for fifo gi/gi/s queues," *Queueing Systems*, vol. 25, no. 1, pp. 77–95, 1997.

[23] S. Foss and D. Korshunov, "Heavy tails in multi-server queue," *Queueing Systems*, vol. 52, no. 1, pp. 31–48, 2006.

[24] A. L. Stolyar, "Pull-based load distribution in large-scale heterogeneous service systems," *Queueing Systems*, vol. 80, no. 4, pp. 341–361, 2015.