Improving Nonpreemptive Multiserver Job Scheduling with Quickswap

Abstract

Modern data center workloads are composed of *multiserver jobs*, computational jobs that require multiple CPU cores in order to run. A data center server can run many multiserver jobs in parallel, as long as it has sufficient resources to meet their individual demands. However, multiserver jobs are generally *stateful*, meaning that job preemptions incur significant overhead from saving and reloading the state associated with running jobs. Hence, most systems try to avoid these costly job preemptions altogether. Given these constraints, a *scheduling policy* must determine what set of jobs to run in parallel at each moment in time to minimize the mean response time across a stream of arriving jobs. Unfortunately, simple non-preemptive policies such as First-Come First-Served (FCFS) may leave many cores idle, resulting in high mean response times or even system instability. Our goal is to design and analyze non-preemptive scheduling policies for multiserver jobs that maintain high system utilization to achieve low mean response time.

One well-known non-preemptive scheduling policy, Most Servers First (MSF), prioritizes jobs with higher core requirements and is known for achieving high resource utilization. However, MSF causes extreme variability in job waiting times, and can perform significantly worse than FCFS in practice. To address this issue, we propose and analyze a class of scheduling policies called *Most Servers First with Quick Swap* (MSFQ) that performs well in a wide variety of cases. MSFQ reduces the variability of job waiting times by periodically granting priority to other jobs in the system. We provide both stability results and an analysis of mean response time under MSFQ to prove that our policy dramatically outperforms MSF in the case where jobs either request one core or all the cores. In more complex cases, we evaluate MSFQ in simulation. We show that, with some additional optimization, variants of the MSFQ policy can greatly outperform MSF and FCFS on real-world multiserver job workloads.

1. Introduction

Modern data centers serve *multiserver jobs* that occupy multiple servers or CPU cores simultaneously [26, 35, 14, 13]. Each multiserver job has an associated *server need*, the number of servers the job requires to run, and *job size*, the amount of time the job must run to be completed. A set of multiserver jobs can run in parallel, but only if the system has enough dedicated servers for each job. A data center *scheduling policy* must select which jobs to run in parallel at every moment in time. Given a fixed number of servers, k, a common goal is to design a scheduling policy that minimizes the *mean response time* across jobs in a stream of arriving multiserver jobs — the average time from when a job arrives to the system until it is completed.

There are two central difficulties in designing performant scheduling policies for multiresource jobs. First, modern datacenter workloads generally exhibit large variability in both the server needs and sizes of their jobs [35]. As a result, it is usually impossible to utilize all available servers using the set of multiserver jobs currently in the system. In general, leaving more servers unutilized on average will lead to higher mean response time or even system instability. Unfortunately, maximizing the number of utilized servers at a specific moment in time requires solving a knapsack problem instance. It is even more difficult, then, to maximize the utilization of the available servers as jobs enter and exit the system over time.

Second, modern multiserver jobs are generally *stateful*, meaning that job preemptions require persisting and/or reloading a significant amount of program state [31]. As a result, job preemptions or migrations can take a prohibitively long amount of time to perform. Due to this overhead, data centers typically employ *non-preemptive* scheduling policies that avoid costly preemptions altogether. Given these two constraints, *this paper designs and analyzes new, non-preemptive scheduling policies that aim to minimize the mean response time across a stream of multiserver jobs.*

1.1. Prior Approaches

Much of the prior work on multiserver job scheduling uses frequent job preemptions to ensure that server utilization remains high as jobs enter and leave the system [11, 24, 20, 19]. These preemptive policies are of limited utility when processing the stateful jobs that are common in data centers.

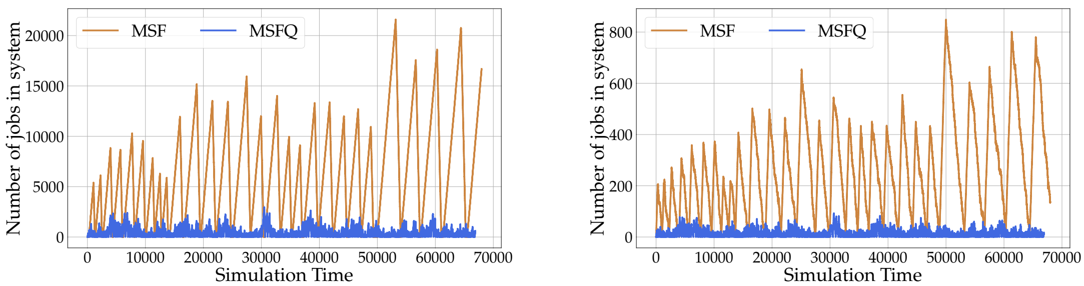
When it comes to non-preemptive policies, there are three central approaches suggested in the literature:

First-Come First-Served (FCFS) is a naïve non-preemptive policy that serves jobs in arrival order until the system runs out of available servers. For example, when a job with a large server need reaches the front of the queue, the system may not have enough available servers to fit this job in service. FCFS stops scheduling additional jobs at this point, even if other jobs in the queue could fit into service. This phenomenon, known as *Head-of-the-Line blocking*, causes FCFS to underutilize servers, resulting in high mean response time. Although FCFS is a simple policy, analyzing it has been proven difficult due to its dependence on the random arrival order of jobs with different server needs. Only recently, [23] derived mean response time bounds that confirm the empirical observation that FCFS performs poorly in practice.

Most Servers First (MSF) [34, 5, 20] is a non-preemptive policy that prioritizes jobs with larger server needs. Specifically, whenever the system has available servers, jobs are considered in descending server need order. Jobs that find their required number of servers being put in service successively. To understand both the benefits and the pitfalls of MSF, consider an example where jobs either need either 1 server or k servers. We refer to this case as the *one-or-all* case for multiserver jobs. In this case, MSF serves jobs in two alternating *phases*. First, MSF serves k-server jobs until none remain in the system. Then MSF serves 1-server jobs until none remain before returning to serve k-server jobs. We will show in Section 5.1 that, by switching between these two phases, MSF achieves optimal long-run average server utilization in the one-or-all case.

While one might hope that MSF leverages its high server utilization to achieve low mean response time, we also find that MSF takes an increasingly long time to switch phases as the job arrival rate increases (see Section 5.3). This creates a feedback loop in the system whereby many 1-server jobs accumulate while the system processes k-server jobs, leading to a long period of serving 1-server jobs during which many k-server jobs will accumulate. The two things to note about this process are that (i) despite its name, MSF can spend long periods of time giving priority to 1-server jobs over k-server jobs and (ii) because the class of jobs not in service accumulates quickly, there are almost always a large number of jobs in the system under MSF. Figure 1 illustrates this problem via simulations that track the number of jobs in the system under MSF for the one-or-all case. As MSF alternates between phases, jobs of the opposite class accumulate quickly in the queue. While all jobs are eventually served, this behavior ensures that a significant fraction of arriving jobs have long queueing times, leading to a high overall mean response time.

First-Fit is a variant of FCFS that avoids head-of-line blocking by continuing to examine jobs in arrival order after finding a job that does not fit in service. One might hope that this policy gets a near-optimal server utilization without the harmful periodic behavior of MSF. Unfortunately, in the one-or-all case, First-Fit has the same alternating behavior as MSF, but First-Fit spends even more time serving 1-server jobs.



(a) Number of small jobs vs time

(b) Number of large jobs vs time

Figure 1: Number of jobs in the system under MSFQ and MSFQ where there are k=32 servers, 90% of the job arrivals are 1-server jobs, mean job sizes are 1 for 1-server jobs and k-server jobs, and jobs arrive at a rate of 7.5 jobs/second.

1.2. A New Approach: Most Servers First with Quickswap

Our central observation is that, while MSF achieves high server utilization, it does a poor job of switching which class of jobs are in service. Specifically, in the one-or-all case, MSF ties the decision to switch from serving 1-server jobs to serving k-server jobs to the time required to drain all 1-server jobs from the system. This time, which is essentially a partial busy period in an $M/M/k$ queue, explodes as the arrival rate increases or as k becomes large [4], allowing many k-server jobs to accumulate in the system. To reduce mean response time, our goal is to maintain the high utilization of MSF while shortening the time the scheduling policy takes to switch between job classes.

With this motivation in mind, we propose a new class of policies called *Most Servers First with Quickswap (MSFQ)*, which is designed to improve the performance of MSF in the one-or-all case. Unlike MSF, which tries to drain all 1-server jobs from the system before switching, MSFQ switches to serving k-server jobs whenever the number of 1-server jobs in the system falls below a threshold, ℓ. That is, when the number of 1-server jobs falls below ℓ, MSFQ stops admitting jobs into service and lets all running 1-server jobs complete. MSFQ can then begin serving k-server jobs. By increasing ℓ, we can shorten the time MSFQ requires to switch phases, reducing the mean response time compared to MSF. Furthermore, we will prove that MSFQ maintains the high server utilization achieved by MSF.

Figure 1 compares the number of jobs in the system under the MSF and MSFQ policies (the yellow and blue curves, respectively). Setting a sufficiently high value of ℓ for MSFQ ($\ell = k - 1$ in this case) greatly dampens the feedback that causes jobs to accumulate under MSF. As a result, an MSFQ policy can achieve a much lower mean response time than MSF. We will also show that MSFQ policies are much better at balancing the mean response time between different job classes. While our MSFQ policy is tailored specifically to the one-or-all case, we will also explore generalizations of this policy to an arbitrary number of job classes.

1.3. Contributions

Inspired by the high server utilization of the MSF policy, this paper formally defines and analyzes the class of MSFQ policies for scheduling multiserver jobs. Our analysis has two main goals. First, we aim to prove *stability* results about MSFQ policies. We say the system is stable under a scheduling policy if the policy achieves sufficiently high server utilization such that the mean number of jobs in the system and the mean response time across jobs are both finite. Second, we will analyze the mean response time under MSFQ policies and their variants both in theory and in simulation to show the advantages of these policies compared to prior approaches. Specifically, the contributions of this paper are as follows.

- First, in Section 4.1, we formally explain the shortcomings of the MSF policy in the one-or-all case. Here, we show how excessively long periods of serving 1-server jobs cause a feedback effect that leads to poor mean response time.
- In Section 4.2 we introduce the MSFQ policy, which uses the Quickswap mechanism to force the policy to switch phases faster. Crucially, in Section 5.1 we show that any MSFQ policy matches the server utilization of MSF by proving that MSFQ is *throughput-optimal.* Here, throughput-optimality means that MSFQ will stabilize the system whenever the system can be stabilized.
- Then, in Section 5.3, we analyze the mean response time under MSFQ by approximating the Laplace Transform of the phase durations and number of jobs at the beginning of each phase. While the MSFQ system resembles a polling system (See Section 2.3), MSFQ cannot be analyzed using existing results from the polling literature. Hence, our analysis of MSFQ also represents a new contribution to the extensive body of work on polling systems.
- Finally, Section 6 examines generalizations of MSFQ to real-world settings where jobs' server needs can vary widely. We evaluate two generalizations of MSFQ, called *Static Quickswap* and *Adaptive Quickswap*, and evaluate these policies in simulation using traces from the Google Borg cluster scheduler [35]. Our simulations show that a well-designed Quickswap policy can improve mean response time by orders of magnitude. Furthermore, Quickswap policies tend to achieve an equitable mean response time between the job classes as compared to a less fair priority policy like MSF.

2. Related Work

We now describe prior work on multiserver jobs from the systems and theory communities in Sections 2.1 and 2.2, respectively. We also note that the Quickswap policies analyzed in this paper bear a resemblance to prior queueing-theoretic work on polling systems. However, because the connection between multiserver jobs and polling is somewhat indirect, we discuss the polling systems literature separately in Section 2.3.

2.1. Systems for Multiserver Job Scheduling

Modern data centers schedule multiserver jobs across thousands of machines, supporting workloads with diverse server needs and job sizes [27, 35, 37]. None of these schedulers make formal performance guarantees about system stability or mean response time, generally relying on heuristics to make scheduling decisions.

SLURM [27] is an open-source cluster scheduler used in data centers and high-performance computing environments. It uses a combination of heuristics and a variant of FCFS scheduling called BackFilling. While this approach can improve resource utilization by running low-priority jobs opportunistically, it requires accurate predictions of job sizes to work well, and can therefore suffer from low server utilization in practice. Borg [35], Google's internal resource management system for data centers, schedules batch jobs by placing incoming jobs in an FCFS queue. Once there is enough capacity to serve the next batch job, complex heuristics are used to assign the job to a specific set of servers. YARN [37], integrated with Hadoop, supports both FCFS and other heuristic policies aimed at optimizing fairness instead of stability or mean response time. Hence, many systems designed to schedule multiserver jobs stand to benefit from improved scheduling policies that are accompanied by formal performance guarantees.

2.2. Multiserver Job Scheduling in Theory

Prior work from the theory community on multiserver jobs has mostly focused on the stability and response time analysis of FCFS. The stability region of FCFS was studied in [33, 29, 1] in the case where all job sizes follow the same exponential distribution. Subsequently, [22, 30] considered the case where jobs belong to one of two job classes, deriving explicit expressions for the stability region of FCFS. For many years, mean response time analysis of FCFS was restricted to systems with just two servers [10, 15]. However, [23] recently derived explicit bounds on mean response time that are tight up to an additive constant. Matrix geometric approaches [2, 3] have also recently been used to characterize the performance of FCFS systems with two job classes under specific service time distributions. While FCFS is becoming well-understood, all of these analyses confirm that it can perform poorly in terms of both stability and mean response time.

There is comparatively little work on more complex and efficient policies that do not require job preemptions. For example, the well-studied MaxWeight policy is throughput optimal, but requires preemption and is computationally costly to implement in practice [28]. Other recent work on scheduling multiserver jobs has also been restricted to the case of preemptible jobs [21, 24]. There are two prominent examples of throughput-optimal, non-preemptive policies for scheduling multiserver jobs. First, Randomized Timers is a throughput-optimal policy based on MaxWeight that is non-preemptive [32]. Unfortunately, there is no known mean response time analysis of Randomized Timers, and the policy has been shown perform poorly in practice. Second, [12] recently analyzed a new class of non-preemptive policies called Markovian Service Rate (MSR) policies. An MSR policy precomputes a set of schedules with high server utilization, and switches between schedules according to a continuous-time Markov chain that is independent of the system state (i.e., queue lengths). The class of MSR policies is throughput-optimal and admits an analysis of mean response time. However, because MSR policies do not consider queue length when switching schedules, they waste capacity unnecessarily, resulting in high mean response time. We will show that MSFQ can significantly outperform MSR policies by considering queue length when switching schedules.

2.3. Polling Systems and Most Servers First

In the one-or-all setting, the MSF and MSFQ policies we study (see Sections 4.1 and 4.2) are theoretically similar to a two-station polling system with exhaustive service, where the first station serves k-server jobs, and the second serves 1-server jobs. Furthermore, the system incurs something like a switchover time when moving from 1-server jobs to k-server jobs.

The literature on polling systems is vast [6]. The single-polling-station and infinite-polling-station systems are well-understood [7, 17], and approximations for waiting time in the multiple-polling-station system have been established [8]. Stability issues caused by switchover times have also been studied [18, 16]. However, our multiserver job system considers a mix of single-server and multiserver operations not found in the polling literature. Furthermore, while MSF essentially uses an exhaustive service discipline for switching phases, the class of MSFQ policies uses a more generalized, threshold-based version of exhaustive service that is not analyzed in the prior work. Hence, the analysis of MSFQ in this paper also serves as a new contribution to the literature on polling systems.

3. Model

3.1. Multiserver Jobs

We consider a system equipped with k servers. A multiserver job can be represented by an ordered pair (x, s), where $x \in \{1, 2, \cdots, k\}$ is the number of servers it needs in order to run and s is the *service duration* (also known as *job size*), the time it needs to run on the servers before completion. Jobs occupy a fixed number of servers throughout their time in service, and cannot be preempted: once started, a job must be run until it is complete. We also refer to a job's server needs as its class. A class-i job is a job that requires i servers. We refer to this job model as the Multiserver Job (MSJ) model.

We consider a stream of multiserver jobs arriving into our system; class-i jobs arrive according to an independent Poisson process with rate λ_i. We assume the service durations of class-i jobs are i.i.d. random variables with exponential distribution, so that $S_i \sim \exp(\mu_i)$.

We define an *arrival rate vector* $\boldsymbol{\lambda} = (\lambda_1, \lambda_2, \cdots, \lambda_k)$ and a *completion rate vector* $\boldsymbol{\mu} = (\mu_1, \mu_2, \cdots, \mu_k)$. Let λ denote the total arrival rate of multiserver jobs into the system. Hence, $\lambda = \|\boldsymbol{\lambda}\|_1$. Let p_i be the fraction of arriving jobs belonging to class i. Equivalently, $p_i = \lambda_i/\lambda$.

We define a *feasible schedule* as a multiset of classes of multiserver jobs that can run in parallel, obeying the rule that the total number of servers requested does not exceed k. We use $\mathbf{u} = (u_1, u_2, \cdots, u_k)$ to denote a feasible schedule in the multiresource system, where it puts u_i class-i jobs in service and $\sum_{i=1}^{k} i\, u_i \leq k$, as the total server demand cannot exceed k.

In the one-or-all setting, $\lambda_i = 0$ for all $1 < i < k$ because jobs can only request one core or all cores in the system. In this case, $\lambda = \lambda_1 + \lambda_k$. A feasible schedule in this case can either be $u_k = 1$ and $u_i = 0$ for $i < k$, or $u_1 \leq k$ and $u_i = 0$ for $i > 1$.

A scheduling policy $\mathbf{u}(t) = (u_1(t), u_2(t), \cdots, u_k(t))$ picks a feasible schedule at every time t, subject to the requirement that no job is preempted: the service policy at time t must contain all jobs whose service has started but not completed by time t. We allow the scheduling policy to select a $u_i(t)$ value that exceeds the number of class-i jobs available for some job class i. In general, a scheduling policy may depend on the system state as well as policy-specific state.

We model the system state with a pair of vectors: The total *occupancy vector* $\mathbf{n}(t)$, where $n_i(t)$ is the number of class-i jobs in the system at time t, and the service vector $\mathbf{u}(t)$, chosen by the scheduling policy and discussed above. In addition, we allow a general policy-specific Markovian state z. The triple $(\mathbf{n}, \mathbf{u}, z)$ represents the system state and forms a countably infinite Markov chain.

For notational convenience, in the one-or-all setting, we neglect all the 0's in the vectors and abbreviate $\mathbf{n}(t)$ as $\mathbf{n}(t) = (n_1(t), n_k(t))$ and $\mathbf{u}(t)$ as $(u_1(t), u_k(t))$. Thus, we can represent the system state with a 5-tuple (n_1, n_k, u_1, u_k, z).

3.2. Stability Region

For a given policy p, the mean response time of the system may or may not be bounded. Equivalently, the system may or may not be positive recurrent. Considering the system where job size distributions and job classes are fixed and we vary the arrival rates, we define the set of arrival rates such that the mean response time is finite under policy p as the *policy stability region* $\mathcal{C}_p$. Formally, $\mathcal{C}_p = \{\boldsymbol{\lambda} \mid \mathbb{E}[T_p(\boldsymbol{\lambda})] < \infty\}$, where $T_p(\boldsymbol{\lambda})$ is the response time of multiserver jobs under policy p when the arrival rate vector is $\boldsymbol{\lambda}$. We also define the *system stability region* $\mathcal{C}$ as the set of arrival rates such that there exists a policy that stabilizes the system. Therefore, the system stability region is the union of the stability regions of all possible policies. When $\mathcal{C}_p = \mathcal{C}$ for some policy p, we say that the policy p is throughput-optimal. In other words, a policy p is throughput-optimal if it stabilizes the system whenever there exists a stable policy.

3.3. General Notation

We define the notation $X \Box Y$ as the independent sum of X copies of the random variable Y: $X \Box Y := \sum_{i=1}^{X} Y_i$, where $X \geq 0$ is an integer-valued random variable and the Y_i's are i.i.d samples of Y. We define the notation $(x)^+$ as the *positive part* of x. Specifically, $(x)^+ := \max(x, 0)$. We define the notation $\widehat{X}(z)$ as the z-transform of the probability mass function of an integer-valued random variable X and $\widetilde{Y}(s)$ as the Laplace-Stieltjes transform of the probability density function of a continuous random variable Y.

4. Policies

In this section, we define the policies we are using in the paper. We first define the Most Servers First (MSF) policy [5, 28], which is known to have shortcomings as discussed in Section 1. Then, based on the MSF policy, we develop the MSF Quickswap (MSFQ) policy for the setting where there are only class-1 and class-k jobs in the system. We analyze the mean response time of our MSFQ policy in Section 5.3 and show response time performance improvements with analysis and simulations in Section 6. Inspired by the improvements obtained with the MSFQ policy, we develop the Static Quickswap policy and the Adaptive Quickswap policy, each of which is a generalization of MSFQ to support general job class distributions. We show that the mean response time performance of Static Quickswap and Adaptive Quickswap each compares favorably to MSF in simulations based on real datacenter traces in Section 6.

4.1. Most Servers First

We analyze the Most Servers First (MSF) policy as described in [5, 28]. Specifically, we define MSF as a non-preemptive policy that favors jobs with the highest server demands. Whenever a job arrives or completes, MSF tries to put as many additional jobs as possible into service, starting with the job that demands the most servers and moving in descending order of server demands. This process ends when either all servers are utilized or MSF has considered all jobs in the queue.

In the one-or-all case where jobs either require 1 server of k servers, MSF has a somewhat simpler structure. At any moment in time, the policy either serves 1 class-k job or up to k class-1 jobs. The two job classes are never served simultaneously. We describe this structure by saying that MSF undergoes two *phases*. In phase 1, MSF serves exclusively class-k jobs one at a time. Doing so, MSF uses all servers in the system and is thus very efficient. In phase 2, MSF serves exclusively class-1 jobs. During phase 2, there can be up to k class-1 jobs in service, depending on how many class-1 jobs are in the system. Whenever the number of class-1 jobs in service is less than k, some of the servers' service capacity is wasted. MSF only switches between phases when it runs out of jobs of the current class.

Considering the phases of MSF in the two-class case demonstrates why this policy can lead to poor performance, particularly as load becomes high. The time to complete phase 1 looks like the busy period of an $M/M/1$ system started by the jobs that arrived in the prior phase 2. The time to complete phase 2 looks like the busy period of an $M/M/k$ system started by the jobs that arrived during the prior phase 1. As load increases, both phases will become longer, causing more class-1 jobs to arrive during phase 1 and vice versa. In this way, MSF amplifies the effect of increased load on mean response time.

4.2. Most Servers First Quickswap (MSFQ)

To reduce the load-amplifying effect found in MSF, we introduce the Most Server First Quickswap (MSFQ) policy in the one-or-all setting. The goal of MSFQ is to shorten the duration of phase 2 of the MSF policy so that fewer class-k jobs are allowed to build up during this phase. Specifically, an MSFQ policy is associated with a threshold, ℓ, that is used to shorten the periods where class-1 jobs are in service. When the number of class-1 jobs drops below ℓ, the system stops allowing class-1 arrivals to enter service.

We define the MSFQ policy formally via the following phases with a threshold $\ell \in [0, n-1]$:

- Phase 1: Serve class-k jobs exclusively until none remain ($n_k = 0$).
- Phase 2: Serve class-1 jobs until there are less than k class-1 jobs in the system ($n_1 < k$).
- Phase 3: Serve class-1 jobs until there are at most ℓ class-1 jobs in service ($n_1 \leq \ell$).
- Phase 4: Complete the class-1 jobs that are already in service ($n_1 = 0$ at the end of the phase). New class-1 arrivals are not allowed to enter service during this phase.

At the end of phase 4, the server returns to phase 1.

By analyzing response time of the MSFQ system in Section 5.3, we find that the mean response time is dependent on the number of jobs in the system at the beginning of the phase, and the *phase duration* — the amount of time from when the system enters phase i until phase i completes. We let random variable N_i^S denote the number of small (class-1) jobs and N_i^L denote the number of large (class-k) jobs at the beginning of phase i. We also let the random variable H_i denote the duration of the ith phase.

Note that, when $\ell = 0$, our MSFQ policy is the same as the MSF policy. We show that the class of MSFQ policies, irrespective of the threshold ℓ, is throughput optimal in Section 5.1. The intuition behind throughput-optimality follows from the fact that we always fully utilize all the cores outside of the switchover period (phases 3 and 4) when there are enough jobs waiting in the system. By searching over the choices of ℓ, we show that our MSFQ policy offers significant performance improvement over MSF. This confirms that the load-amplifying effect found in MSF has a huge impact on response time performance.

4.3. Static Quickswap

Our one-or-all MSFQ analysis relies heavily on the fact that there are only two classes of jobs in the system. The key difficulty in generalizing the MSFQ policy to general job class distributions lies in selecting the next job to serve, once the phase corresponding to the current class of jobs is complete.

We therefore define the *Static Quickswap* scheduling policy, which cycles through all classes of jobs in a fixed order, with the following two phases for each class of jobs:

- Working Phase: Run schedule $u_j = \lfloor k/i \rfloor \mathbb{1}\{j = i\}$ and serve class-i jobs exclusively until the number of idle servers exceeds $k - \ell$.
- Draining Phase: Complete the class-i jobs that are already in service. New class-i arrivals are not allowed to enter service during this phase.

When a given class's draining phase is complete, the next class in the cycle is served. We do not focus on the choice of cyclic ordering of the phases – we leave studying the effects of that ordering to future work.

In Remark 1, we give a proof sketch that the Static Quickswap policy achieves optimal stability region whenever all classes of jobs perfectly divide k, and thus fully utilize the k servers.

We will empirically show that the response time performance of Static Quickswap compares favorably to MSF in Section 6.

4.4. Adaptive Quickswap

Inspired by MSFQ and Static Quickswap, we develop a policy called Adaptive Quickswap, which allows multiple classes of jobs at the same time. In the general setting, if the number of servers required by a class does not perfectly divide the total number of servers, the system cannot fully utilize the server when serving some class of jobs exclusively. Hence, our Adaptive Quickswap policy prioritizes jobs to serve in MSF order and switches when it finds that serving these jobs becomes inefficient. The Adaptive Quickswap policy first admits jobs according to MSF order and then operates according to the following phases:

- Working Phase: Whenever servers become available, the job in the queue with the largest server need that is at most the number of unoccupied servers is admitted to service. This continues until the quickswap is triggered.

- Quickswap trigger: Switch from the working phase to the draining phase when there is a job class that is in the queue and not in service, and every job class in service has no jobs of that class waiting to receive service.
- Draining Phase: No jobs may enter service, except for the job in the queue with the largest server need. Once this job has entered service, switch to the working phase.

5. Analysis of the MSFQ Policy

In this section, we analyze the Most Servers First Quickswap (MSFQ) policy under the one-or-all setting, where there are only class-1 and class-k jobs. We call the class-1 jobs small jobs and the class-k jobs large jobs. We begin by proving that the MSFQ policy is throughput-optimal in the simplified setting.

Theorem 1 (MSFQ Throughput-Optimality)**.** *In the 2-class one-or-all MSJ system, the Most Servers First with Quickswap policy has optimal stability region for all thresholds ℓ, $0 \leq \ell < n$.*

We then analyze the mean response time under MSFQ. We observe that the mean response time depends on two central factors: the amount of time the policy spends in each phase, and how many jobs are in the system at the beginning of each phase. We therefore approximate the Laplace-Stieltjes transforms of the distributions of phase durations and the z-transforms of the number of jobs distributions at the beginning of each phase. We then show how to use these transforms to approximate the mean response time for small and large jobs. These three steps give the following summary theorem regarding $\mathbb{E}[T]$.

Theorem 2 (MSFQ Summary Theorem)**.** *The mean response time under MSFQ, $\mathbb{E}[T]$, depends on the first and second moments of H_i and N_i for all phases i. Hence, one can compute $\mathbb{E}[T]$ using the transforms $\widehat{N_i^S}(z)$ and $\widehat{N_i^L}(z)$ for all i using Lemmas 1-4.*

All of our analysis applies to the original MSF policy, by setting the Quickswap parameter ℓ to 0.

5.1. Throughput-optimality of MSFQ

We now prove that the MSFQ policy achieves optimal stability region, for any Quickswap threshold parameter ℓ. We start by lower-bounding its stability region:

Theorem 3. *In the 2-class one-or-all MSJ system, the Most Servers First with Quickswap policy is positive recurrent for all thresholds ℓ, $0 \leq \ell < k$, whenever $\frac{\lambda_1}{k\mu_1} + \frac{\lambda_k}{\mu_k} < 1$.*

Proof. We will use the Foster-Lyapunov theorem, with a carefully designed Lyapunov function.

Recall from Section 3.1 that we represent the system state as the five-tuple (n_1, n_k, u_1, u_k, z), where z is the phase, as defined in Section 4.2. Note that $u_1 \leq k$, $u_k \leq 1$, and at least one of u_1 and u_k must be 0.

Now, we can define our Lyapunov function $V(\cdot)$:

$$V(n_1, n_k, u_1, u_k, z) := \frac{n_1}{k\mu_1} + \frac{n_k}{\mu_k} + \mathbb{1}\{z = 2\}\frac{\epsilon n_k}{2\lambda_k} + \mathbb{1}\{z \notin \{1, 2\}\}(c^k - c^{k-u_1}), \tag{1}$$

where $\epsilon = 1 - \frac{\lambda_1}{k\mu_1} - \frac{\lambda_k}{\mu_k}$, and where $c = \max\big(2, \frac{\lambda_1}{(\ell-1)\mu_1 + 1}, \frac{1}{\mu_1}\big)$.

Intuitively, the two non-indicator terms ensure negative drift in phases 1 and 2, where all servers are busy. However, they do not suffice for the other phases, when some servers are idle. The $\mathbb{1}\{z = 2\}$ term reduces the negative drift in phase 2 slightly to build up a potential, which is used in the other phases by the $\mathbb{1}\{z \notin \{1, 2\}\}$ term to maintain negative drift in that phase.

We specifically use the continuous-time Foster-Lyapunov theorem [36]. We must demonstrate that this Lyapunov function has three properties, two of which are defined with reference to the drift $E[G \circ V(\cdot)]$, where G is the instantaneous generator operator of the system.

1. There exists a finite set of states B and a positive constant $\delta > 0$ such that for all states outside of B, $E[G \circ V(\cdot)] \leq -\delta$.
2. There exists a constant C such that for all states, $E[G \circ V(\cdot)] \leq C$.
3. There exists a lower bound D such that for all states, $V(\cdot) \geq D$.

Demonstrating Item 1 is the primary challenge. Item 3 holds with $D = 0$. Item 2 follows from the fact that V is linear in the two unbounded inputs n_1, n_k, which change by at most 1 upon any transition. Furthermore, from any state, the total transition rate is at most $\lambda + \max\{k\mu_1, \mu_k\}$. Hence, there exists an upper bound on drift from any system state.

It thus remains to prove Item 1, which we show holds with $\delta = \epsilon/2$ and the following finite exception set B, consisting of all states in phase 2 in which $n_1 = k$, so phase 2 has the potential to end, and where n_k is below a threshold, as well as the empty state:

$$B = \left\{(n_1, n_k, u_1, u_k, z) \mid \left(n_1 = k \,\&\, n_k \leq \frac{2\lambda_k(c^k - 1)}{\epsilon} \,\&\, z = 2\right) \text{ or } \left(n_1 = 0 \,\&\, n_k = 0\right) \right\}.$$

We specialize our argument based on the current phase of the system ($z = 1, 2, 3, 4$), and whether the system is a state on the border of switching phases.

We start with non-border states in phase 1. In this case, $u_k = 1, z = 1$. As a result, the drift of $V(\cdot)$ is:

$$E[G \circ V(\cdot)] = \frac{\lambda_1}{k\mu_1} + \frac{\lambda_k}{\mu_k} - u_k = \frac{\lambda_1}{k\mu_1} + \frac{\lambda_k}{\mu_k} - 1 = -\epsilon \leq -\delta,$$

where we use the fact that the drift of n_1 is λ_1 and of n_k is λ_k.

Next, consider states in phase 1 that are on the border of switching to phase 2. In the subsequent phase 2 state, $n_k = 0$, because the MSFQ policy only switches from serving large jobs to small jobs when there are no large jobs remaining. As a result, the $\mathbb{1}\{z = 2\}$ term in (1) is 0 when beginning phase 2, so Item 1 holds throughout phase 1.

Next, in phase 2, in states which are not on the border, we have $u_1 = k, z = 2$, by the definition of phase 2 in Section 4.2. As a result, the drift of $V(\cdot)$ is

$$E[G \circ V(\cdot)] = \frac{\lambda_1}{k\mu_1} - \frac{u_1}{k} + \frac{\lambda_k}{\mu_k} + \frac{\epsilon\lambda_k}{2\lambda_k} = -\epsilon + \frac{\epsilon}{2} = -\epsilon/2 = -\delta.$$

When the system is in phase 2 and is on the border of switching to phase 3, note that when a phase change occurs, the $\mathbb{1}\{z = 2\}$ term in (1) will change from $\frac{\epsilon n_k}{2\lambda_k}$ to 0, and the $\mathbb{1}\{z = 3\}$ term will change from 0 to $c^k - 1$. Recall that the exception set B includes all states in phase 2 on the border where n_k is below the threshold: $n_k \leq (c^k - 1)2\lambda_k/\epsilon$. For all phase 2 border states outside this set, the change in indicators from phase 2 to phase 3 results in a negative change in $V(\cdot)$, as desired. Thus, Item 1 holds throughout phase 2.

Finally, in the remaining phases 3 and 4, we use different arguments depending on the value of u_1. We start with states that do not transition directly to phase 1.

First, in the case where $u_1 = k$, the servers are fully occupied by small jobs. In this case, if $n_1 > k$, the $\mathbb{1}\{z = 2\}$ term in (1) does not change on the next arrival or completion, because $u_1 = k$ will remain true after the next event. Thus,

$$E[G \circ V(\cdot)] = \frac{\lambda_1}{k\mu_1} - \frac{u_1}{k} + \frac{\lambda_k}{\mu_k} = -\epsilon \leq -\delta.$$

If $u_1 = k$ and $n_1 = k$, the indicator term has a negative instantaneous drift: If a job completes, $n_1 = n - 1$, so the indicator term becomes $c^k - 1$, while under any other event, the indicator term remains c^k. Thus,

$$E[G \circ V(\cdot)] = \frac{\lambda_1}{k\mu_1} - \frac{u_1}{k} + \frac{\lambda_k}{\mu_k} - k\mu_1 = -\epsilon - k\mu_1 \leq -\delta.$$

In the case where $\ell < u_1 < k$, some servers are idle, and the system is in phase 3, so small jobs continue to enter service. In this case, we upper bound the drift of V as follows:

$$
\begin{aligned}
E[G \circ V(\cdot)] &= \frac{\lambda_1}{k\mu_1} - \frac{u_1}{k} + \frac{\lambda_k}{\mu_k} + \lambda_1(c^{k-u_1} - c^{k-(u_1+1)}) + u_1\mu_1(c^{k-u_1} - c^{k-(u_1-1)}) \\
&= \frac{\lambda_1}{k\mu_1} - \frac{u_1}{k} + \frac{\lambda_k}{\mu_k} + c^{k-u_1}(1-c)(\lambda_1 c^{-1} - u_1\mu_1) \le 1 + c^{k-u_1}(1-c)(\lambda_1 c^{-1} - (\ell-1)\mu_1)\,. \quad (2)
\end{aligned}
$$

Recalling that $c = \max\big(2, \frac{\lambda_1}{(\ell-1)\mu_1 + 1}\big)$, we substitute into (2). As a result,

$$E[G \circ V(\cdot)] \le 1 + c^{k-u_1}(1-c)(\lambda_1 c^{-1} - (\ell-1)\mu_1) \le 1 + 2^1(-1)(1) = -1 \le -\epsilon \le -\delta\,.$$

In the case where $0 < u_1 \le \ell$: the system is in phase 4, and small jobs are blocked from entering service. In this case, arriving jobs do not increase u_1, so the drift is much simpler:

$$
\begin{aligned}
E[G \circ V(\cdot)] &= \frac{\lambda_1}{k\mu_1} - \frac{u_1}{k} + \frac{\lambda_k}{\mu_k} + u_1\mu_1(c^{k-u_1} - c^{k-(u_1-1)}) \\
&\le 1 + u_1\mu_1(c^{k-u_1} - c^{k-(u_1-1)}) \le 1 + \mu_1(c - c^2) = 1 + \mu_1 c(1-c)\,.
\end{aligned}
$$

Recalling that $c \ge 2$ and that $c \ge \frac{1}{\mu}$, we find that $E[G \circ V] \le -1$, which completes this step.

The remaining case within phases 3 and 4 is the case where $u_1 = 0$. In this case, the system must be empty, because we would otherwise switch to phase 1. Recall that this case is in the finite set B of exceptional high-drift states, so it does not affect Item 1.

Having handled states in phases 3 and 4 that do not transition directly to phase 1, we now verify the drift condition for states where the system may transition from phase 3 or 4 to phase 1. When entering phase 1, $u_k = 1$, so $u_1 = 0$: All small jobs in service have been completed. As a result, the $\mathbb{1}\{z \notin \{1, 2\}\}$ term in (1) is 0, as desired.

With all states verified, Item 1 holds, so the Foster-Lyapunov theorem demonstrates stability. □

We now upper bound the optimal possible stability region in the one-or-all system:

Theorem 4. *In the 2-class one-or-all MSJ system, no scheduling policy is stable if* $\lambda_1/k\mu_1 + \lambda_k/\mu_k \ge 1$*,*

Proof. Define a job's *work* as the product of server need and mean service duration, scaled down by k. The system can never complete work at rate above 1, because there are k servers. Work arrives to the one-or-all system at a rate of $\lambda_1/k\mu_1 + \lambda_k/\mu_k$. If this rate is 1 or more, the system cannot be stable. This argument can be further formalized by comparing with a resource-pooled M/G/1. □

The throughput-optimality of the MSFQ policy now follows by combining Theorems 3 and 4:

Theorem 1 (MSFQ Throughput-Optimality)**.** *In the 2-class one-or-all MSJ system, the Most Servers First with Quickswap policy has optimal stability region for all thresholds* ℓ*,* $0 \le \ell < n$*.*

Remark 1. *Note that the same argument as in Theorem 3 applies to the Static Quickswap policy in the general-class system, allowing us to similarly lower-bound its stability region. A similar argument to Theorem 3 proves that the Static Quickswap policy is stable whenever* $\sum_j \frac{\lambda_j}{\lfloor k/j \rfloor \mu_j} < 1$*, where* j *ranges over all classes of server needs in the system.*

A similar workload-based argument to Theorem 4 proves that no policy can be stable if $\sum_j \frac{\lambda_j}{(k/j)\mu_j} \ge 1$*. If all* j *are perfect divisors of the total number of classes, and so* n/j *is always an integer, Static Quickswap thus has optimal stability region.*

5.2. Approximations in Response Time Analysis

To make the MSFQ system more tractable to response time analysis, we assume in Sections 5.3 and 5.4 that there is at least 1 large job in the system at the beginning of phase 1, and at least k small jobs in the system at the beginning of phase 2. This approximation ensures that all the phases are not skipped in a cycle, thus making the system easier to analyze despite being highly accurate as shown in Section 6. Intuitively, when the system load gets high, there would be at least 1 large job in the system at the beginning of phase 1 and at least k small jobs in the system at the beginning of phase 2 with high probability.

5.3. Response time analysis

Given that the MSFQ policy is throughput-optimal, we would like to analyze the mean response time of a stable MSFQ system. To analyze $\mathbb{E}[T]$, we analyze the conditional response time of a small job or a large job given the phase in which it arrives.

Specifically, in Lemma 2 we will analyze the conditional mean response time, $\mathbb{E}[T_1^L]$, of a large job that arrives in phase 1, and in Lemma 3 the conditional mean response time, $\mathbb{E}[T_{2,3,4}^L]$, of a large job that arrives in any of phases 2, 3, or 4. Similarly, we analyze the conditional mean response times $\mathbb{E}[T_{1,4}^S], \mathbb{E}[T_2^S]$, and $\mathbb{E}[T_3^S]$ of small jobs that arrive in either phases 1 or 4, phase 2, and phase 3, respectively. We also analyze m_i, the fraction of time the system spends in phase i.

We then characterize the mean response time under MSFQ, $\mathbb{E}[T]$, as follows:

$$\mathbb{E}[T] = \frac{\lambda_k}{\lambda}\left(\mathbb{E}[T_1^L]m_1 + \mathbb{E}[T_{2,3,4}^L](m_2+m_3+m_4)\right) + \frac{\lambda_1}{\lambda}\left(\mathbb{E}[T_{1,4}^S](m_1+m_4) + \mathbb{E}[T_2^S]m_2 + \mathbb{E}[T_3^S]m_3\right). \quad (3)$$

Our analysis of $\mathbb{E}[T]$ has three steps. First, in Lemma 1, we show that each m_i can be computed as a function of $\mathbb{E}[H_i]$, the mean duration of phase i. Second, in Lemmas 2 to 4, we derive explicit expressions for the conditional mean response times listed above, and show that these expressions depend on just the first and second moments of H_i and N_i for all i. Hence, to compute $\mathbb{E}[T]$ it suffices to find the transforms, $\widehat{N_i^S}(z)$, $\widehat{N_i^L}(z)$, and $\widetilde{H_i}(s)$ for all i. Third, we compute the necessary transforms in Section 5.4.

We begin by showing in Lemma 1 that the fraction of time m_i that the system spends in state i is proportional to the phase length $\mathbb{E}[H_i]$. If the phase cycles of the MSFQ policy formed a renewal process, one could apply a renewal reward argument [25] to prove Lemma 1. Unfortunately, the phase cycles do not form a renewal process. A long cycle accumulates many arrivals, leading to another long cycle. Hence, to relate m_i and $\mathbb{E}[H_i]$, we first prove a generalization of the renewal-reward theorem, Theorem 5, and then use this theorem to prove Lemma 1.

Theorem 5 (Extended renewal reward theorem)**.** *Consider a positive continuous-time stochastic process $\{I_t\}_{t\geq 0}$. Associated to the stochastic process is a reward function $R(t)$ specifying the total reward up to time t, and certain distinguished events which occur at times $\{K_n\}_{n\geq 1}$, whenever the process $\{I_t\}_{t\geq 0}$ enters a distinguished subset of its state space.*

We define a discrete-time stochastic process $\{(X_n, Y_n, R_n), n \geq 1\}$, where X_n is the interarrival time $K_{n+1} - K_n$ of the nth step, where $Y_n = I_{K_n}$ is the state at the beginning of step n, and $R_n = R(K_{n+1}) - R(K_n)$ is the reward of step n. We assume the mean duration of each step and the mean reward per step are finite: $\mathbb{E}[X] < \infty, E[R] < \infty$. We assume that the step durations and step rewards are identically and independently distributed given the state:

$$[X_i \mid Y_i = y] \sim [X_j \mid Y_j = y]\, \forall i,j,y, \qquad [R_i \mid Y_i = y] \sim [R_j \mid Y_j = y]\, \forall i,j,y.$$

Then with probability 1, the average reward accrued by time t in the underlying continuous-time process converges almost surely to the ratio of the mean reward per step and the mean duration of a step:

$$\frac{R(t)}{t} \to \frac{\mathbb{E}[R]}{\mathbb{E}[X]} \text{ as } t \to \infty.$$

Proof. See Appendix A. □

Lemma 1. *The fraction of time the MSFQ system spends in phase i is $m_i = \dfrac{\mathbb{E}[H_i]}{\sum_{i=1}^{4}\mathbb{E}[H_i]}$.*

Proof. Follows directly from applying Theorem 5, where we set the "distinguished events" to be the beginning of phase 1 and the reward function to be the time spent in phase i. □

Next, we will show that the conditional mean response times in (3) depend only on the first and second moments of H_i and N_i for each phase. We handle $\mathbb{E}[T_1^L]$ and $\mathbb{E}[T_2^S]$ in Lemma 2, $\mathbb{E}[T_{2,3,4}^L]$ and $\mathbb{E}[T_{1,4}^S]$ in Lemma 3, and $\mathbb{E}[T_3^S]$ in Lemma 4.

To analyze $\mathbb{E}[T_1^L]$ and $\mathbb{E}[T_2^S]$, we relate these terms to an $M/G/1$ *with Exceptional First Service* (EFS) system [9] as defined in Remark 2.

Remark 2 (EFS system)**.** *As stated in [9], an $M/G/1$* with Exceptional First Service *(EFS) system serves two different classes of jobs. Normally, job sizes are drawn i.i.d. according to some job size distribution S. However, the first job in each busy period experiences exceptional first service, and has a job size distributed as S'. Let $\mathbb{E}[W^{EFS}(\lambda, S, S')]$ be the mean work in an EFS system with arrival rate λ. From [9], we have*

$$E[W^{EFS}(\lambda, S, S')] = \frac{\lambda E[S^2]}{2(1-\lambda\mathbb{E}[S])} + \frac{\lambda(E[S'^2]-E[S^2])}{2(1-\lambda\mathbb{E}[S]+\lambda E[S'])}.$$

Let $p^{EFS}(\lambda, S, S')$ be the probability that a job arrives to an empty system and experiences exceptional service. We have that

$$p^{EFS}(\lambda, S, S') = \frac{1-\lambda\mathbb{E}[S]}{1-\lambda\mathbb{E}[S]+\lambda E[S']}.$$

Lemma 2. *The mean response time of large jobs arriving into phase 1, $\mathbb{E}[T_1^L]$, and the mean response time of small jobs arriving into phase 2, $\mathbb{E}[T_2^S]$, can be characterized as follows:*

$$\begin{cases} \mathbb{E}[T_1^L] &= \dfrac{\mathbb{E}[W^{EFS}(\lambda_k, S_k, N_1^L\Box S_k)]}{1-p^{EFS}(\lambda_k, S_k, N_1^L\Box S_k)} + \dfrac{1}{\mu_k} \\ E[T_2^S] &= \dfrac{\mathbb{E}[W^{EFS}(\lambda_1, S_1/k, (N_2^S-k+1)\Box(S_1/k))]}{1-p^{EFS}(\lambda_1, S_1/k, (N_2^S-k+1)\Box(S_1/k))} + \dfrac{1}{\mu_1} \end{cases},$$

where $X\Box Y$ is defined in Section 3.3. These expressions only depend on the first and second moments of S_1, S_k, N_1^L, and N_2^S.

Proof. We compare a tagged large job that arrives into phase 1 with a job in the EFS system to compute its response time. Specifically, we compare the mean work in the system a tagged large job sees on arrival during phase 1 to the mean work a tagged job that does not receive exceptional first service sees.

Consider the case where jobs arrive into an EFS system with rate λ_k, job sizes are i.i.d. exponentially distributed according to S_k except for the jobs that receive exceptional first service, whose job sizes are sampled i.i.d from $N_1^L\Box S_k$. In our MSFQ system, the first job that arrives in phase 1 needs to wait for N_1^L large jobs to complete. In the EFS system, the second job in a busy period also needs to wait for N_1^L large jobs' work to complete. The subsequent jobs in the busy period also see the same work in the queue as the subsequent jobs in phase 1 in our MSFQ system. In other words, a tagged job that arrives into the MSFQ system during phase 1 sees the same mean work compared to a job that does not receive exceptional service in the EFS system we consider. Hence, we can use the results in Remark 2 as if the exceptional job size distribution is $S' \sim N_1^L\Box Exp(\mu_k)$ and the non-exceptional distribution is $S \sim Exp(\mu_k)$. Then,

$$\begin{aligned} \mathbb{E}[T_1^L] &= \mathbb{E}[\text{mean work a tagged job sees}] + \frac{1}{\mu_k} \\ &= \mathbb{E}[W^{EFS}(\lambda_k, S_k, N_1^L\Box S_k) \mid \text{no exceptional service}] + \frac{1}{\mu_k} = \frac{\mathbb{E}[W^{EFS}(\lambda_k, S_k, N_1^L\Box S_k)]}{1-p^{EFS}(\lambda_k, S_k, N_1^L\Box S_k)} + \frac{1}{\mu_k}. \quad (4) \end{aligned}$$

The proof for $\mathbb{E}[T_2^S]$ follows a similar argument. We compare a tagged small job that arrives during phase 2 and an EFS system with an arrival rate λ_1, exceptional job size distribution $S' \sim (N_2^S - k + 1)\square(S_1/k)$, and non-exceptional distribution $S \sim S_1/k$. Then,

$$\mathbb{E}[T_2^S] = \frac{\mathbb{E}[W^{EFS}(\lambda_1, S_1/k, (N_2^S - k + 1)\square(S_1/k))]}{1 - p^{EFS}(\lambda_1, S_1/k, (N_2^S - k + 1)\square(S_1/k)))} + \frac{1}{\mu_1}. \tag{5}$$

Based on Remark 2 and (4), we can see that our $\mathbb{E}[T_1^L]$ formula depends on the first and second moments of S_k and $N_1^L\square S_k$. It is easy to see that the first and second moments of S_k is $\mathbb{E}[S_k] = 1/\mu_k$ and $\mathbb{E}[S_k^2] = 2/\mu_k^2$. We can compute $\mathbb{E}[N_1^L\square S_k] = \mathbb{E}[N_1^L]/\mu_k$ because N_1^L is independent from the job sizes. Lastly, to compute $\mathbb{E}[(N_1^L\square S_k)^2]$, we have

$$\mathbb{E}[(N_1^L\square S_k)^2] = \sum_{i=0}^{\infty}\mathbb{E}[(i\square S_k)^2]P\{N_1^L = i\} = \sum_{i=0}^{\infty}\frac{i + i^2}{\mu_k^2}P\{N_1^L = i\} = \frac{\mathbb{E}[(N_1^L)^2] + \mathbb{E}[N_1^L]}{\mu_k^2}.$$

Similarly, it suffices to compute first and second moments of S_1/k and $(N_2^S - k + 1)\square(S_1/k)$ to compute $\mathbb{E}[T_2^S]$. It is easy to see that $\mathbb{E}[S_1/k] = 1/(k\mu_1)$ and $\mathbb{E}[(S_1/k)^2] = 2/(k\mu_1)^2$. Likewise, we can compute $\mathbb{E}[(N_2^S - k + 1)\square(S_1/k)] = \mathbb{E}[N_2^S]/(k\mu_1)$. Lastly, to compute $\mathbb{E}[((N_2^S - k + 1)\square(S_1/k))^2]$, we have

$$\mathbb{E}[((N_2^S - k + 1)\square(S_1/k))^2] = \frac{\mathbb{E}[(N_2^S)^2] - (2k - 3)\mathbb{E}[N_2^S] + k^2 - 3k + 2}{k^2\mu_1^2}.$$

Therefore, we have shown that it suffices to compute the first and second moments of N_1^L and N_2^S to compute $\mathbb{E}[T_1^L]$ and $\mathbb{E}[T_2^S]$. $\square$

We now analyze the mean response time over all large jobs that arrive in any of phases 2, 3, or 4, and the mean response time over all small jobs that arrive in either phases 1 or 4.

Lemma 3. *The mean response time $\mathbb{E}[T_{2,3,4}^L]$ of large jobs which arrive during phases 2, 3, or 4, and the mean response time $\mathbb{E}[T_{1,4}^S]$ of small jobs which arrive during phases 1 or 4, are given by:*

$$\mathbb{E}[T_{2,3,4}^L] = \frac{(\lambda_k/\mu_k + 1)\mathbb{E}[(H_2 + H_3 + H_4)^2]}{2\mathbb{E}[H_2 + H_3 + H_4]} + \frac{1}{\mu_k}, \qquad \mathbb{E}[T_{1,4}^S] = \frac{(\lambda_1/(k\mu_1) + 1)\mathbb{E}[(H_4 + H_1)^2]}{2\mathbb{E}[H_4 + H_1]} + \frac{1}{\mu_1}.$$

Proof. Consider a tagged large job that arrives into the system in any of phases 2, 3, or 4. On arrival, this tagged job sees all the large job arrivals between the start of the most recent phase 2 and the current time in the system. Hence, the tagged job needs to wait until these jobs are completed in the upcoming phase 1 before it can begin receiving service. Let H_a^L be the *age* of the $H_2 + H_3 + H_4$ period, the elapsed time from the beginning of phase 2 an arrival during this period sees, and H_e^L be the *excess* of the $H_2 + H_3 + H_4$ period, the time until the end of phase 4 an arrival during this period sees. On average, the tagged large job sees $\lambda_k\mathbb{E}[H_a^L]$ large jobs in the system at the time it arrives into the system. Hence, the mean work the large job sees is $\frac{\lambda_k}{\mu_k}\mathbb{E}[H_a^L]$.

The response time of this tagged large job is composed of 3 elements: the time before any large job receives service, the mean work the tagged job sees in the system, and the time to serve itself. Hence,

$$\mathbb{E}[T_{2,3,4}^L] = \mathbb{E}[H_e^L] + \frac{\lambda_k}{\mu_k}\mathbb{E}[H_a^L] + \mathbb{E}[S_k]. \tag{6}$$

Combining Theorem 5, standard results on ages and excess [25], and (6), we have:

$$\mathbb{E}[T_{2,3,4}^L] = \frac{\mathbb{E}[(H_2 + H_3 + H_4)^2]}{2\mathbb{E}[H_2 + H_3 + H_4]} + \frac{\lambda_k}{\mu_k}\frac{(\lambda_k/\mu_k + 1)\mathbb{E}[(H_2 + H_3 + H_4)^2]}{2\mathbb{E}[H_2 + H_3 + H_4]} + \frac{1}{\mu_k}$$

$$= \frac{(\lambda_k/\mu_k + 1)\mathbb{E}[(H_2 + H_3 + H_4)^2]}{2\mathbb{E}[H_2 + H_3 + H_4]} + \frac{1}{\mu_k}.$$

Similarly, consider a tagged job that arrives into the system in either phase 4 or phase 1. On arrival, this tagged job sees all of the small job arrivals between the start of the most recent phase 4 and the current moment. let H_a^S and H_e^S be the age and excess of this $H_4 + H_1$ period. The tagged small job that arrives during phase 4 or phase 1 also needs to wait for an excess and $\frac{\lambda_1}{k\mu_1}\mathbb{E}[H_a^S]$ of work before receiving service. Its response time is composed of the excess duration, the mean work it sees on arrival, and the time it takes to complete:

$$\mathbb{E}[T_{1,4}^S] = \mathbb{E}[H_e^S] + \frac{\lambda_1}{k\mu_1}\mathbb{E}[H_a^S] + \mathbb{E}[S_1] = \frac{\mathbb{E}[(H_4 + H_1)^2]}{2\mathbb{E}[H_4 + H_1]} + \frac{\lambda_1}{k\mu_1}\frac{\mathbb{E}[(H_4 + H_1)^2]}{2\mathbb{E}[H_4 + H_1]} + \frac{1}{\mu_1}$$
$$= \frac{(\lambda_1/(k\mu_1) + 1)\mathbb{E}[(H_4 + H_1)^2]}{2\mathbb{E}[H_4 + H_1]} + \frac{1}{\mu_1}. \qquad \square$$

Therefore, we have shown that it suffices to compute the first and second moments of H_i in order to compute $\mathbb{E}[T_{2,3,4}^L]$ and $\mathbb{E}[T_{4,1}^S]$.

We finish by analyzing the mean response $\mathbb{E}[T_3^S]$ of small jobs that arrive during phase 3.

Lemma 4. *The mean response time $\mathbb{E}[T_3^S]$ of small jobs that arrive during phase 3 is given by:*

$$\mathbb{E}[T_3^S] = \frac{\sum_{j=\ell+1}^{\infty} \frac{C_j}{\lambda_1 + \min(k,j)\mu_1} \frac{k + (j-k+1)^+}{k\mu_1}}{\sum_{j=\ell+1}^{\infty} \frac{C_j}{\lambda_1 + \min(k,j)\mu_1}},$$

where C_j is defined recursively as follows, for each positive integer j:

$$C_j = \begin{cases} \frac{\lambda_1 + (\ell+1)\mu_1}{(\ell+1)\mu_1}\mathbb{1}\{\ell + 1 \le k - 1\} & j = \ell + 1 \\ C_{j-1}\frac{\lambda_1(\lambda_1 + j\mu_1)}{j\mu_1(\lambda_1 + (j-1)\mu_1)} + \frac{\lambda_1 + j\mu_1}{j\mu_1}\mathbb{1}\{j \le k-1\} & \ell + 1 < j \le k \\ \frac{\lambda_1}{k\mu_1}C_k & j > k \end{cases}.$$

Proof. Consider the stochastic process $\{n_1(t)\}$ during phase 3. By the definition of MSFQ, $n_1 = k - 1$ at the beginning of phase 3 and $n_1 = \ell$ at the end of phase 3. The process $\{n_1(t)\}$ forms an absorbing Markov chain corresponding to an $M/M/k$ system with arrival rate λ_1, and job size distribution S_1. In order to characterize the mean response time of a small job that arrives in phase 3, we condition the arrival based on the state it sees in $\{n_1(t)\}$. Therefore, it suffices to compute C_j, the number of visits to state j in $\{n_1(t)\}$, to characterize the probability a small job arrives in state j.

We first consider the case where there are at least $k + 1$ small jobs in the system. When $j \ge k + 1$, an arrival from state $j - 1$ or j will accrue one visit to state j,

$$C_j = C_{j-1}\frac{\lambda_1}{\lambda_1 + k\mu_1} + C_j\frac{\lambda_1}{\lambda_1 + k\mu_1} = C_{j-1}\frac{\lambda_1}{k\mu_1}.$$

Therefore, for all $j \ge k+1$, we have $C_j = \frac{\lambda_1}{k\mu_1}C_k$. When $\ell + 2 \le j \le k$, an arrival from state $j - 1$ or j will accrue one visit to state j with different rates:

$$C_j = C_{j-1}\frac{\lambda_1}{\lambda_1 + (j-1)\mu_1} + C_j\frac{\lambda_1}{\lambda_1 + j\mu_1} + \mathbb{1}\{j \le k-1\} = C_{j-1}\frac{\lambda_1(\lambda_1 + j\mu_1)}{j\mu_1(\lambda_1 + (j-1)\mu_1)} + \frac{\lambda_1 + j\mu_1}{j\mu_1}\mathbb{1}\{j \le k-1\}.$$

Finally, we handle $C_{\ell+1}$ separately:

$$C_{\ell+1} = C_{\ell+1}\frac{\lambda_1}{\lambda_1 + (\ell+1)\mu_1} + \mathbb{1}\{\ell + 1 \le k - 1\} = \frac{\lambda_1 + (\ell+1)\mu_1}{(\ell+1)\mu_1}\mathbb{1}\{\ell + 1 \le k - 1\}.$$

To compute $\mathbb{E}[T_3^S]$, note that the response time of a small job that arrives during phase 3 depends only on the number of small jobs $n_1(t)$ when the small job arrives. A small job that sees j jobs on arrival has expected response time $\dfrac{k+(j-k+1)^+}{k\mu_1}$. As a result, we can compute $E[T_3^S]$ by conditioning on $n_1(t)$ seen on arrival. By PASTA, this is simply the time-average distribution of the number of jobs seen during phase 3, which is given by the pre-absorption average of C_i. Specifically,

$$\mathbb{E}[T_3^S] = \frac{\sum_{j=\ell+1}^{\infty} \mathbb{E}[\text{total time spent in state } j] \cdot \mathbb{E}[T_3^S \mid \text{the job arrives during state } j]}{\sum_{j=\ell+1}^{\infty} \mathbb{E}[\text{total time spent in state } j]}$$

$$= \frac{\sum_{j=\ell+1}^{\infty} \frac{C_j}{\lambda_1+\min(k,j)\mu_1} \frac{k+(j-k+1)^+}{k\mu_1}}{\sum_{j=\ell+1}^{\infty} \frac{C_j}{\lambda_1+\min(k,j)\mu_1}}. \qquad \square$$

We have now shown that to compute the $\mathbb{E}[T_i]$ terms and the m_i terms in (3) for each phase i, it suffices to compute the first and second moments of H_i and N_i for all phases i. To compute these first and second moments, it suffices in turn to compute the transforms of H_i and N_i for all i. We tackle these transforms in the following section.

5.4. Phase Duration Analysis

In this section, we compute the required transforms of H_i and N_i for all phases i in order to compute their first and second moments. Taken together with Lemmas 1-4, this completes the proof of Theorem 2.

In our analysis, it will be useful to refer to *busy periods* of the system when serving either small or large jobs. We define these busy periods as follows.

Remark 3 (Busy Periods)**.** *We consider a busy period started by a random amount of work W to be the time required for an M/G/1 system to empty when starting with W work in the system. We let B_W^L be the duration of this busy period in an M/G/1 where only large jobs arrive, with arrival rate λ_k. Similarly, let B_W^S be the duration of this busy period in an M/G/1 where only small jobs arrive, with arrival rate λ_1. Using standard queueing-theoretic techniques [25], we have*

$$\widetilde{B_W^L}(s) = \widetilde{W}(s + \lambda_k - \lambda_k \widetilde{B_{S_k}^L}(s)), \qquad \widetilde{B_W^S}(s) = \widetilde{W}(s + \lambda_1 - \lambda_1 \widetilde{B_{S_1}^S}(s)).$$

To begin, we observe that the duration of phase 1 is equal to a busy period started by the number of large jobs that arrive during the preceding phases 2-4. Similarly, the duration of phase 2 is a busy period started by the number of small jobs that arrive during the preceding phases 4 and 1. This insight allows us to analyze $\widetilde{H_1}(s)$ and $\widetilde{H_2}(s)$ in Lemma 5.

Lemma 5. *The transforms of the distributions of phase 1 and phase 2 durations are given by:*

$$\widetilde{H_1}(s) = \widehat{N_1^L}(\widetilde{B_{S_k}^L}(s)), \qquad \widetilde{H_2}(s) = \widehat{N_2^S}(\widetilde{B_{S_1}^S}(s))(\widetilde{B_{S_1}^S}(s))^{1-k}.$$

Proof. At the beginning of phase 1, the system has $N_1^L \square S_k$ amount of work in terms of large jobs. At the end of phase 1, the system empties all the large jobs in the system. Therefore, the length of phase 1 can be seen as a busy period for large jobs started by $N_1^L \square S_k$ amount of work.

$$\widetilde{H_1}(s) = \widetilde{B_{N_1^L \square S_k}^L}(s) = \widehat{N_1^L}(\widetilde{S_k}(s + \lambda_k - \lambda_k \widetilde{B_{S_k}^L}(s))) = \widehat{N_1^L}(\widetilde{B_{S_k}^L}(s)).$$

Here, we use the fact that $\widetilde{X \square Y}(s) = \widehat{X}(\widetilde{Y}(s))$ where X is either independent of Y or is a stopping time relative to the sequence of durations Y [25].

Similarly, the system has N_2^S small jobs at the beginning of phase 2 and will have $k-1$ small jobs at the end of phase 2. In this case, the system finishes $N_2^S - k + 1$ jobs over the course of phase 2. Therefore, the length of phase 2 can be seen as a busy period for small jobs started by $(N_2^S - k + 1) \square S_k$ amount of work.

$$\widetilde{H_2}(s) = \widetilde{B_{(N_2^S - k+1)\square S_k}^S}(s) = \widehat{N_2^S - k + 1}(\widetilde{B_{S_1}^S}(s)) = \widehat{N_2^S}(\widetilde{B_{S_1}^S}(s))(\widetilde{B_{S_1}^S}(s))^{1-k}. \qquad \square$$

Next, we compute the z-transforms of the number of jobs in the system at the start of each phase. We note that our response time analysis depends only on the moments of N_1^L and N_2^S, hence it suffices to compute z-transforms $\widehat{N_1^L}(z)$ and $\widehat{N_2^S}(z)$. We show how these transforms depend on the Laplace transforms of the phase durations in Lemma 6.

Lemma 6. *The z-transforms of the distributions of the number of large jobs at the beginning of phase 1 and the number of small jobs at the beginning of phase 2 are given by:*

$$\widehat{N_1^L}(z) = \widetilde{H_2}(\lambda_k(1-z))\widetilde{H_3}(\lambda_k(1-z))\widetilde{H_4}(\lambda_k(1-z))$$
$$\widehat{N_2^S}(z) = \widetilde{H_2}(\lambda_k(1-\beta(z)))\widetilde{H_3}(\lambda_k(1-\beta(z)))\widetilde{H_4}(\lambda_k(1-\beta(z)) + \lambda_1(1-z)),$$

where $\beta(z) = \widetilde{B_{S_k}^L}(\lambda_1(1-z))$.

Proof. The number of large jobs at the beginning of phase 1 can be seen as the number of arrivals accrued during a $H_2 + H_3 + H_4$ time period because the large jobs are emptied at the end of phase 1. Formally, we have $N_1^L \sim A_{H_2+H_3+H_4}^L$, where A_X^L denotes the number of large job arrivals in X seconds.

Similarly, the number of small jobs at the beginning of phase 2 can be seen as the number of arrivals accrued during phase 4 and phase 1. However, note that phases 4 and 1 are not independent, as they are positively correlated. Intuitively, a longer phase 4 will result in a longer phase 1 because of more large jobs arriving into the system. In this case, we use $H_{4,1}$ to denote the length of the joint phase 4 and phase 1 period. As a result, we have $N_2^S \sim A_{H_{4,1}}^S$. Note that, if we are only interested in the mean of this joint period, $\mathbb{E}[H_{4,1}] = \mathbb{E}[H_4] + \mathbb{E}[H_1]$ due to the linearity of expectations.

Next, we use the standard transform formulas for Poisson arrivals during a random interval: $\widehat{A_X^L}(z) = \widetilde{X}(\lambda_k(1-z)), \widehat{A_X^S}(z) = \widetilde{X}(\lambda_1(1-z))$ [25]. Plugging into N_1^L, we have

$$\widehat{N_1^L}(z) = \widehat{A_{H_2+H_3+H_4}^L}(z) = \widetilde{H_2 + H_3 + H_4}(\lambda_n(1-z)) = \widetilde{H_2}(\lambda_k(1-z))\widetilde{H_3}(\lambda_k(1-z))\widetilde{H_4}(\lambda_k(1-z)).$$

Plugging into N_1^S, we have

$$\begin{aligned}\widehat{N_2^S}(z) = \widehat{A_{H_{4,1}}^S}(z) = \widetilde{H_{4,1}}(\lambda_1(1-z)) &= \int_0^\infty \widetilde{H_{4,1}}(\lambda_1(1-z) \mid H_4 = x)P\{H_4 = x\}dx \\ &= \int_0^\infty \widehat{N_1^L}(\beta(z) \mid H_4 = x)e^{-\lambda_1(1-z)x}P\{H_4 = x\}dx \\ &= \widetilde{H_2}(\lambda_k(1-\beta(z)))\widetilde{H_3}(\lambda_k(1-\beta(z)))\widetilde{H_4}(\lambda_k(1-\beta(z)) + \lambda_1(1-z)).\end{aligned}$$

□

Compared to H_1 and H_2, the durations H_3 and H_4 are relatively straightforward to analyze. Specifically, the length of phases 3 and 4 depends only on the definition of the number of servers, k, and MSFQ threshold, ℓ. These phases are independent of the lengths of other prior phases. We compute $\widetilde{H_3}(s)$ and $\widetilde{H_4}(s)$ in Lemma 7 and Lemma 8, deferring the proofs to Appendix B:

Lemma 7. *The Laplace transform of the duration of phase 3 is given by:* $\widetilde{H_3}(s) = \prod_{j=\ell+1}^{n-1} \widetilde{H_{3,j}}(s)$, *where* $H_{3,j}$ *is the transit time from* j *small jobs in the system to* $j-1$ *small jobs in the system, with transform:*

$$\widetilde{H_{3,j}}(s) = \begin{cases} \dfrac{j\mu_1}{\lambda_1 + j\mu_1 + s - \lambda_1\widetilde{H_{3,j+1}}(s)} & j < k \\ \widetilde{B_{S_1}^S}(s) & j = k\,. \end{cases} \tag{7}$$

Lemma 8. *The Laplace transform of the duration of phase 4 is given by:* $\widetilde{H_4}(s) = \prod_{j=1}^{\ell} \dfrac{j\mu_1}{j\mu_1 + s}$.

Given Lemmas 5-8, we are now ready to prove the summary theorem, Theorem 2.

Theorem 2 (MSFQ Summary Theorem)**.** *The mean response time under MSFQ, $\mathbb{E}[T]$, depends on the first and second moments of H_i and N_i for all phases i. Hence, one can compute $\mathbb{E}[T]$ using the transforms $\widehat{N_i^S}(z)$ and $\widehat{N_i^L}(z)$ for all i using Lemmas 1-4.*

Proof of Theorem 2. Lemmas 2-4 show that the mean response time of an MSFQ policy, $\mathbb{E}[T]$, depends on the system parameters $(k, \lambda_1, \lambda_k, \mu_1, \mu_k)$ and the first and second moments of H_i and N_i for all phases i. Lemmas 5-8 show how to compute the transforms of H_i and N_i that suffice to compute these moments. Specifically, while the formulas in Lemma 5-8 are recursive, the necessary moments can be computed by differentiating these formulas and solving a system of X equations. Plugging these moments into (3) and Lemma 1 yields an approximation of $\mathbb{E}[T]$ as desired. A calculator that performs these computations and returns the desired approximation will be available online (removed for anonymity). □

6. Simulation Results

In Section 5, we derived an approximation of the mean response time under MSFQ in the one-or-all case. This raises two important questions. First, how does MSFQ compare to other non-preemptive scheduling policies in the one-or-all case? And second, how do these results generalize to workloads with additional classes of jobs? To address these questions, we now evaluate MSFQ and several policies from the literature, comparing the mean response time predicted by our theoretical results to simulations of various other policies. Specifically, we compare against MSF, against the First-Fit backfilling policy [20], and against the nonpreemptive Markovian Service Rate policy (nMSR) [12]. We will show that our approximations from Section 5 are highly accurate, and that MSFQ can outperform all competitor policies by orders of magnitude.

We begin by simulating policies in the one-or-all case in Section 6.2 to show that our response time analysis is accurate and that MSFQ is by far the best of the non-preemptive scheduling policies. We then provide two natural generalizations of MSFQ to workloads with more than two job classes. We show that these generalizations, Adaptive Quickswap and Static Quickswap, perform well under more general workloads using both synthetic traces (Section 6.3) and traces from the Google Borg cluster scheduler (Section 6.4).

For simulation results, we wrote a discrete event simulation framework specifically developed for MSJ systems. Our simulator implements a wide range of scheduling policies and can either generate synthetic workloads or use real-world traces. This framework will be made available on GitHub (omitted for anonymity).

6.1. Simulation Metrics

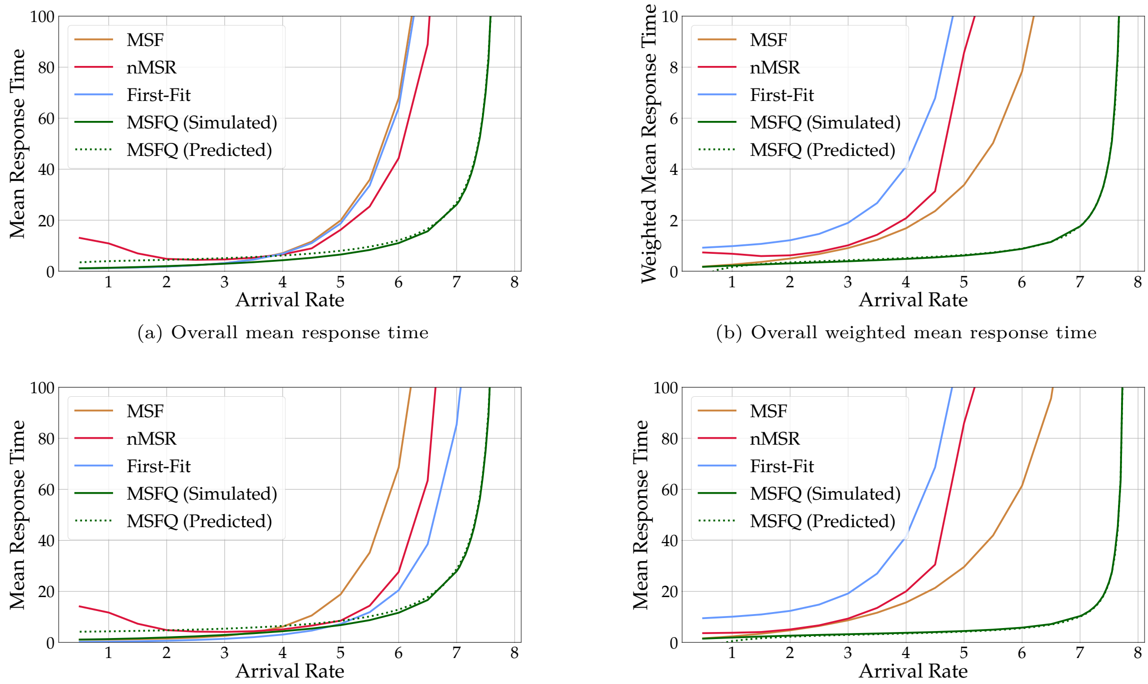
To evaluate our simulations, we will use a variety of response time metrics. For each class of jobs, j, we define $\mathbb{E}[T^{(j)}]$ to be the mean response time of class-j jobs. This allows us to examine the mean response time of each class separately to see how a policy balances the response times between job classes.

Assuming a workload consisting of m job classes, we can write the mean response time across all jobs as

$$\mathbb{E}[T] = \sum_{j=1}^{k} p_j \mathbb{E}[T^{(j)}].$$

We note, however, that as the number of job classes grows and the server needs and job sizes vary more between job classes, $\mathbb{E}[T]$ is not always the most meaningful metric. Specifically, in more complex cases, a large fraction of the system load can be composed of a small fraction of the jobs in the system. These jobs, which generally have large server needs and large mean job sizes, will be mostly ignored in the computation of $\mathbb{E}[T]$ because their p_i terms are small. For example, we find that in the Google Borg cluster scheduler, 85.8% of the system load is contributed by just 0.34% of the jobs in the workload. This aligns with the findings of the original Google Borg papers [38, 35].

Modern cluster schedulers cannot simply ignore such a large fraction of the system load. In practice, customers (whether internal or external to an organization) are billed for the number of server hours they consume. Because ignoring users who account for 85.8% of this revenue is not sustainable for data center



(a) Overall mean response time

(b) Overall weighted mean response time

(c) Mean response time of small jobs

(d) Mean response time of large jobs

Figure 2: Mean response time as a function of job arrival rate in a one-or-all MSJ system with $k = 32$, $p_1 = 0.9$, and $\mu_1 = \mu_k = 1$. MSFQ beats all other non-preemptive policies in terms of both mean response time and weighted mean response time. In particular, MSFQ can be two orders of magnitude better than MSF and nMSR with respect to both metrics.

operators, we propose a *weighted mean response time* metric that increases the weight of classes with high server needs and large average job sizes. Specifically, we define weighted mean response time as

$$E[T^w] = \frac{\sum_{j=1}^{k} j/\mu_j \cdot p_j E[T^{(j)}]}{\sum_{i=1}^{k} i/\mu_i \cdot p_i} = \sum_{j=1}^{k} \frac{\rho_j}{\rho} E[T^{(j)}],$$

where $\rho_j = \dfrac{j\lambda_j}{\mu_j}$ is the system load contributed by class-j jobs and $\rho = \sum_{j=1}^{k} \rho_j$ is the total system load. Under this definition, each job class's weight corresponds to the fraction of load it contributes to the system. Said another way, a class's weight corresponds to the server-hours used by the class (cost paid) as a fraction of the total server usage in the system (total system revenue).

6.2. Two job classes: one-or-all MSJ

We first evaluate the performance of MSFQ with $\ell = k - 1$ in the one-or-all setting analyzed in Section 5. We compare MSFQ to MSF, as well as the First-Fit and nMSR policies examined in the prior work. Our simulations consider a system with $k = 32$ servers, where 90% of job arrivals are small jobs and the mean job size is 1 for both large and small jobs. That is, $p_1 = 0.9$, $p_k = 0.1$, and $\mu_1 = \mu_k = 1$. These parameters reflect the common setting where 10% of the jobs (the large jobs) comprise about 80% of the load on the system. Fig. 2 shows the effect of varying the arrival rate, λ, on weighted and unweighted mean response time. We see that our analysis of the mean response time under MSFQ is highly accurate at a wide range of arrival rates. Furthermore, our MSFQ policy achieves the best weighted and unweighted mean response time in all cases, outperforming the competitor policies by two orders of magnitude when the arrival rate is high.

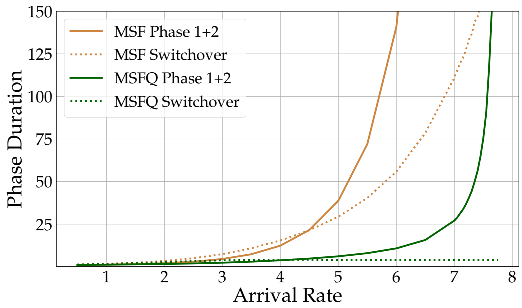


Figure 3: Service phase durations for the MSFQ policy evaluated in Figure 2.

We also measure the mean response time for each job class in Figs. 2c and 2d. These results confirm that MSFQ improves both classes' mean response time individually in order to improve overall mean response time, and especially improves the mean response time of large jobs. Note that the choice of weighted versus unweighted mean response time impacts the ordering of the competitor policies. The nMSR and First Fit policies benefit in the unweighted case by prioritizing small jobs, while MSF benefits in the weighted case by prioritizing large jobs. In both cases, none of the baseline policies can match MSFQ.

Our response time analysis in Section 5.3 confirmed that MSFQ improves mean response time by switching faster between service phases. To illustrate this point, we measured the phase durations of MSF and MSFQ during the above simulations. Recall that MSF is equivalent to an MSFQ policy with threshold $\ell = 0$, and the MSFQ policy in this case uses $\ell = k - 1$. Hence, both policies have full server utilization in phases 1 and 2, and use the remaining phases to switch the class of job in service. Fig. 3 compares the phase durations of these policies, showing that MSFQ has shorter switching phases, leading to much shorter durations of phases 1 and 2.

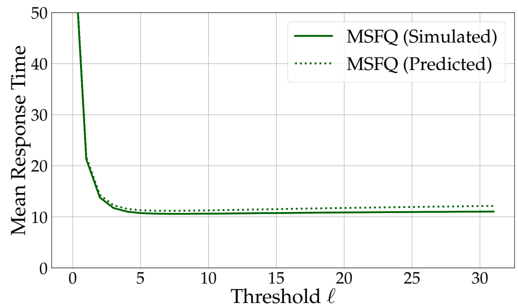


Figure 4: Impact of the threshold value, ℓ, on mean response time of the MSFQ policy evaluated in Figure 2.

We further illustrate this point in Fig. 4 by examining the effect of the threshold value, ℓ, on the mean response time of MSFQ. Using any threshold value larger than 0 has a dramatic benefit on mean response time by allowing faster switchover times and shorter phase durations. We also note that, while setting an arbitrarily large threshold could waste capacity by causing the system to switch too frequently, this effect is limited in practice. Hence, while our theoretical results can be used to select the optimal value of ℓ, a good heuristic appears to be to choose $\ell = k - 1$.

6.3. Generalizing to Additional Job Classes

While MSFQ has superb performance in the one-or-all case, it is not immediately clear how these results generalize to cases with additional job classes.

Hence, we now simulate the Static Quickswap and Adaptive Quickswap policies defined in Section 4. These policies generalize MSFQ to cases with many job classes, using the Quickswap mechanism to try and maintain the short phase durations of MSFQ. Given the added variability in server needs, we focus on weighted mean response time in this multiclass case. We consider a system with $k = 15$ servers and 4 classes: class-1, class-3, class-5, and class-15. Each class has a mean job size of 1, and we set $p_1 = 0.5$, $p_3 = 0.25$, $p_5 = 0.2$, and $p_{15} = 0.05$. By Remark 1, it is possible to stabilize the system when $\lambda < 5$. Note that we have chosen the server needs to divide k so that any one class of jobs can utilize all k servers.

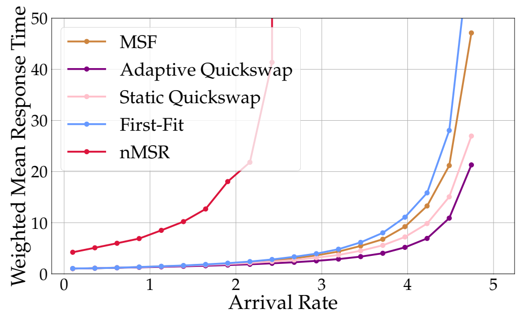


Figure 5: Weighted mean response time as a function of job arrival rate in a 4-class MSJ system with $k = 15$, $p_1 = 0.5$, $p_2 = 0.25$, $p_5 = 0.2$, and $p_{15} = 0.05$.

Figure 5 shows that Static and Adaptive Quickswap both provide an advantage over the competitor policies. Adaptive Quickswap performs the best in practice because it uses more complex switching logic to avoid having unused servers. Static Quickswap performs slightly worse than Adaptive Quickswap in all cases. However, Static Quickswap is guaranteed to be throughput-optimal by Remark 1, while Adaptive Quickswap has no such guarantee. Both policies outperform MSF and First-Fit in all cases.

6.4. Workloads Derived from Google Borg Traces

To further evaluate the Adaptive and Static Quickswap policies, we simulate these policies using workloads derived from the Google Borg cluster scheduler traces [35]. Specifically, we use the methodology of [39] to extract a workload with the same arrival rates, mean job sizes, and server needs as the Google Borg traces. Our workload consists of 26 job classes from Cell B of the 2019 Borg traces [35]. We set k based on the server need of the largest class, so $k = 2048$ in our experiments. The resulting stability region is defined by $\lambda < 4.94$.

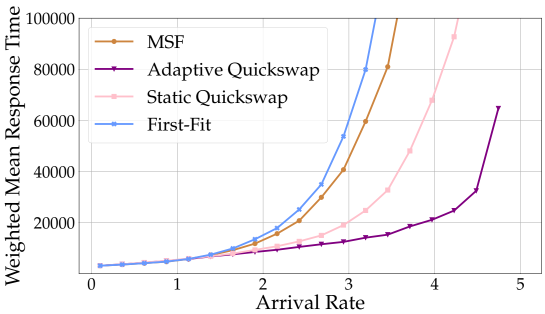


Figure 6: Weighted mean response time as a function of job arrival rate for MSJ systems serving a Google Borg workload. Here, $k = 2048$ and the workload is composed of 26 classes based on real-world trace data.

Figure 6 shows the benefit of using Static and Adaptive Quickswap instead of a competitor policy. While all policies remain stable, Adaptive and Static Quickswap are once again dominant, improving weighted mean response time by two orders of magnitude when the arrival rate is high. Note that, due to its poor performance in prior experiments, nMSR is omitted from Figure 6. These results generally match the synthetic workloads of Section 6.4, showing a significant benefit to using Adaptive Quickswap. However, even using Static Quickswap provides a 5x reduction in weighted mean response time compared to the next closest competitor, MSF.

7. Conclusion

This paper describes new, non-preemptive scheduling policies for multiserver jobs. While the non-preemptive MSF policy has been observed to remain stable at high loads, and while we prove it is throughput-optimal in the one-or-all case, it suffers from high mean response time because it switches service phases too slowly. We introduce the MSFQ class of policies, and its generalizations, Static Quickswap and Adaptive Quickswap. These Quickswap policies use a queue length threshold to decide when to switch phases, allowing for the design of policies that switch phases much faster than MSF. We prove that MSFQ is throughput-optimal in the one-or-all case, analyze the mean response time of MSFQ in the one-or-all case, and demonstrate the benefits of Quickswap policies in simulations based on traces from the Google Borg cluster scheduler.

Before this paper, there were two state-of-the-art choices for non-preemptive multiserver scheduling. There was MSF, which suffers from slow phase changes, and the class of MSR policies, which use a Markov chain to select schedules and allow phase changes at a faster rate. The drawback to MSR policies is that their scheduling decisions do not consider queue length information. As a result, an MSR policy may waste system capacity by reserving servers for jobs that are not in the system. Hence, before MSFQ, one had to choose between either high server utilization and slow phases switching, or low server utilization and fast phase switching. This paper shows that MSFQ policies get the best of both worlds, using queue length information to switch phases faster without wasting servers. Although MSFQ is more complex than either of the prior policies, we still provide an accurate analysis of its mean response time in the one-or-all case.

References

[1] Afanaseva, L., Bashtova, E., and Grishunina, S. Stability Analysis of a Multi-server Model with Simultaneous Service and a Regenerative Input Flow. *Methodology and Computing in Applied Probability 22*, 4 (2020), 1439–1455.

[2] Anggraito, A., Olliaro, D., Marin, A., and Ajmone Marsan, M. The Non-Saturated Multiserver Job Queuing Model with Two Job Classes: a Matrix Geometric Analysis. In *2024 MASCOTS* (2024), IEEE, pp. 1–8.

[3] Anggraito, A., Olliaro, D., Marin, A., and Ajmone Marsan, M. The multiserver job queuing model with two job classes and cox-2 service times. *Performance Evaluation 169* (2025), 102486.

[4] Artalejo, J. R., and Lopez-Herrero, M. Analysis of the busy period for the m/m/c queue: An algorithmic approach. *Journal of Applied Probability 38*, 1 (2001), 209–222.

[5] Beloglazov, A., and Buyya, R. Energy Efficient Allocation of Virtual Machines in Cloud Data Centers. In *2010 10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing* (2010), IEEE, pp. 577–578.

[6] Borst, S., and Boxma, O. Polling: past, present, and perspective. *Top 26* (2018), 335–369.

[7] Borst, S. C., and Boxma, O. J. Polling models with and without switchover times. *Operations Research 45*, 4 (1997), 536–543.

[8] Borst, S. C., and van der Mei, R. D. Waiting-time approximations for multiple-server polling systems. *Performance Evaluation 31*, 3-4 (1998), 163–182.

[9] Bose, S. K. *Advanced Queueing Networks*. Springer US, Boston, MA, 2002, ch. 4, pp. 98–101.

[10] Brill, P., and Green, L. Queues in which Customers receive Simultaneous Service from a Random Number of Servers: a System Point Approach. *Management Science 30*, 1 (1984), 51–68.

[11] Chen, W., Rao, J., and Zhou, X. Preemptive, Low Latency Datacenter Scheduling via Lightweight Virtualization. In *USENIX ATC* (2017), pp. 251–263.

[12] Chen, Z., Grosof, I., and Berg, B. Improving multiresource job scheduling with markovian service rate policies. *Proceedings of the ACM on Measurement and Analysis of Computing Systems 9*, 2 (2025), 1–36.

[13] Dean, J., and Ghemawat, S. Mapreduce: simplified data processing on large clusters. *Communications of the ACM 51*, 1 (2008), 107–113.

[14] Delimitrou, C., and Kozyrakis, C. Quasar: Resource-efficient and qos-aware cluster management. *ACM Sigplan Notices 49*, 4 (2014), 127–144.

[15] Filippopoulos, D., and Karatza, H. An M/M/2 Parallel System Model with Pure Space sharing among Rigid Jobs. *Mathematical Computer Modelling 45*, 5 (2007), 491–530.

[16] Foss, S. G., and Chernova, N. I. Dominance theorems and ergodic properties of polling systems. *Problems of Information Transmission 32*, 4 (1996), 342–364.

[17] Foss, S. G., and Chernova, N. I. On polling systems with infinitely many stations. *Siberian Mathematical Journal 37*, 4 (1996), 832–846.

[18] Fricker, C., and Jaibi, M. R. Monotonicity and stability of periodic polling models. *Queueing systems 15*, 1 (1994), 211–238.

[19] Georgiadis, L., Neely, M. J., and Tassiulas, L. Resource Allocation and Cross-Layer Control in Wireless Networks. *Foundations and Trends in Networking 1* (2006), 1–144.

[20] Grosof, I., and Harchol-Balter, M. Invited Paper: ServerFilling: A Better Approach to Packing Multiserver Jobs. In *Proceedings of the 5th Workshop on Advanced Tools, Programming Languages, and PLatforms for Implementing and Evaluating Algorithms for Distributed Systems* (2023), Association for Computing Machinery, pp. 1–5.

[21] Grosof, I., Harchol-Balter, M., and Scheller-Wolf, A. WCFS: a new framework for analyzing multiserver systems. *Queueing Systems 102*, 1 (2022), 143–174.

[22] Grosof, I., Harchol-Balter, M., and Scheller-Wolf, A. New Stability Results for Multiserver-job Models via Product-form Saturated Systems. *SIGMETRICS Performance Evaluation Review 51*, 2 (2023), 6–8.

[23] Grosof, I., Hong, Y., Harchol-Balter, M., and Scheller-Wolf, A. The RESET and MARC techniques, with application to multiserver-job analysis. *Performance Evaluation 162* (2023), 102378.

[24] Grosof, I., Scully, Z., Harchol-Balter, M., and Scheller-Wolf, A. Optimal Scheduling in the Multiserver-job Model under Heavy Traffic. *Proceedings of the ACM on Measurement and Analysis of Computing Systems 6*, 3 (2022), 1–32.

[25] Harchol-Balter, M. *Performance modeling and design of computer systems: queueing theory in action*. Cambridge University Press, 2013.

[26] Harchol-Balter, M. The Multiserver Job Queueing Model. *Queueing Systems: Theory and Applications 100*, 1-2 (2022), 201–203.

[27] Jette, Morris A.and Wickberg, T. Architecture of the Slurm Workload Manager. In *Job Scheduling Strategies for Parallel Processing* (2023), pp. 3–23.

[28] Maguluri, S. T., Srikant, R., and Ying, L. Stochastic Models of Load Balancing and Scheduling in Cloud Computing Clusters. In *IEEE INFOCOM 2012 - IEEE Conference on Computer Communications* (2012), IEEE, pp. 702–710.

[29] Morozov, E., and Rumyantsev, A. S. Stability Analysis of a MAP/M/s Cluster Model by Matrix-Analytic Method. In *Computer Performance Engineering (EPEW), Lecture Notes in Computer Science* (2016), Springer, pp. 63–76.

[30] Olliaro, D., Ajmone Marsan, M., Balsamo, S., and Marin, A. The saturated Multiserver Job Queuing Model with two classes of jobs: Exact and approximate results. *Performance Evaluation 162* (2023), 102370.

[31] Psychas, K., and Ghaderi, J. On non-preemptive vm scheduling in the cloud. *Proceedings of the ACM on Measurement and Analysis of Computing Systems 1*, 2 (2017), 1–29.

[32] Psychas, K., and Ghaderi, J. Randomized algorithms for scheduling multi-resource jobs in the cloud. *IEEE/ACM Transactions on Networking 26*, 5 (2018), 2202–2215.

[33] Rumyantsev, A., and Morozov, E. Stability Criterion of a Multiserver Model with Simultaneous Service. *Annals of Operational Research 252*, 1 (2017), 29–39.

[34] Speitkamp, B., and Bichler, M. A Mathematical Programming Approach for Server Consolidation Problems in Virtualized Data Centers. *IEEE Transactions Services Computing 3*, 4 (2010), 266–278.

[35] Tirmazi, M., Barker, A., Deng, N., Haque, M. E., Qin, Z. G., Hand, S., Harchol-Balter, M., and Wilkes, J. Borg: the next generation. In *Proceedings of the fifteenth European conference on computer systems* (2020), pp. 1–14.

[36] Tweedie, R. L. Sufficient conditions for regularity, recurrence and ergodicity of markov processes. *Mathematical Proceedings of the Cambridge Philosophical Society 78*, 1 (1975), 125–136.

[37] Vavilapalli, V. K., Murthy, A., Douglas, C., Agarwal, S., Konar, M., Evans, R., Graves, T., Lowe, J., Shah, H., Seth, S., Saha, B., Curino, C., O'Malley, O., Radia, S.and Reed, B., and Baldeschwieler, E. Apache Hadoop YARN: yet another Resource Negotiator. In *Proceedings of the 4th Annual Symposium on Cloud Computing* (2013), Association for Computing Machinery, pp. 1–16.

[38] Verma, A., Pedrosa, L., Korupolu, M. R., Oppenheimer, D., Tune, E., and Wilkes, J. Large-scale cluster management at Google with Borg. In *EuroSys* (2015).

[39] Yildiz, M., and Baiocchi, A. Data-driven workload generation based on google data center measurements. In *Proc. of HPSR* (2024), pp. 143–148.

Appendix A. Proof of Theorem 5

Theorem 5 (Extended renewal reward theorem)**.** *Consider a positive continuous-time stochastic process $\{I_t\}_{t\geq 0}$. Associated to the stochastic process is a reward function $R(t)$ specifying the total reward up to time t, and certain distinguished events which occur at times $\{K_n\}_{n\geq 1}$, whenever the process $\{I_t\}_{t\geq 0}$ enters a distinguished subset of its state space.*

We define a discrete-time stochastic process $\{(X_n, Y_n, R_n), n \geq 1\}$, where X_n is the interarrival time $K_{n+1} - K_n$ of the nth step, where $Y_n = I_{K_n}$ is the state at the beginning of step n, and $R_n = R(K_{n+1}) - R(K_n)$ is the reward of step n. We assume the mean duration of each step and the mean reward per step are finite: $\mathbb{E}[X] < \infty, E[R] < \infty$. We assume that the step durations and step rewards are identically and independently distributed given the state:

$$[X_i \mid Y_i = y] \sim [X_j \mid Y_j = y] \, \forall i, j, y, \qquad [R_i \mid Y_i = y] \sim [R_j \mid Y_j = y] \, \forall i, j, y.$$

Then with probability 1, the average reward accrued by time t in the underlying continuous-time process converges almost surely to the ratio of the mean reward per step and the mean duration of a step:

$$\frac{R(t)}{t} \to \frac{\mathbb{E}[R]}{\mathbb{E}[X]} \text{ as } t \to \infty.$$

Proof. First, note that the discrete-time stochastic process $\{Y_n\}$ forms a Markov chain, because it is defined by the moments when $\{I_t\}$ enters a distinguished subset of states. Note also that $\{Y_n\}$ is positive recurrent, because $\{I_t\}$ is positive recurrent.

Let 0 denote an arbitrary state in the state space of $\{Y_n\}$. We define a *zero-cycle* as a renewal cycle that begins at every distinguished time K_n such that $Y_n = 0$. Let A_m be the reward of the mth zero-cycle and let B_m be the length of the mth zero-cycle in time, and let A and B be the corresponding stationary random variables. Applying the conventional Renewal Reward theorem to the $\{I_t\}$ process,

$$\frac{R(t)}{t} \to \frac{\mathbb{E}[A]}{\mathbb{E}[B]}, \text{ as } t \to \infty. \tag{A.1}$$

To evaluate the reward per zero-cycle and the length of a zero-cycle, let Z_m be the number of steps in the mth zero-cycle. Z_m is the intervisit time of the $\{Y_n\}$ Markov chain to the 0 state, so Z_m is an i.i.d. random variable. Let Z denote the corresponding stationary random variable. Because $\{Y_n\}$ is positive recurrent, $E[Z] < \infty$. Let C_m be the number of steps up to the beginning of the mth zero-cycle.

Then we can write the reward and length of a given zero-cycle as follows:

$$A_m = \sum_{n=1}^{Z_m} R_{C_m+n}, \qquad B_m = \sum_{n=1}^{Z_m} X_{C_m+n}$$

We now apply the general form of Wald's equation, which states that

$$E[A_m] = E\left[\sum_{n=1}^{Z_m} E[R_{C_m+n}]\right] \qquad E[B_m] = E\left[\sum_{n=1}^{Z_m} E[X_{C_m+n}]\right]$$

By the conditional independence of R given Y, R_{C_m+n} only depends on the offset n, not the starting point C_m. Thus,

$$E[A] = E\left[\sum_{n=1}^{Z} E[R_n]\right], \qquad E[B] = E\left[\sum_{n=1}^{Z} E[X_n]\right],$$

By conditioning on Z, we can rewrite these expectations:

$$E[A] = \sum_{n=1}^{\infty} E[R_n \mathbb{1}\{Z \geq n\}] \qquad E[B] = \sum_{n=1}^{\infty} E[X_n \mathbb{1}\{Z \geq n\}] \tag{A.2}$$

But we can similarly write $E[R]$ based on the number of steps since the most recent beginning of a zero cycle. Specifically, we can apply renewal reward to the $\{Y_n\}$ process where the reward function is $R(t)$ in the nth step into a zero-cycle, and zero otherwise, and where the reward function is 1 in the nth step into a zero-cycle, and zero otherwise. By doing so we, find that

$$E[n\text{th step reward}] = \frac{E[R_n \mathbb{1}\{Z \geq n\}]}{E[Z]} \qquad E[n\text{th step length}] = \frac{E[X_n \mathbb{1}\{Z \geq n\}]}{E[Z]}$$

Summing over all n, we find that

$$E[R] = \sum_{n=1}^{\infty} \frac{E[R_n \mathbb{1}\{Z \geq n\}]}{E[Z]} \qquad E[X] = \sum_{n=1}^{\infty} \frac{E[X_n \mathbb{1}\{Z \geq n\}]}{E[Z]} \tag{A.3}$$

Combining (A.2) and (A.3), we find that

$$E[A] = E[R]E[Z], \qquad E[B] = E[X]E[Z].$$

Substituting into (A.1), we find that

$$\frac{R(t)}{t} \to \frac{\mathbb{E}[R]\mathbb{E}[Z]}{\mathbb{E}[X]\mathbb{E}[Z]} = \frac{\mathbb{E}[R]}{\mathbb{E}[X]}, \ \textit{as } t \to \infty.$$

□

Appendix B. Proofs of Lemma 7 and Lemma 8

Lemma 7. *The Laplace transform of the duration of phase 3 is given by:* $\widetilde{H_3}(s) = \prod_{j=\ell+1}^{n-1} \widetilde{H_{3,j}}(s)$, *where* $H_{3,j}$ *is the transit time from* j *small jobs in the system to* $j-1$ *small jobs in the system, with transform:*

$$\widetilde{H_{3,j}}(s) = \begin{cases} \dfrac{j\mu_1}{\lambda_1 + j\mu_1 + s - \lambda_1 \widetilde{H_{3,j+1}}(s)} & j < k \\ \widetilde{B_{S_1}^S}(s) & j = k\,. \end{cases} \tag{7}$$

Proof. To characterize $H_{3,j}$, we condition on the first event that happens during the $H_{3,j}$ period. When the first event is an arrival, the remainder of $H_{3,j}$ consists of a $H_{3,j+1}$ period, followed by another independent $H_{3,j}$ period. When the first event is a completion, the period ends. In particular, we have

$$H_{3,j} = \begin{cases} Exp(j\mu_1 + \lambda_1) + H_{3,j+1} + H_{3,j} & \text{next event is an arrival} \\ Exp(j\mu_1 + \lambda_1) & \text{next event is a departure} \end{cases}.$$

Note that $H_{3,k} \sim B^S_{S_1}$ as the time going from having k to $k-1$ small jobs is the busy period because the completion rate of small jobs here is $n\mu_1$. Then, the phase duration of phase 3 is the sum of these periods. Formally, $H_3 = \sum_{j=\ell+1}^{n-1} H_{3,j}$. Then, by standard transform techniques, Lemma 7 holds. □

Lemma 8. *The Laplace transform of the duration of phase 4 is given by:* $\widetilde{H_4}(s) = \prod_{j=1}^{\ell} \frac{j\mu_1}{j\mu_1 + s}$.

Proof. In phase 4, no further small job arrivals are allowed into service, and there are ℓ small jobs at the beginning of phase 4. Therefore, we can write H_4 as a sum of i.i.d. exponential distributions: $H_4 = \sum_{j=1}^{\ell} Exp(j\mu_1)$. Therefore, by standard transform techniques, $\widetilde{H_4}(s) = \prod_{j=1}^{\ell} \widetilde{Exp(j\mu_1)}(s) = \prod_{j=1}^{\ell} \frac{j\mu_1}{j\mu_1 + s}$. □