The Finite-Skip Method for Multiserver Analysis

ISAAC GROSOF, MOR HARCHOL-BALTER, ALAN SCHELLER-WOLF

Multiserver queueing systems are found at the core of a wide variety of practical systems. Unfortunately, existing tools for analyzing multiserver models have major limitations: Techniques for exact analysis often struggle with high-dimensional models, while techniques for deriving bounds are often too specialized to handle realistic system features, such as variable service rates of jobs. New techniques are needed to handle these complex, important, high-dimensional models.

In this paper we introduce the work-conserving finite-skip class of models. This class includes many important models, such as the heterogeneous M/G/k, the limited processor sharing policy for the M/G/1, the threshold parallelism model, and the multiserver-job model under a simple scheduling policy.

We prove upper and lower bounds on mean response time for any model in the work-conserving finite-skip class. Our bounds are separated by an additive constant, giving a strong characterization of mean response time at all loads. When specialized to each of the models above, these bounds represent the first bounds on mean response time known for each setting.

1 INTRODUCTION

Multiserver systems are pervasive in computing, including datacenters, supercomputing centers, multicore chips, and all server farm architectures. On the plus side, multiserver systems cost far less to build than purchasing a single super-fast sever (which may not even be available), and they also offer versatility in adjusting capacity. Unfortunately, the queueing-theoretic analysis of multiserver models is far more complicated than that of a single-server model. While there are a multitude of analytic results for single-server models, analytic results for multiserver models are limited and hard to come by.

In analyzing multiserver models, we are held back by the relatively small set of techniques available to analyze multiserver models, and the limitations of these techniques. Exact Markov-chain methods suffer from the curse of dimensionality as the system grows large [18, 29, 44]. Asymptotic methods such as fluid and diffusion limits are often only applicable at high load, many servers, or both [45, 47, 50, 51]. Lindley-type recursions can only be applied when the job completion process has a specific structure, renewing after every arrival [22, 25]. Multiserver SOAP requires that the scheduling policy is an index policy, and that that the servers are homogeneous [38, 39].

Prior analyses often require that all jobs are served at a constant service rate. For instance, the analysis by Levy and Yechiali [23] of the M/M/k system with server vacations uses an approach based on generator functions that depends crucially on the system having homogeneous servers. The equivalent analysis with heterogeneous servers is far more difficult, having only recently been solved for the M/M/2 [41]. Similarly, in the multiserver priority model of Van Harten and Sleptchenko [44], under homogeneous servers the dimensionality of the state space is dramatically reduced, allowing for an exact Markovian analysis. In their conclusion, the authors state that the heterogeneous problem would be intractable, due to an exponential blow-up in the dimension of the Markov chain. Heterogeneous settings typically violate the structure that Lindley-type recursions require, and Multiserver SOAP specifically assumes that the servers are homogeneous.

There are many important models where the assumption that all jobs are served at a constant rate does not hold and the existing techniques therefore fail. Some of the most important analytical models with variable service rate include:

Heterogeneous M/G/k: While the M/G/k is a natural model for many multiserver systems, traditional queueing analysis focuses on the homogeneous M/G/k, where every server runs at the same speed. In practice, such as in datacenter computing, it is much more common that the servers have different speeds, sometimes widely different. This comes up in part because the servers were bought in different years, but even more so because very different types of hardware are used within a datacenter. We refer to an M/G/k with different server speeds as the Heterogeneous M/G/k model.

Limited Processor Sharing: Processor-Sharing is the most common queueing abstraction for describing a time-sharing server, where multiple jobs are served simultaneously at a server, each job receiving an equal fraction of the server. In practice, however, the number of jobs that we time-share across is purposely limited, for many reasons such as fitting jobs in memory, limiting overheads and thrashing, and minimizing switching costs. Specifically, systems like databases are traditionally run with some Multi-Programming Level (MPL) k, where the system limits its processor-sharing to the k jobs in the system which arrived earliest, and all other jobs are held in a First-Come-First-Served (FCFS) queue. This is referred to in the literature as the Limited Processor-Sharing model. While Limited Processor-Sharing is not technically a multiserver model, the fact that it serves many jobs at once, and that the number of jobs being served simultaneously can range from 0 to k depending on the number of jobs in the system, lends it similar characteristics to multiserver models.

Threshold Parallelism: Today's jobs are often *moldable*, meaning that they can run on any number of servers. Specifically, the job can run on a single server for a processing time of x, or be parallelized across two servers for a processing time of $x/2$, or across three servers for a time of $x/3$, etc. Typically these moldable jobs have some threshold to their degree of parallelizability, e.g., the job can only be parallelized across at most p servers. Different jobs have different thresholds. In a multiserver system, it is common to serve arriving parallelizable jobs in FCFS order, where each job is allocated its threshold number of servers, until servers are no longer available. We refer to this as the Threshold Parallelism model.

Multiserver-Job Model: In contrast to moldable jobs, there are also jobs that require a *fixed* number of servers to run; these jobs are not flexible in their server need. This comes up a lot in cloud computing, where each arriving job can be viewed as a *pair*, (v, x), where v denotes the job's server need and x denotes its runtime requirement, i.e., all v servers will be occupied simultaneously for x time. In such cloud settings, v is always known, but x may be known or unknown. Because each job spans v servers, where typically $v > 1$, this is referred to as the Multiserver-Job model.

While all these models are common and practical, very few theoretical results exist on their performance. Specifically, none of these models can be formulated as index policies for multiserver SOAP or as a Lindley-type recursion, and all of the models are too multidimensional to apply exact Markov-chain methods, except in a purely numerical fashion. In particular, almost nothing is known about mean response time in these models, which is the focus of our paper. Here a job's *response time* is the time from when the job arrives until it completes.

1.1 New Tool: Work-Conserving Finite-Skip Models

We introduce a new technique for analyzing a wide variety of multiserver systems, including all of the systems detailed above. Specifically, we define the class of *finite-skip* models. Loosely speaking, in a finite-skip model, jobs receive service in near-FCFS order. In particular, a job can only receive service if it is in "the front" of the system, where the front has room for at most the n oldest (arrived earliest) jobs. The constant n is referred to as the maximum front size. We further focus on *work-conserving* finite-skip models. Here "work-conserving" refers to the fact that the system completes work at its maximum possible rate so long as the "front" is "full." All these terms will be defined precisely in Section 2. We develop a technique for analyzing any work-conserving finite-skip (WCFS) model. We refer to our technique as the *finite-skip technique*.

This class of models is general enough to capture all of the models listed above, but structured enough to admit analysis. In particular, the Heterogeneous M/G/k and Limited Processor Sharing models naturally fit in this class. The Threshold Parallelism model fits in this class under First-Come-First-Served (FCFS) service. In the Multiserver-Job model, FCFS service is not work-conserving, so our new technique does not apply. However, in Section 7, we introduce a simple and novel policy for the Multiserver-Job model called ServerFilling, which is a WCFS policy. For each of these models, our new technique allows us to derive one of the first results on mean response time.

1.2 Results

In Section 3 we prove upper and lower bounds on mean response time in any work-conserving finite-skip (WCFS) model. Moreover, our bounds are separated by an *additive* constant, making the bounds quite strong, especially under higher load. Specifically, in Corollary 3.8, we show that in any WCFS model M,

$$c_1 \leq E[T^M] - E[T^{M/G/1}] \leq c_2,$$

where $E[T^{M/G/1}]$ denotes the mean response time of an M/G/1 system with complete resource pooling, the same job size distribution and arrival rate as the model M, and FCFS service; and where c_1 and c_2 are explicit constants that depend on the specific model.

To get an intuitive sense of this result, imagine a utopian setting where the multiple servers could be pooled together so they function as a single super-fast FCFS server. Our result shows that mean response time in any real, complicated WCFS model exceeds mean response time in the utopian M/G/1 by at most an explicit additive constant. Our matching lower bound proves that the response time of any WCFS model must exactly match the response time of the utopian M/G/1, up to an additive constant.

Intuitively, our result provides two important insights:

- WCFS models are guaranteed to have good mean response times, where by "good" we mean comparable to a utopian M/G/1.
- To achieve even better mean response times, superior to that of the utopian M/G/1, one must look outside of the WCFS class of models.

1.3 Contributions

This paper makes the following contributions:

- In Section 2, we introduce the WCFS class of models.
- In Section 3, we bound the mean response time of any WCFS model.
- In Section 4, we apply our results to the Heterogeneous M/G/k system, thereby deriving the first bounds on mean response time in the Heterogeneous M/G/k. We also discuss prior work on simpler heterogeneous models.
- In Section 5, we apply our results to the Limited Processor Sharing system, thereby deriving the first bounds on mean response time in the system. We also compare our results to the prior heavy-traffic results.

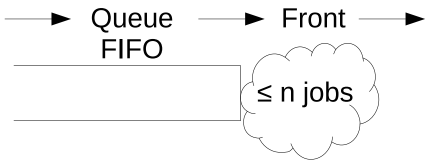


Fig. 1. Diagram of a Finite-Skip Model

- In Section 6, we discuss the FCFS policy for the Threshold Parallelism model, thereby deriving the first bounds on mean response time in the model. We also discuss prior work on related models.
- In Section 7, we introduce the ServerFilling policy for the multiserver-job model. We show that with this policy the resulting system is a WCFS model. We thereby derive mean response time bounds for the ServerFilling policy, the first such result for any multiserver-job policy.

2 WORK-CONSERVING FINITE-SKIP (WCFS) MODELS

We now define the class of finite-skip models. In a finite-skip model, as depicted in Fig. 1, we divide the system into two parts: the *queue* and the *front.* A job arrives to the queue, potentially waits for some period of time, and then is requested by the front. At that time, the job moves from the queue to the front in first-in-first-out order. Once a job moves to the front, it cannot leave the front until it finishes.

The front is defined as follows:

Definition 2.1. The **front** consists the following set of jobs:

(i) all jobs that are in service;
(ii) all jobs that have received service previously, but have not completed;
(iii) any jobs that arrived before any job that has received service; and
(iv) any jobs that have been requested but have not yet received service.

Based on the state of the front and the dynamics of the particular model, the front may request additional jobs from the queue, to make more jobs eligible for service. If the queue is not empty, the oldest job in the queue is immediately moved to the front. If the queue is empty and the front is requesting, the next job to arrive will immediately move to the front.

Definition 2.2. The front is **full** if it is not requesting additional jobs.

Note that if jobs remain in the queue, the front must be full.

The key requirement of a finite-skip model is that the number of jobs in the front is bounded by some finite constant n, which we refer to as the **maximum front size**. In particular, if there are n jobs at the front, the front must be full (the front could be full even if fewer than n jobs are present). As a result, a job can only skip ahead of at most $n-1$ other jobs, relative to FCFS service.

We normalize the total system capacity to 1, regardless of the number of servers in the system. We assume that every job has a **size** drawn i.i.d. from some general size distribution. Let S be a random variable denoting a job's size. The fraction of service that a job receives, integrated over its time in service, must equal the job's size for the job to complete. A job's size is in units of system capacity multiplied by time. For instance, if a job required half of the system capacity for 10 seconds, its size would be 5 system-seconds. We refer to the fraction of system capacity a job receives as its **service rate**.

In the general finite-skip model, any subset of the jobs at the front may receive service at any rate; the subset of jobs receiving service may be specified or constrained by the specific application.

See Sections 4 to 7 for examples. A job's remaining size decreases at a rate equal to its service rate. We assume that the set of jobs in service at the front, the rates of those jobs, and whether the front is full are solely determined by the state of the front, ignoring the state of the queue.

Whenever the full capacity of the system is in use, we call the system busy:

Definition 2.3. The system is **busy** whenever the total service rate is 1.

We define a finite-skip model to be a **work-conserving finite-skip** (WCFS) model if whenever the front is full, the system is also busy.

2.1 Examples of WCFS Models

2.1.1 Heterogeneous M/G/k. See Section 4 for a detailed definition of this model.

In the Heterogeneous M/G/k, the front consists of exactly the jobs in service. Jobs in service never leave service, so there are no other jobs in the system that have previously received service. The system is FCFS, so there are no jobs in the system that arrived before any of the jobs in service. The front is full if and only if all k servers are occupied. As a result, the model is finite-skip with maximum front size $n = k$.

We normalize the speeds of the servers so that the total service rate is 1 if all servers are occupied. As a result, the system is busy if and only if all of the servers are occupied. Therefore, the model is work-conserving, because the system is busy exactly when the front is full.

2.1.2 Limited Processor Sharing. See Section 5 for a detailed definition of this model.

In the Limited Processor Sharing model, the front consists of exactly the jobs in service. Once a job enters service, it never leaves service, and jobs enter service in FCFS order. The front is full if and only if k jobs are in service, where k is the Multi-Programming Level. As a result, the model is finite-skip with maximum front size $n = k$.

The system evenly splits its full service rate among the jobs in service, whenever one or more jobs are in service. Therefore, the system is busy whenever at least one job is present. In particular, whenever the front is full, the system is busy. Therefore, the model is work-conserving.

2.1.3 Threshold Parallelism. See Section 6 for a detailed definition of this model.

In the Threshold Parallelism model with FCFS service order, the front consists of exactly the jobs in service. Once a job enters service, it never leaves service, and jobs enter service in FCFS order. There are k servers.

The front is full if and only if jobs that request a total of k or more servers are in service, and hence all servers are occupied. If k jobs are in service, the front must be full, because each job requests at least one server. The model is therefore finite-skip with maximum front size $n \leq k$.

Because jobs parallelize perfectly, the total service rate is proportional to the number of occupied servers. In particular, if all of the servers are occupied, the system achieves its maximum total service rate, and hence is busy. The model is thus work-conserving.

2.1.4 Multiserver-Job Model. In Section 7, we give a detailed definition of this model. In particular, we define a novel policy for the Multiserver-Job model called ServerFilling. We define this policy because the simple FCFS policy for the Multiserver-Job setting is not work-conserving. In particular, the FCFS policy can reach a situation where a job in the front does not fit into service, leaving some servers idle, resulting in a failure of work-conservation.

The ServerFilling policy is a preemptive policy, so the front consists of jobs in service, preempted jobs, and incomplete jobs that arrived before some jobs currently in service.

As we describe in Section 7, the ServerFilling policy requests additional jobs until all k servers can be perfectly filled, taking into account the fact that jobs require a specific number of servers.

For the natural settings we focus on, we guarantee a finite upper bound on the number of jobs that must be requested this way, thereby bounding the maximum front size n and ensuring that the model is finite-skip.

In the Multiserver-Job model, the size of a job is the product of the fraction of servers the job requires and its service time requirement. As a result, the system's total service rate is proportional to the number of occupied servers, reaching a maximum of 1 when the front is full. The model therefore must be busy whenever it is full, and hence is work-conserving.

2.2 Further Technical Assumptions

We assume that jobs arrive to the queue according to a Poisson process with rate λ. Let load $\rho = \lambda E[S]$; we assume $\rho < 1$ to ensure stability. Throughout this paper, when we consider preemptive service policies, we assume preempt-resume service with no loss of work.

The WCFS class of models can handle both known-size and unknown-size settings. In general, we assume that a job evolves according to some Markovian state descriptor. In a known-size setting, the state descriptor might be a job's remaining size. In an unknown-size setting, the state descriptor might be a job's age (time in service).

In both cases, we define

$$\text{rem}_{\max} := \sup_s E[\text{Rem}(s)],$$

where $\text{Rem}(s)$ is the remaining size distribution from state s. Our only requirement is that $\text{rem}_{\max}$ is finite. For example, in the unknown-size setting, an arbitrary phase-type job size distribution has finite $\text{rem}_{\max}$, but a Pareto distribution does not.

We assume $\text{rem}_{\max} < \infty$ to ensure that whenever the front is not full, the amount of work in the system is bounded. We already have assumed that a finite number of jobs are present, but we also need to ensure that each job contributes a bounded amount of work. This assumption is used in Definition 3.1 and Theorem 3.3.

As currently defined, a WCFS system would be allowed to have service rate 0 if fewer than n jobs are in the front. This pathological behavior could lead to arbitrarily large response times as $\lambda \to 0$. To rule out this scenario, we assume that in all states where a job is present, there is some minimum service rate $b_{\min} > 0$. This assumption is used in Definition 3.6 and Theorem 3.7.

2.3 Notation

Let $W(t)$ be the total work in the system at time t. Work is defined as the sum of the remaining sizes of all jobs in the system. Let $W_Q(t)$ and $W_F(t)$ be the work in the queue and the work at the front, respectively. Let W, W_Q, and W_F denote the corresponding time-stationary random variables.

Let $N(t)$ be the number of jobs in the system at time t. Define $N_Q(t)$, $N_F(t)$, etc. accordingly.

Let $B(t)$ be the total service rate at time t, or equivalently the fraction of service capacity in use at time t. Let B denote the corresponding time-stationary random variable. The system is busy whenever $B(t) = 1$. Note that $E[B] = \lambda E[S] = \rho$.

Sometimes we will write the above quantities in terms of a state s rather than a time t, i.e. $N(s), B(s)$, etc. We use $\mathcal{S}$ to represent the set of all states.

Let T^M be a random variable denoting a job's time-stationary response time in model M. Where the model M is clear, we omit the superscript. Let T_Q and T_F be random variables denoting the time that jobs spend in the queue and in the front, respectively. Note that in a finite-skip model, $T = T_Q + T_F$.

3 RESULTS

Our main goal is to prove explicit bounds on mean response time $E[T]$ in a work-conserving finite-skip system. In order to do so, we bound mean queueing time $E[T_Q]$ in Theorem 3.3 and we bound mean front time $E[T_F]$ in Theorem 3.7.

The key property of a WCFS system for our analysis is that the system is always either busy ($B(t) = 1$) or has bounded work present ($W(t)$ is "small"). To quantify this, we define the maximum nonbusy work:

Definition 3.1. Let $w_{nonbusy}$ be the maximum expected work among all states which are not busy.

$$w_{nonbusy} := \max_{s \in \mathcal{S}} [E[W(s)] \mid B(s) < 1]$$

Note that $w_{nonbusy}$ is finite, because: (i) $\text{rem}_{max} < \infty$, (ii) there are at most n jobs in the front, and (iii) the queue is empty whenever the front is not full, which must occur when the system is not busy.

We also define the maximum nonfull work:

Definition 3.2. Let $w_{nonfull}$ be the maximum expected work among states which are not full.

Note that $w_{nonfull}$ is likewise finite, by the same reasoning. In general, $w_{nonfull} \geq w_{nonbusy}$, as busy states may not be full, but all full states are busy.

The quantities $w_{nonfull}$ and $w_{nonbusy}$ are typically easy to compute. However, note that $w_{nonfull}$ and $w_{nonbusy}$ are always at most $(n-1)\text{rem}_{\text{max}}$, which can be used as a bound in models where the specific values are hard to compute.

Now, we can state our bound on mean queueing time:

Theorem 3.3. *In a work-conserving finite skip model M, mean queueing time is bounded above and below:*

$$\frac{\lambda E[S^2]}{2(1-\rho)} - w_{nonfull} \leq E[T_Q^M] \leq \frac{\lambda E[S^2]}{2(1-\rho)} + w_{nonbusy}.$$

Proof. Let us start by writing time in queue T_Q in terms of the work in the system. To do so, we will consider an arbitrary tagged job j. When j arrives, let $W^A(j)$ be the amount of work j sees in the system. When j leaves the queue and enters the front, let $W_F^F(j)$ be the amount of work j sees in the front, other than j itself. Because jobs move from the queue to the front in FIFO order, all of the $W^A(j)$ work that was in the system when j arrived is either complete or in the front when j enters the front. Job j is in the queue for $T_Q(j)$ time. While j is in the queue, the front is full, so the system is busy and work completes at rate 1. As a result,

$$W^A(j) - W_F^F(j) = T_Q(j).$$

Note that just before j was admitted to the front, the system must not have been full. Therefore $W_F^F(j)$, the amount of work in the front just before j is admitted to the front, is at most $w_{nonfull}$. On the other hand, $W_F^F(j)$ is nonnegative. We can therefore bound $T_Q(j)$:

$$W^A(j) - w_{nonfull} \leq T_Q(j) \leq W^A(j)$$

Because j is an arbitrary job, and because Poisson arrivals see time averages,

$$W - w_{nonfull} \leq T_Q \leq W$$
$$E[W] - w_{nonfull} \leq E[T_Q] \leq E[W]. \tag{1}$$

Therefore, to bound $E[T_Q]$, we simply need to bound $E[W]$.

To do so, consider the stationary random variable W^2, the square of the stationary work in system. Because $W(t)$ is positive recurrent at zero, so too is $W^2(t)$, implying convergence to a stationary distribution.

W^2 evolves in two ways: continuous decrease as work is completed, and stochastic jumps as jobs arrive. Because W^2 is a stationary random variable, the expected rate of decrease and increase must be equal, as long as the expected rates are finite.

To calculate the expected rate of decrease, note that $\frac{d}{dt}W(t) = B(t)$. As a result, $\frac{d}{dt}W(t)^2 = 2W(t)B(t)$, and the expected rate of decrease of W^2 is $2E[WB]$.

To calculate the expected rate of increase, let t^- be the time just before a job arrives to the system. When the job arrives, $W(t^-)^2$ increases to $(W(t^-)+S)^2$, a change of $2W(t^-)S+S^2$. Note that $W(t^-)$ is distributed as W, by PASTA [17]. Note also that W and S are independent, because S is sampled i.i.d.. Arrivals occur at rate λ. As a result, the expected rate of increase is $2\lambda E[W]E[S] + \lambda E[S^2]$.

To prove that the rates of increase and decrease are equal, we must show that the rates are well defined, by showing that $E[W]$ is finite for any given value of $\rho < 1$.

Let us consider $E[W_F]$ and $E[W_Q]$ separately. Because there are at most n jobs in the front, and each has expected remaining size at most $\text{rem}_{\max}$, we know that $E[W_F] \le n\text{rem}_{\max}$. Therefore, we can focus on $E[W_Q]$.

Lemma 3.4. *For any $\rho < 1$, $E[W_Q]$ is finite.*

Proof. Consider the busy periods of the queue, the intervals when the queue is occupied. If the queue is occupied, the front is full, and so the system is busy, and completes work at rate 1. As a result, W can be bounded by the work in an variant of an M/G/1 during these busy periods. At the beginning of such a busy period, W is equal to $W_F(s)$ for some state of the front s. Therefore, W during a given busy period of the queue can be bounded by work during a busy period of an M/G/1 system with exceptional first service [46] distributed as $W_F(s)$ for some state of the front s. The largest mean work in the exceptional first service system over all states s therefore forms an upper bound on $E[W_Q]$. Therefore, Lemma 3.4 follows from the finiteness of mean work in the exceptional first-service system. □

Lemma 3.5. *For any $\rho < 1$ and for any state of the front s, mean work in an M/G/1 with exceptional first service distributed as $W_F(s)$ is finite.*

Proof. By prior results on the M/G/1 with exceptional first service [46], we know that this system's mean work is finite as long as $E[S^2]$ is finite, and $E[W_F(s)^2]$ is finite for all states s. These both follow from our assumption that $\text{rem}_{\max}$ is finite. We defer the details to Appendix A. □

Thus, $E[W]$ is finite, and the expected rates of increase and decrease of W^2 are equal. Setting the expected rates of increase and decrease equal, we find that

$$\begin{aligned}
2E[WB] &= 2\lambda E[W]E[S] + \lambda E[S^2] \\
E[WB] &= \lambda E[W]E[S] + \frac{\lambda}{2}E[S^2] = \rho E[W] + \frac{\lambda}{2}E[S^2] \\
E[W] - E[W(1-B)] &= \rho E[W] + \frac{\lambda}{2}E[S^2] \\
E[W](1-\rho) &= E[W(1-B)] + \frac{\lambda}{2}E[S^2] \\
E[W] &= \frac{E[W(1-B)]}{1-\rho} + \frac{\lambda E[S^2]}{2(1-\rho)}. \qquad (2)
\end{aligned}$$

Now, it merely remains to bound $E[W(1-B)]$. By the definition of $w_{nonbusy}$,

$$E[W \mid B < 1] \leq w_{nonbusy}.$$

Applying this to $W(1-B)$, we find that

$$E[W(1-B)] \leq w_{nonbusy}E[1-B].$$

Note that $E[B] = \rho$, because work arrives to the system at rate $\lambda E[S] = \rho$ and leaves at rate $E[B]$. Thus, $E[1-B] = 1-\rho$. Substituting this into (2), we find that

$$E[W] \leq w_{nonbusy} + \frac{\lambda E[S^2]}{2(1-\rho)}.$$

Noting that $W \geq 0$ and $B \leq 1$, we can drop the first term of (2) to show that

$$E[W] \geq \frac{\lambda E[S^2]}{2(1-\rho)}.$$

Substituting into our bounds on $E[T_Q]$ (1), we find that

$$\frac{\lambda E[S^2]}{2(1-\rho)} - w_{nonfull} \leq E[W] - w_{nonfull} \leq E[T_Q] \leq E[W] \leq \frac{\lambda E[S^2]}{2(1-\rho)} + w_{nonbusy}. \qquad \square$$

Now, we must bound mean front time $E[T_F]$. Note that we can often make use of a model's specific dynamics to more tightly bound $E[T_F]$. Here, however, we give a generic bound.

The key property of the system for this step is the fact that work is always performed if the system is not empty, even if fewer than n jobs are present. To quantify this property we define the service-to-number ratio.

Definition 3.6. Let the service-to-number ratio $R(s)$ in state s be defined as the ratio of the total service rate to the number of jobs in the front:

$$R(s) := \frac{B(s)}{N_F(s)}$$

We assume that $R(s)$ is lower bounded away from 0 over all states s where $N_F(s) > 0$. Specifically, let $r_{\min}$ be the infimum service-to-number ratio over all such states s. Note that $r_{\min} \geq b_{\min}/n > 0$ by assumption.

Theorem 3.7. *In a work-conserving finite-skip system M with infimum service-to-number ratio $r_{\min}$, mean front time is bounded above and below:*

$$E[S] \leq E[T_F^M] \leq \frac{E[S]}{r_{\min}}$$

Proof. In all states s,

$$r_{\min} \leq \frac{B(s)}{N_F(s)} \implies r_{\min}N_F(s) \leq B(s).$$

In expectation, the same must hold:

$$r_{\min}E[N_F] \leq E[B].$$

Note that $E[B] = \rho$ and that $E[N_F] = \lambda E[T_F]$ by Little's Law. Thus,

$$r_{\min}\lambda E[T_F] \leq \rho \implies E[T_F] \leq \frac{E[S]}{r_{\min}}.$$

Note also that the maximum possible service rate of a job is 1, so $E[T_F] \geq E[S]$. $\square$

Combining Theorems 3.3 and 3.7, we derive upper and lower bounds on mean response time:

Corollary 3.8.

$$-w_{nonfull} + E[S] \leq E[T^M] - \frac{\lambda E[S^2]}{2(1-\rho)} \leq w_{nonbusy} + \frac{E[S]}{r_{\min}}$$

For comparison, mean response time in a resource-pooled M/G/1 with FCFS service is

$$E[T^{M/G/1}] = \frac{\lambda E[S^2]}{2(1-\rho)} + E[S].$$

4 HETEROGENEOUS M/G/K

4.1 Motivation

Traditionally, multiserver queueing analysis has focused on homogeneous settings such as the standard M/G/k, where every server runs at the same speed. However, in many modern computing applications, different servers have dramatically different speeds. For example, modern datacenters are composed of servers with a wide variety of different types of hardware, leading to massive heterogeneity of performance [28, 30]. Similarly, the big.LITTLE architecture [8] brings heterogeneity to the mobile device setting. The heterogeneous M/G/k is a natural model for these computing systems. We specify this model in Section 4.2.

There has been little progress in analyzing the heterogeneous M/G/k model, but we outline the prior work that does exist in Section 4.3. In Section 4.4, we apply our results to the heterogeneous M/G/k setting, deriving the first bounds on mean response time for the setting.

4.2 Model

In the heterogeneous M/G/k, jobs arrive according to a Poisson process with rate λ, and job sizes are are sampled i.i.d. from some general size distribution. Let S be the random variable corresponding to a job's size, where size is measured in units of system capacity multiplied by time. As always, we focus on the bounded $\text{rem}_{\max}$ setting, as described in Section 2.2.

Let each server i have speed v_i. Let k be the number of servers. We order the servers so that $v_1 \leq v_2 \leq \ldots \leq v_k$, and we normalize the server speeds so that $\sum_i v_i = 1$. While a job is being served by server i, the job's remaining size decreases at a rate of v_i, completing when remaining size reaches 0. If job j has size s_j, and is running at server i, it will complete after $\frac{s_j}{v_i}$ time in service.

There are many possible server assignment policies for placing jobs at open servers, including Fastest Server First, Random Server Assignment, and Slowest Server First [1, 10]. One can consider both preemptive and nonpreemptive variants of these policies.

We do not focus on any particular assignment policy. We only assume that jobs are served in FCFS order, and that no job is left waiting while a server is idle. Our results apply to *all* such assignment policies; different policies may give rise to different additive terms in the bounds.

4.3 Prior Work

4.3.1 Heterogeneous M/M/k. Much of the previous work on multiserver models with heterogeneous service rates has focused on the much simpler M/M/k setting, where jobs are memoryless, so the only variation is between the server speeds [1, 10, 11, 24]. In this model, one can analyze the preemptive Fastest-Server-First policy to derive a natural lower bound on the mean response time of any server assignment policy. One can similarly analyze the preemptive Slowest-Server-First policy to derive an upper bound. These two policies each lead to a single-dimensional birth-death Markov chain, allowing for straightforward analysis [1]. One can think of our bounds as essentially extending these bounds for the M/M/k to the much more complex setting of the M/G/k.

4.3.2 Heterogeneous M/H_m/k. Van Harten and Sleptchenko [44] primarily study a *homogeneous* multiserver setting with hyperexponential job sizes. However, in their conclusion, they mention that their methods could be extended to a setting with heterogeneous servers, but at the cost of making their Markov chain grow exponentially. This exponential blowup seems inevitable when applying exact Markovian methods to a heterogeneous setting with differentiated jobs.

4.3.3 M/(M+G)/2 Model. Another intermediate model is the M/(M+G)/2 model of Boxma et al. [5]. In this model, jobs are not differentiated. Instead, the service time distribution is entirely dependent on the server. Server 1, the first server to be used, has an exponential service time distribution, while server 2 has a general service time distribution. Boxma et al. [5] make partial progress by deriving an implicit expression for the Laplace-Stieltjes transform of response time in this setting, which they are only able to make explicit when the general service time distribution has rational transform. Subsequent work has fully solved the M/(M+G)/2 model, giving exact stationary distributions under both FCFS service and related service disciplines [21, 35, 37].

Our results are not directly applicable to the M/(M+G)/2 setting, because the servers have different distributions of service time, not just different speeds. However, the slow progress on this two-server model illustrates the immense difficulty of applying prior methods to any but the simplest heterogeneous multiserver models. In contrast, our finite-skip technique handles both differentiated jobs and an arbitrary number of servers with no additional effort.

4.4 Results

The heterogeneous M/G/k is a work-conserving finite-skip model, and therefore can be analyzed using the finite-skip technique. The front, in this case, consists of the set of jobs in service. The front is full whenever all k servers are occupied, so maximum front size $n = k$. When all servers are occupied, the system has a total service rate of 1, in which case the system is busy, by Definition 2.3. The model has finite $\text{rem}_{\max}$ and nonzero minimum service rate whenever a job is present. Because all of the assumptions in Section 2 are satisfied, we can apply our main result, Corollary 3.8.

To apply Corollary 3.8 to this model, we must quantify the amounts of work $w_{nonfull}$ and $w_{nonbusy}$, and the service-to-number ratio $r_{\min}$.

In the heterogeneous M/G/k, the system is full if and only if it is busy. Both occur exactly when at least k jobs are in the system. As a result, $w_{nonfull} = w_{nonbusy}$. If the system is not full, then there are at most $k-1$ jobs present. Each job has remaining size at most $\text{rem}_{\max}$. As a result,

$$w_{nonfull} = w_{nonbusy} \leq (k-1)\text{rem}_{\max}. \tag{3}$$

As for the service-to-number ratio $r_{\min}$, note that every job in the front receives service at rate $\geq v_k$, the speed of the slowest server. As a result, $r_{\min} \geq v_k$. Under some server assignment policies, we can improve this bound; for instance, under preemptive Fastest-Server-First, $r_{\min} \geq 1/k$.

Combining (3) and our bound on r_{min} with Corollary 3.8 yields specific bounds on the mean response time $E[T^{Het\,M/G/k}]$ for the heterogeneous M/G/k setting:

$$-(k-1)\text{rem}_{\max} + E[S] \leq E[T^{Het\,M/G/k}] - \frac{\lambda E[S^2]}{2(1-\rho)} \leq (k-1)\text{rem}_{\max} + \frac{E[S]}{v_k}.$$

4.5 Visualization of Our Bounds

We have derived the first analytical bounds on mean response time in the heterogeneous M/G/k. Previous work was only able to analyze simpler settings, such as the M/M/k and the M/(M+G)/2.

In Fig. 2, we show a heterogeneous M/G/k with $k = 4$ servers, where the server speeds vary by a factor of 4. In blue, we show the exact mean response time of the preemptive Fastest-Server-First policy via simulation, because there is no known exact analysis of mean response time of any

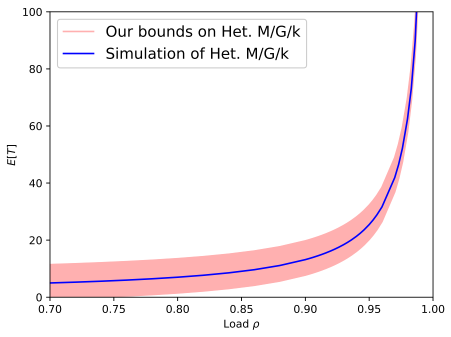


Fig. 2. Heterogeneous M/G/k with $k = 4$ servers running at speeds $[0.4, 0.3, 0.2, 0.1]$. Job size S is Hyperexponential: $Exp(2)$ w.p. 1/2, $Exp(2/3)$ w.p. 1/2. $E[S] = 1, E[S^2] = 2.5$. Server assignment is preemptive Fastest-Server-First. Bounds use $r_{\min} = 1/k$. 10^7 arrivals simulated.

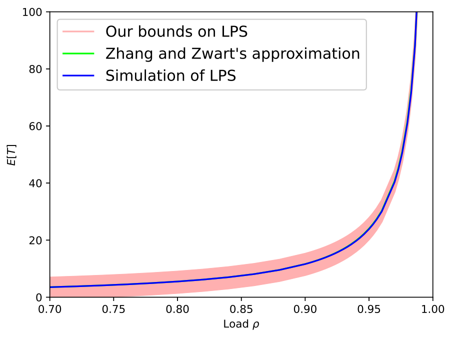


Fig. 3. Limited Processor Sharing (LPS) with Multi-Programming Level $k = 4$. Job size S is Hyperexponential: $Exp(2)$ w.p. 1/2, $Exp(2/3)$ w.p. 1/2. $E[S] = 1, E[S^2] = 2.5$. Zhang and Zwart's approximation is given in Section 5.3.1. 10^7 arrivals simulated.

server assignment policy. In the pink shaded region, we show our upper and lower bounds, which dramatically constrain the mean response time of the heterogeneous M/G/k.

5 LIMITED PROCESSOR SHARING

5.1 Motivation

The Processor Sharing policy for the M/G/1 is of great theoretical interest, and has been extensively studied [48]. However, in real systems, there is often an overhead to running too many jobs at once. A natural remedy for this is to set a Multi-Programming Level k, and only processor-share up to k jobs at a time, in FCFS order. This policy is known as Limited Processor Sharing (LPS), which we formally define in Section 5.2. While LPS has received significant study, analytical characterization of mean response time has proven elusive, as we discuss in Section 5.3. We give the first bounds on mean response time for the LPS policy in Section 5.4.

5.2 Model

Limited Processor Sharing (LPS) is a service policy for the M/G/1. Jobs arrive according to a Poisson process with rate λ, and job sizes are sampled i.i.d. from some general size distribution. Let S be the random variable corresponding to a job's size. As always, we focus on the bounded $\text{rem}_{\max}$ setting described in Section 2.2.

The LPS policy is defined by some Multi-Programming Level k. If at most k jobs are in the system, then the policy is equivalent to Processor Sharing, serving all jobs at an equal rate, with total service rate 1. When more than k jobs are present, the k oldest jobs in FCFS order are each served at rate $1/k$ and no other jobs receive service.

5.3 Prior Work

The Limited Processor Sharing policy has been studied by a wide variety of authors [16, 31, 42, 49–51], but none have analyzed mean response time under general load ρ.

5.3.1 Asymptotic Regimes. A series of papers by Zhang, Dai and Zwart [49–51] derive the strongest known results on Limited Processor Sharing in a variety of asymptotic regimes. In their three papers, the authors derive the measure-valued fluid limit [50], the diffusion limit [51] and a steady-state approximation [49], which they prove is accurate in the heavy traffic limit ($\rho \to 1$).

Of their results, the most comparable to our work is their steady-state approximation. When specialized to mean response time in the M/G/1, their approximation states that

$$E[T] \approx \frac{E[S]}{1-\rho}(1-\rho^k) + \frac{E[S^2]}{2E[S]}\frac{\rho^k}{1-\rho}$$

While they prove that this approximation is accurate in the heavy-traffic limit, they give no specific error bounds. However, it is empirically an excellent approximation at all load ρ [49]. Our results therefore complement the results of Zhang et al., by proving concrete error bounds, in contrast to their approximation.

5.3.2 State-dependent Server Speed. Gupta and Harchol-Balter [16] introduce a variant of the Limited Processor Sharing model, where the total server speed is a function of the number of jobs in service. They focus on a setting where server speed increases to a peak, and then slowly declines as more jobs enter service. They derive a two-moment approximation for mean response time, and use it to derive a heuristic policy for choosing the Multi-Programming Level (MPL). Notably, the optimal MPL for minimizing mean response time may be significantly larger than the service-rate-maximizing MPL, if job size variability is large and load is not too high.

Subsequently, Telek and Van Houdt [42] derive the Laplace-Stieltjes transform of response time in this model, under phase-time job sizes. Unfortunately, the transform takes the form of a complicated matrix equation. As a result, it is difficult to derive general insights from this result across general job size distributions. Instead, the authors numerically invert the Laplace transform for a handful of specific distributions to derive empirical insights.

The variable server-speed Limited Processor Sharing model is very nearly a WCFS model. It has a finite front, and a constant total service rate when the front is full. Let $\mu(k)$ denote this service rate. The only difficulty lies in the fact that $\mu(k)$ may not be the maximum service rate of the system. To handle this, we can adjust our definition of "busy" to refer to any time when the total service rate is at least $\mu(k)$. By doing so, we can derive very similar bounds on mean response time to those derived in Section 3. These bounds are the first to apply to general job size distributions, complementing the prior approximations [16] and results for phase-type distributions [42].

5.4 Results

The Limited Processor Sharing policy for the M/G/1 forms a work-conserving finite-skip model, and can therefore be analyzed using the finite-skip technique. The front, in this case, consists of the set of jobs in service. The front is full whenever all k servers are occupied, so the maximum front size $n = k$. The system achieves total service rate 1 whenever any jobs are present, in which case we describe the system is busy, by Definition 2.3. The model has finite $\text{rem}_{\max}$ and nonzero service rate whenever a job is present. Because all of the assumptions in Section 2 are satisfied, we can apply our main result, Corollary 3.8.

To apply Corollary 3.8 to this model, we must quantify the amounts of work $w_{nonfull}$ and $w_{nonbusy}$, and the service-to-number ratio $r_{\min}$. In this model, if the system is not full, then there are at most $k-1$ jobs present. As a result, $w_{nonfull} = (k-1)\text{rem}_{\max}$. If the system is not busy, then the system must be empty. As a result, $w_{nonbusy} = 0$. Because the total service rate is 1 whenever a job is present, the service-to-number ratio $r_{\min}$ is minimized when the front is full, making $r_{\min} = 1/k$.

Substituting these values into Corollary 3.8 yields specific bounds on mean response time $E[T^{LPS}]$ for the Limited Processor Sharing setting:

$$-(k-1)\text{rem}_{\max} + E[S] \le E[T^{LPS}] - \frac{\lambda E[S^2]}{2(1-\rho)} \le kE[S].$$

5.5 Visualization and Comparison to Prior Work

We have derived the first analytical bounds on mean response time under Limited Processor Sharing. Previous work by Zhang and Zwart [49] derived an approximation for mean response time based on heavy traffic analysis, but without any provable bounds. Their approximation is empirically accurate across all load ρ, however.

In Fig. 3, we show a LPS system with Multi-Programming Level $k = 4$ and hyperexponential job size distribution. In blue, we show the simulated mean response time. In green, we show the approximation of Zhang and Zwart [49]. Note that the two curves overlap, making the green curve hard to see. In the pink shaded region, we show our upper and lower bounds on mean response time. By provably constraining the mean response time of LPS for the first time, we complement Zhang and Zwart's excellent approximation.

6 THRESHOLD PARALLELISM

6.1 Motivation

In modern data centers, it is increasingly common for jobs to be parallelizable across different numbers of servers, where the level of parallelism is chosen by the scheduler [9, 32]. One model for such systems assumes that for each job, the user gives the ideal, maximum number of servers that the job can utilize. One natural policy in this setting is the FCFS policy, where the scheduler gives each job the maximum number of servers requested until servers run out, at which point the final job receives a partial allocation. We define this model and policy formally in Section 6.2. Unfortunately, no prior analysis of this FCFS policy exists in this model. In Section 6.3 we review the prior work that has been done in this model, and in Section 6.4 we present the first bounds on mean response time for the FCFS policy.

6.2 Model

In the Threshold Parallelism model, jobs arrive according to a Poisson process with rate λ, and job sizes are sampled i.i.d. from some general size distribution. Let S be the random variable corresponding to a job's inherent size, in units of server capacity multiplied by time. Let k be the number of servers. Each job j has a parallelism threshold $p_j \in [1, k]$ sampled i.i.d. from some general distribution. Let P be the corresponding random variable. Note that S and P can be correlated.

Job j may be parallelized across up to p_j servers, with linear speedup. If job j has size s_j and is served by 1 server, it receives $\frac{1}{k}$ of the server capacity and will complete after time ks_j. If job j is served by $q \le p_j$ servers, it will complete after $\frac{ks_j}{q}$ time in service, which is job j's realised runtime. As always, we focus on the bounded $\text{rem}_{\max}$ setting described in Section 2.2.

We focus on the FCFS service policy. Under this policy, jobs are placed into service in FCFS order until their parallelism thresholds sum to at least k, or no more jobs remain. Any job j which is not the newest job in service is served by p_j servers, the maximum possible number of servers. The newest job in service is served by the remaining servers. Thus a job may be served by an increasing number of servers over its time in service.

6.3 Prior Work

The Threshold Parallelism model can be thought of as a special case of the multiple speedup function model [2, 3], where the service rate of a job is a concave sublinear function of the number of servers it receives. However, results are only known in the multiple speedup function model in the setting where the job size distribution is exponential, and there are exactly two speedup functions, which corresponds to exactly two parallelism thresholds. In this setting, the optimal policy is shown to be one called "GREEDY*," which corresponds to the policy which preemptively prioritizes the jobs with smaller parallelism threshold. Even with these restrictions, no analytic bounds on response time are known. Our bounds apply to general job size distributions, and arbitrary parallelism thresholds.

6.3.1 Elastic and Inelastic Jobs. A special case of the Threshold Parallelism policy is the Elastic/Inelastic model of Berg et al. [4]. This model assumes that all jobs are either "inelastic," with parallelism threshold 1, or "elastic," with parallelism threshold k. They also assume that inelastic jobs have size distributed as $Exp(\mu_I)$, and elastic jobs have size distributed as $Exp(\mu_E)$, with sizes unknown to the scheduler. They focus on two preemptive-priority service policies for this setting: Inelastic First (IF) and Elastic First (EF). They prove that if $\mu_I \geq \mu_E$, then IF is the optimal service policy for minimizing mean response time, over all possible policies. They empirically show that if $\mu_I < \mu_E$, then EF often has lower mean response time than IF. They also perform an approximate response time analysis of EF and IF with a combination of the Busy-Period Transitions technique and Matrix-Analytic methods, to overcome the difficulties of a multidimensional Markov chain. This gives a numerical approximation that is empirically within 1% of simulation.

We prove the first analytic bounds for any service policy and any parallelism thresholds, subsuming the Elastic/Inelastic setting. Our bounds thus form a baseline for judging the performance of policies like IF and EF. Moreover, we handle arbitrary parallelism thresholds, not just 1 and k.

6.4 Results

The Threshold Parallelism model under FCFS scheduling is a work-conserving finite-skip model, and therefore can be analyzed using the finite-skip technique. The front, in this case, consists of the set of jobs in service. The front is full whenever all k servers are occupied, which occurs whenever the parallelism thresholds of the jobs in the system sum to at least k. This must occur when k jobs are in the system, so the maximum front size $n \leq k$. Service rate is proportional to the number of servers occupied, so when all servers are occupied the system is busy, by Definition 2.3. The model has finite $\text{rem}_{\max}$ and nonzero minimum service rate whenever a job is present. Because all of the assumptions in Section 2 are satisfied, we can apply our main result, Corollary 3.8.

To apply Corollary 3.8 to this model, we must quantify the amounts of work $w_{nonfull}$ and $w_{nonbusy}$, and the service-to-number ratio $r_{\min}$. In the Threshold Parallelism model, if there are $k-1$ jobs in the system, each with parallelism threshold 1, the system will be neither full nor busy. If k jobs are present, the system will be full and busy. As a result,

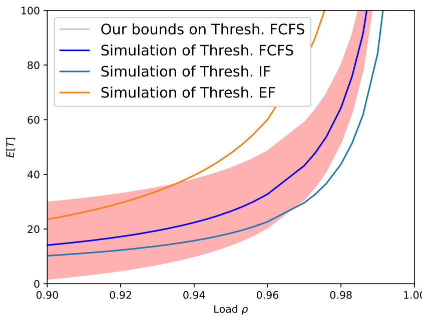
$$w_{nonfull} = w_{nonbusy} \leq (k-1)\text{rem}_{\max}. \tag{4}$$

Note that it may be possible to derive a better bound if jobs with parallelism threshold 1 do not achieve the maximum expected remaining size rem_{max}.

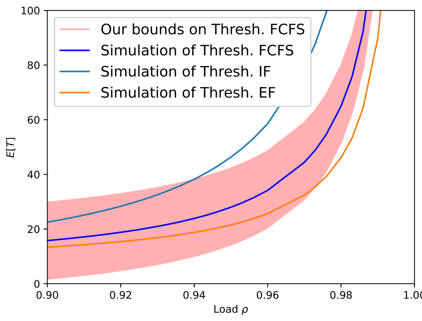
Every job in the front receives service at rate $\geq 1/k$, the rate of a single server, so $r_{\min} \geq 1/k$.

Plugging (4) and $r_{\min}$ into Corollary 3.8 yields specific bounds on the mean response time $E[T^{Thresh}]$ for the Threshold Parallelism setting:

$$-(k-1)\text{rem}_{\max} + E[S] \leq E[T^{Thresh}] - \frac{\lambda E[S^2]}{2(1-\rho)} \leq (k-1)\text{rem}_{\max} + kE[S].$$



(a) Joint distribution (P, S) is $(1, Exp(2))$ w.p. 1/2, $(8, Exp(2/3))$ w.p. 1/2.



(b) Joint distribution (P, S) is $(1, Exp(2/3))$ w.p. 1/2, $(8, Exp(2))$ w.p. 1/2.

Fig. 4. Threshold Parallelism system with $k = 8$ servers. Inelastic First (IF) prioritizes jobs with parallelism threshold 1, Elastic First (EF) prioritizes jobs with parallelism threshold 8. FCFS serves jobs in the order they arrive, on as many servers as possible. 10^7 arrivals each.

6.5 Visualization and Comparison to Prior Work

We have derived the first analytical bounds on mean response time for the FCFS policy in the Threshold Parallelism model.

To compare with prior work, we focus on the Elastic/Inelastic special case of the Threshold Parallelism model. Here, all jobs have parallelism threshold either 1 or k. Berg et al. [4] showed that in this model, if job sizes of each class of jobs are exponential, and if jobs that can only utilize one server (inelastic jobs) have smaller mean size, the optimal policy prioritizes jobs requiring one server (IF). Figure Fig. 4a depicts this situation. Here we see that our bounds on the FCFS policy cleanly separate it from the two prior policies. FCFS outperforms EF, and IF outperforms FCFS. In Figure Fig. 4b, the job size distributions have been reversed, so inelastic jobs have larger mean size. Here we see that EF now outperforms IF, and our bounds once again place FCFS firmly in the middle. Our bounds demonstrate that FCFS is a balanced choice, unlike IF and EF. The mean response time of FCFS is never too high, regardless of which kind of job is smaller.

7 MULTISERVER-JOB MODEL

7.1 Motivation

When we look at jobs in cloud computing systems [27] or in supercomputing systems [7, 12, 40], a job typically requires an exact number of servers for the entire time the job is in service. To illustrate, in Fig. 5 we show the distribution of the number of CPUs requested by the jobs in Google's recently published trace of its "Borg" computation cluster [15, 43]. The distribution is highly variable, with jobs requesting anywhere from 1 to 100,000 normalized CPUs[1].

The Multiserver-Job model is a natural model for these computing systems, which we specify in Section 7.2. We explore the extensive prior work on the Multiserver-Job model in Section 7.3. Unfortunately, the model is very difficult to analyze, so little is known about its mean response time, despite the extensive literature. We apply our results to give the first bounds on mean response time for any policy in the Multiserver-Job setting, in Section 7.4.

[1]The data was published in a scaled form [43]. We rescale the data so the smallest job in the trace uses one normalized CPU.

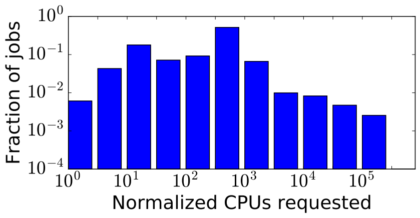


Fig. 5. The distribution of number of CPUs requested in Google's recently published Borg trace [43]. Number of CPUs is normalized to the size of the smallest request observed, not an absolute value.

7.2 Model

In the Multiserver-Job model, jobs arrive according to a Poisson process with rate λ. A job j has two service requirements: A number of servers v_j and an amount of time x_j. We assume that the pair (v_j, x_j) is sampled i.i.d. from some joint general distribution. Let (V, X) be the corresponding random variables. Note that V and X may be correlated.

If job j requires v_j servers, then job j can only be served if exactly v_j servers are allocated to it. We think of v_j as a quantity given by the user to the scheduler, while x_j may be known or unknown.

Let k be the number of servers. Recall from Section 2 that the size S of a job is the fraction of system capacity that a job uses when in service, multipled by the time the job spends in service. A given job j uses $\frac{v_j}{k}$ of the system capacity for time x_j, so it has size

$$s_j = \frac{v_j x_j}{k} \qquad S = \frac{VX}{k}.$$

As always, we focus on the bounded rem_{max} setting described in Section 2.2.

There are many possible scheduling policies for placing jobs at open servers, including FCFS, First Fit Backfilling, Most Servers First, and many more. One can consider both preemptive and nonpreemptive variants of these policies.

7.2.1 *Stability region.*

In the Multiserver-Job model, not all policies achieve the same stability region. For instance, the FCFS policy, while natural, often leaves servers empty due to head-of-the-blocking, shrinking its stability region [6, 36]. As a result, FCFS does not have optimal stability region, or equivalently it is not throughput optimal. We use these terms interchangeably.

A natural requirement for the system to be stable is that

$$\frac{\lambda E[VX]}{k} = \lambda E[S] = \rho < 1.$$

This requirement holds because ρ is the mean fraction of busy servers, in a stable system.

It can be shown that there exists a policy that achieves the maximum stability region $\rho < 1$ if for all $v \in \text{supp}(V)$, the support of V, v divides k. We denote the relation v divides k by $v|k$. We assume that $v|k$ for all $v \in \text{supp}(V)$, which we call the "divisible setting." A natural subset of the divisible setting, which we call the "exponential setting," holds when all of the server requirements v divide each other. Specifically, for each $v, v' \in \text{supp}(V)$ with $v \leq v'$, we assume that $v|v'$ in addition to $v|k$. The exponential assumption is often satisfied in the supercomputing literature, where it is common for all server requirements v to be powers of 2, and for k to be a power of 2 [20].

For a finite-skip policy to have optimal stability region $\rho < 1$ in the divisible setting, the policy must be work-conserving. No prior finite-skip policy for the Multiserver-Job model is also work-conserving. Therefore, we introduce a novel WCFS policy, which we call "ServerFilling."

7.2.2 ServerFilling. The ServerFilling policy ensures that if enough jobs are present in the system, all the servers will be occupied. The ServerFilling policy is only defined in the divisible setting. We define a variant of ServerFilling in the divisible setting, and another variant in the more specific exponential setting. In each setting, the ServerFilling policy serves jobs preemptively, based only on the number of servers required by the jobs in the system and the order the jobs arrived in. In particular, the policy ignores the ages of the jobs in the system.

7.2.3 ServerFilling in the Exponential Setting. In the exponential setting, ServerFilling starts by finding the minimal set of oldest jobs that collectively require at least k servers. In particular, let job 1 be the oldest job in the system, then job 2, etc. Let m be the minimal number of oldest jobs that collectively require at least k servers:

$$m := \inf_{m \le N} \left[m \,\middle|\, \sum_{j=1}^{m} v_j \ge k \right]$$

where N is the number of jobs in the system.

ServerFilling only serves jobs among the m oldest jobs in the system. If there is no such m, then all jobs in the system collectively require fewer than k servers, so ServerFilling serves them all. Assuming such an m exists, ServerFilling prioritizes jobs in order of server requirement v among the m oldest jobs. Ties are broken FCFS.

LEMMA 7.1. *If jobs requiring at least k servers are present, ServerFilling fills all k servers.*

Proof deferred to Appendix B.

We also defer the definition of ServerFilling in the divisible setting to Appendix B.

7.3 Prior Work

The Multiserver-Job model has been extensively studied, in both practical [7, 12, 40] and theoretical settings [6, 13, 19, 26, 27, 33, 34, 36]. Characterizing the stability region of policies in this model is already a challenging problem, and there were no bounds on mean response time for any policy in this model, prior to our bound on ServerFilling.

7.3.1 FCFS Scheduling. The most natural policy is FCFS scheduling, where the oldest jobs are placed into service until a job requires more servers than remain, at which point the queue is blocked until the job at the head of the queue can enter service. Therefore, the FCFS policy can leave a large number of servers idle even when many jobs are present. As a result, one can show that FCFS does not in general achieve an optimal stability region. Even worse, deriving the stability region of FCFS is major open problem, and has only been achieved in a few special cases [6, 36]. Policies with better theoretical guarantees, such as ServerFilling, are therefore desirable.

7.3.2 MaxWeight Scheduling. One natural throughput-optimal policy is the MaxWeight scheduling policy [27]. MaxWeight optimizes over all possible packings of jobs onto servers, selecting the packing that maximizes a value function based on the number of jobs in the system requiring each possible number of servers. While MaxWeight is throughput optimal, it has a major drawback: it is very computationally intensive, solving an NP-hard optimization problem whenever a job arrives or departs. For comparison, ServerFilling is also throughput-optimal in the divisible setting, but it is far computationally simpler, requiring approximately linear time as a function of k. Moreover, no bounds on mean response time are known on MaxWeight, due in part to its high complexity.

7.3.3 Nonpreemptive Scheduling. In certain practical settings such as supercomputing, a nonpreemptive scheduling policy is preferred. In such settings, a backfilling policy such as EASY backfilling or conservative backfilling is often used [7, 12, 40]. Backfilling policies start by scheduling jobs in FCFS order, until a job is reached that requires more servers than remain. At this point, newer jobs that require fewer servers are scheduled, but only insofar as this will not delay older jobs, based on user-provided service time upper bounds. While these policies are popular in practice, little is known about them theoretically, including their response time characteristics.

Finding any nonpreemptive throughput-optimal policy is already a challenging problem. Several such policies have been designed [13, 26, 34], typically by slowly shifting between different server configurations to alleviate overhead. Because such policies can have very large renewal times, many jobs can back up while the system is in a low-efficiency configuration. This can empirically lead to very high mean response times. However, no theoretical mean response time analysis exists for any policy in the Multiserver-Job setting. As a result, there is no good baseline policy to compare against novel policies, and it is thus impossible to tell whether a policy has low mean response time in a comparative or absolute sense. Our bounds on the mean response time of ServerFilling can therefore serve as such a baseline, albeit in the more permissive setting of preemptive scheduling.

7.4 Results

The Multiserver-Job model under the ServerFilling policy is a WCFS model in each of the divisible and exponential settings, and can therefore be analyzed using the finite-skip technique. We give our results for the exponential setting here, and defer the divisible setting to Appendix B.2.

The front consists of the m oldest jobs, where m is the minimal number of jobs for which the m oldest jobs require at least k servers (see Section 7.2.3). Without the mth oldest job, all k servers cannot be filled completely, and with it they are filled completely by Lemma 7.1, so the mth oldest job must be in service and must be the newest job in service. Thus, the front consists the m oldest jobs in the system. Note that m varies over time, which is allowed in a WCFS model. Because every job requires at least one server, $m \leq k$ at all times. As a result, the maximum front size $n \leq k$.

The front is full if and only if all k servers are occupied. The remaining size of a job j in service falls at rate $\frac{v_j}{k}$. As a result, whenever all k servers are occupied, the system is busy, by Definition 2.3. The model has finite $\text{rem}_{\max}$ and nonzero minimum service rate whenever a job is present. Because all of the assumptions in Section 2 are satisfied, we can apply our main result, Corollary 3.8.

To apply Corollary 3.8, we must quantify the amounts of work $w_{nonfull}$ and $w_{nonbusy}$, and the service-to-number ratio $r_{\min}$. The system is full if and only if it is busy, so $w_{nonfull} = w_{nonbusy}$. Specifically, $w_{nonfull} \leq (k-1)\text{rem}_{\max}$. The number of servers filled is at least the number of jobs present, so $r_{\min} \geq 1/k$.

Plugging the requisite values into Corollary 3.8 yields specific bounds on the mean response times $E[T^{MSJ\,Exp}]$ in the Multiserver-Job model in the Exponential setting:

$$-(k-1)\text{rem}_{\max} + E[S] \leq E[T^{MSJ\,Exp}] - \frac{\lambda E[S^2]}{2(1-\rho)} \leq (k-1)\text{rem}_{max} + kE[S].$$

These bounds can be improved by focusing on the random variables V and X, rather than focusing on the just the size S. We give such improved bounds on $E[T^{MSJ\,Exp}]$ in Appendix B.1.1.

7.5 Visualization and Comparison to Prior Work

We have introduced the ServerFilling policy, and derived analytical bounds on its mean response time. Our bounds are the first analytical bounds on mean response time under any scheduling policy for the Multiserver-Job setting.

To get an idea of the mean response time behavior of ServerFilling (SF) in the exponential setting, we compare it in simulation to several other policies for the Multiserver-Job system: FCFS [6, 36], MaxWeight [27], Least Servers First (LSF) [19], Most Servers First (MSF) and Preemptive First Fit

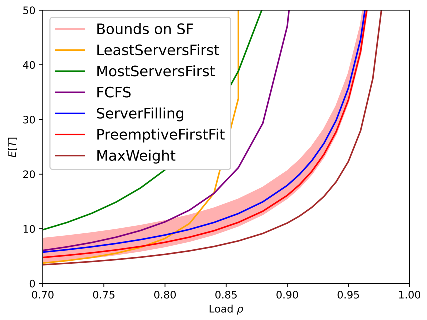


Fig. 6. Multiserver-Job system with $k = 4$ servers and a requirement distribution (V, X) of $(1, Exp(5))$ w.p. 1/3, $(2, Exp(2))$ w.p. 1/3, $(4, Exp(0.5))$ w.p. 1/3. $5 \cdot 10^6$ arrivals each.

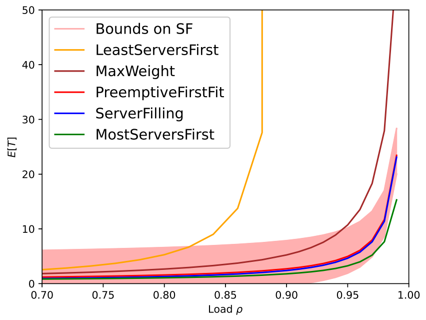


Fig. 7. Multiserver-Job system with $k = 4$ servers and a requirement distribution (V, X) of $(1, Exp(1))$ w.p. 4/7, $(2, Exp(8))$ w.p. 2/7, $(4, Exp(64))$ w.p. 1/7. $5 \cdot 10^6$ arrivals each.

backfilling (PFF). PFF serves jobs in FCFS order, but it skips over jobs which do not fit into service to serve additional jobs. All of these simulations are compared against our upper and lower bounds.

In Fig. 6, jobs requiring more servers have larger expected size. In this setting, FCFS and LSF have high mean response time, because neither has optimal stability region. MSF also has high mean response time, though it does empirically have optimal stability region. ServerFilling and PFF have good mean response times, both within our bounds on ServerFilling. Finally, MaxWeight has the best mean response time.

In Fig. 7, jobs requiring more servers have smaller expected size. In this setting, some of the policies are swapped around, while some perform similarly. FCFS (off the chart) and LSF still do not have optimal stability region. Now, MaxWeight has relatively high mean response time, while MSF has the best mean response time. ServerFilling and PFF again have good mean response times.

Empirically, PFF and ServerFilling have mean response times that are relatively robust to the job size distribution, while MSF and MaxWeight are highly affected. In the case of ServerFilling alone, we can prove this observation, using our bounds on mean response time.

8 CONCLUSION

We introduce the work-conserving finite-skip (WCFS) class of models, which contains several important multiserver queueing models, including the Heterogeneous M/G/k, the Limited Processor Sharing M/G/1, the Threshold Parallelism model, and the Multiserver-Job model. For each of these models, we derive the first analytic bounds on mean response time. Specifically, we characterize the mean response time behavior of any WCFS model up to an explicit additive constant.

One of the major insights of this paper is that characterizing mean response time primarily rests on understanding the behavior of the system when there are many jobs present. WCFS models are well-behaved when the "front is full," allowing us to bound mean response time, up to the uncertainty introduced by the model's behavior when the front is not full.

One direction for future work is applying the WCFS technique to new multiserver models. For instance, one could take an existing model, such as the Threshold Parallelism model, and add heterogeneous servers. Another direction is studying *non*-work-conserving finite-skip models. Can we prove anything of interest in a model when jobs are served in nearly-FCFS order, but not in a work-conserving fashion?

REFERENCES

[1] FSQ Alves, HC Yehia, LAC Pedrosa, FRB Cruz, and Laoucine Kerbache. Upper bounds on performance measures of heterogeneous m/m/c queues. *Mathematical Problems in Engineering*, 2011, 2011.

[2] Benjamin Berg and Mor Harchol-Balter. Optimal scheduling of parallel jobs with unknown service requirements. In *Handbook of Research on Methodologies and Applications of Supercomputing*, pages 18–40. IGI Global, 2021.

[3] Benjamin Berg, Jan-Pieter Dorsman, and Mor Harchol-Balter. Towards optimality in parallel scheduling. *Proc. ACM Meas. Anal. Comput. Syst.*, 1(2), December 2017. doi: 10.1145/3154499.

[4] Benjamin Berg, Mor Harchol-Balter, Benjamin Moseley, Weina Wang, and Justin Whitehouse. Optimal resource allocation for elastic and inelastic jobs. In *Proceedings of the 32nd ACM Symposium on Parallelism in Algorithms and Architectures*, SPAA '20, page 75–87, New York, NY, USA, 2020. Association for Computing Machinery. ISBN 9781450369350. doi: 10.1145/3350755.3400265.

[5] Onno J. Boxma, Qing Deng, and Albertus Petrus Zwart. Waiting-time asymptotics for the m/g/2 queue with heterogeneous servers. *Queueing Systems*, 40(1):5–31, 2002.

[6] Percy H. Brill and Linda Green. Queues in which customers receive simultaneous service from a random number of servers: A system point approach. *Management Science*, 30(1):51–68, 1984. doi: 10.1287/mnsc.30.1.51.

[7] Danilo Carastan-Santos, Raphael Y. De Camargo, Denis Trystram, and Salah Zrigui. One can only gain by replacing easy backfilling: A simple scheduling policies case study. In *2019 19th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID)*, pages 1–10, 2019. doi: 10.1109/CCGRID.2019.00010.

[8] Hyun-Duk Cho, Ph D Principal Engineer, Kisuk Chung, and Taehoon Kim. Benefits of the big. little architecture. *EETimes, Feb*, 2012.

[9] Christina Delimitrou and Christos Kozyrakis. Quasar: Resource-efficient and qos-aware cluster management. In *Proceedings of the 19th International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS '14, page 127–144, New York, NY, USA, 2014. Association for Computing Machinery. ISBN 9781450323055. doi: 10.1145/2541940.2541941.

[10] D. V. Efrosinin and V. V. Rykov. On performance characteristics for queueing systems with heterogeneous servers. *Automation and Remote Control*, 69(1):61–75, January 2008. ISSN 1608-3032. doi: 10.1134/S0005117908010074.

[11] Dmitry Efrosinin, Natalia Stepanova, Janos Sztrik, and Andreas Plank. Approximations in performance analysis of a controllable queueing system with heterogeneous servers. *Mathematics*, 8(10), 2020. ISSN 2227-7390. doi: 10.3390/math8101803.

[12] Dror G. Feitelson, Larry Rudolph, and Uwe Schwiegelshohn. Parallel job scheduling—a status report. In *Workshop on Job Scheduling Strategies for Parallel Processing*, pages 1–16. Springer, 2004.

[13] Javad Ghaderi. Randomized algorithms for scheduling VMs in the cloud. In *IEEE INFOCOM 2016 - The 35th Annual IEEE International Conference on Computer Communications*, pages 1–9, April 2016. doi: 10.1109/INFOCOM.2016.7524536.

[14] TH Gronwall. Some asymptotic expressions in the theory of numbers. *Transactions of the American Mathematical Society*, 14(1):113–122, 1913.

[15] Isaac Grosof, Mor Harchol-Balter, and Alan Scheller-Wolf. Stability for two-class multiserver-job systems. *arXiv preprint arXiv:2010.00631*, 2020.

[16] Varun Gupta and Mor Harchol-Balter. Self-adaptive admission control policies for resource-sharing systems. *SIGMETRICS Perform. Eval. Rev.*, 37(1):311–322, June 2009. ISSN 0163-5999. doi: 10.1145/2492101.1555385.

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

[18] Mor Harchol-Balter, Takayuki Osogami, Alan Scheller-Wolf, and Adam Wierman. Multi-server queueing systems with multiple priority classes. *Queueing Systems*, 51(3):331–360, 2005.

[19] Yige Hong and Weina Wang. Sharp zero-queueing bounds for multi-server jobs. 2021.

[20] James Patton Jones and Bill Nitzberg. Scheduling for parallel supercomputing: A historical perspective of achievable utilization. In Dror G. Feitelson and Larry Rudolph, editors, *Job Scheduling Strategies for Parallel Processing*, pages 1–16, Berlin, Heidelberg, 1999. Springer Berlin Heidelberg. ISBN 978-3-540-47954-3.

[21] Thaga Keaogile, A. Fatai Adewole, and Sivasamy Ramasamy. Geo (λ)/Geo (μ)+ G/2 queues with heterogeneous servers operating under fcfs queue discipline. *Am. J. Appl. Math. Stat*, 3(2):54–58, 2015.

[22] Wai Kin and Victor Chan. Generalized lindley-type recursive representations for multiserver tandem queues with blocking. *ACM Trans. Model. Comput. Simul.*, 20(4), November 2010. ISSN 1049-3301. doi: 10.1145/1842722.1842726.

[23] Yonatan Levy and Uri Yechiali. An m/m/s queue with servers' vacations. *INFOR: Information Systems and Operational Research*, 14(2):153–163, 1976. doi: 10.1080/03155986.1976.11731635.

[24] Woei Lin and P. Kumar. Optimal control of a queueing system with two heterogeneous servers. *IEEE Transactions on Automatic Control*, 29(8):696–703, 1984. doi: 10.1109/TAC.1984.1103637.

[25] D. V. Lindley. The theory of queues with a single server. *Mathematical Proceedings of the Cambridge Philosophical Society*, 48(2):277–289, 1952. doi: 10.1017/S0305004100027638.

[26] Siva Theja Maguluri and R. Srikant. Scheduling Jobs With Unknown Duration in Clouds. *IEEE/ACM Transactions on Networking*, 22(6):1938–1951, December 2014. ISSN 1558-2566. doi: 10.1109/TNET.2013.2288973. Conference Name: IEEE/ACM Transactions on Networking.

[27] Siva Theja Maguluri, Rayadurgam Srikant, and Lei Ying. Stochastic models of load balancing and scheduling in cloud computing clusters. In *2012 Proceedings IEEE Infocom*, pages 702–710. IEEE, 2012.

[28] Jason Mars, Lingjia Tang, and Robert Hundt. Heterogeneity in "homogeneous" warehouse-scale computers: A performance opportunity. *IEEE Computer Architecture Letters*, 10(2):29–32, 2011. doi: 10.1109/L-CA.2011.14.

[29] Rashid Mehmood and Jie A Lu. Computational markovian analysis of large systems. *Journal of Manufacturing Technology Management*, 2011.

[30] Ripal Nathuji, Canturk Isci, and Eugene Gorbatov. Exploiting platform heterogeneity for power efficient data centers. In *Fourth International Conference on Autonomic Computing (ICAC'07)*, pages 5–5, 2007. doi: 10.1109/ICAC.2007.16.

[31] Misja Nuyens and Wemke Van Der Weij. Monotonicity in the limited processor sharing queue. *resource*, 4:7, 2008.

[32] Yanghua Peng, Yixin Bao, Yangrui Chen, Chuan Wu, and Chuanxiong Guo. Optimus: An efficient dynamic resource scheduler for deep learning clusters. In *Proceedings of the Thirteenth EuroSys Conference*, EuroSys '18, New York, NY, USA, 2018. Association for Computing Machinery. ISBN 9781450355841. doi: 10.1145/3190508.3190517.

[33] Konstantinos Psychas and Javad Ghaderi. On Non-Preemptive VM Scheduling in the Cloud. *Proceedings of the ACM on Measurement and Analysis of Computing Systems*, 1(2):35:1–35:29, December 2017. doi: 10.1145/3154493.

[34] Konstantinos Psychas and Javad Ghaderi. Randomized algorithms for scheduling multi-resource jobs in the cloud. *IEEE/ACM Transactions on Networking*, 26(5):2202–2215, 2018. doi: 10.1109/TNET.2018.2863647.

[35] Sivasamy Ramasamy, Onkabetse A. Daman, and Sulaiman Sani. An M/G/2 queue where customers are served subject to a minimum violation of FCFS queue discipline. *European Journal of Operational Research*, 240(1):140–146, 2015. Publisher: Elsevier.

[36] Alexander Rumyantsev and Evsey Morozov. Stability criterion of a multiserver model with simultaneous service. *Annals of Operations Research*, 252(1):29–39, 2017.

[37] Sulaiman Sani and Onkabetse A. Daman. The M/G/2 Queue with Heterogeneous Servers Under a Controlled Service Discipline: Stationary Performance Analysis. *IAENG International Journal of Applied Mathematics*, 45(1), 2015.

[38] Ziv Scully, Mor Harchol-Balter, and Alan Scheller-Wolf. Soap: One clean analysis of all age-based scheduling policies. *Proc. ACM Meas. Anal. Comput. Syst.*, 2(1), April 2018. doi: 10.1145/3179419.

[39] Ziv Scully, Isaac Grosof, and Mor Harchol-Balter. Optimal multiserver scheduling with unknown job sizes in heavy traffic. *Performance Evaluation*, 145:102150, 2021. ISSN 0166-5316. doi: https://doi.org/10.1016/j.peva.2020.102150.

[40] S. Srinivasan, R. Kettimuthu, V. Subramani, and P. Sadayappan. Characterization of backfilling strategies for parallel job scheduling. In *Proceedings. International Conference on Parallel Processing Workshop*, pages 514–519, 2002. doi: 10.1109/ICPPW.2002.1039773.

[41] V Suvitha and P Kaviya. Markovian two server single vacation queue with balking and heterogeneous service rates. *J. Math. Comput. Sci.*, 11(5):5793–5801, 2021.

[42] Miklos Telek and Benny Van Houdt. Response time distribution of a class of limited processor sharing queues. *SIGMETRICS Perform. Eval. Rev.*, 45(3):143–155, March 2018. ISSN 0163-5999. doi: 10.1145/3199524.3199548.

[43] Muhammad Tirmazi, Adam Barker, Nan Deng, Md E. Haque, Zhijing Gene Qin, Steven Hand, Mor Harchol-Balter, and John Wilkes. Borg: The next generation. In *Proceedings of the Fifteenth European Conference on Computer Systems*, EuroSys '20, New York, NY, USA, 2020. Association for Computing Machinery. ISBN 9781450368827. doi: 10.1145/3342195.3387517.

[44] Aart Van Harten and Andrei Sleptchenko. On markovian multi-class, multi-server queueing. *Queueing systems*, 43(4): 307–328, 2003.

[45] Weina Wang, Qiaomin Xie, and Mor Harchol-Balter. Zero queueing for multi-server jobs. In *Abstract Proceedings of the 2021 ACM SIGMETRICS / International Conference on Measurement and Modeling of Computer Systems*, SIGMETRICS '21, page 13–14, New York, NY, USA, 2021. Association for Computing Machinery. ISBN 9781450380720. doi: 10.1145/3410220.3453924.

[46] Peter D. Welch. On a generalized m/g/1 queuing process in which the first customer of each busy period receives exceptional service. *Operations Research*, 12(5):736–752, 1964. doi: 10.1287/opre.12.5.736.

[47] Wentao Weng and Weina Wang. Achieving zero asymptotic queueing delay for parallel jobs. *Proc. ACM Meas. Anal. Comput. Syst.*, 4(3), November 2020. doi: 10.1145/3428327.

[48] SF Yashkov and AS Yashkova. Processor sharing: A survey of the mathematical theory. *Automation and Remote Control*, 68(9):1662–1731, 2007.

[49] Jiheng Zhang and Bert Zwart. Steady state approximations of limited processor sharing queues in heavy traffic. *Queueing Systems*, 60(3):227–246, 2008.

[50] Jiheng Zhang, J. G. Dai, and Bert Zwart. Law of large number limits of limited processor-sharing queues. *Mathematics of Operations Research*, 34(4):937–970, 2009. doi: 10.1287/moor.1090.0412.

[51] Jiheng Zhang, J. G. Dai, and Bert Zwart. Diffusion limits of limited processor sharing queues. *The Annals of Applied Probability*, 21(2):745 – 799, 2011. doi: 10.1214/10-AAP709.

A DETAILS OF THE PROOF OF THEOREM 3.3

LEMMA 3.5. *For any $\rho < 1$ and for any state of the front s, mean work in an M/G/1 with exceptional first service equal to $W_F(s)$ is finite.*

PROOF. By prior results on the M/G/1 with exceptional first service [46], we know that this system's mean work is finite as long as $E[S^2]$ is finite, and $E[W_F(s)^2]$ is finite for all states s. These both follow from our assumption that $\text{rem}_{\max}$ is finite.

To show that $E[S^2]$ is finite, note that $E[S^2] = 2E[S_e]E[S]$, where S_e is the equilibrium distribution over S [17, Chapter 23]. Because S_e is a mixture of remaining size distributions, $E[S_e] \le \text{rem}_{\max}$. As a result,

$$E[S^2] \le 2E[S]\text{rem}_{\max},$$

which is finite, as desired.

As for $E[W_F(s)^2]$, note that

$$W_F(s) \le \sum_{i=1}^{n} \text{Rem}(s_i)$$

where s_i is the state of the ith job in the front. Because $E[\text{Rem}(s_i)]$ is finite, we need only focus on terms of the form $E[\text{Rem}(s_i)^2]$ to prove that $E[W_F(s)]$ is finite. With essentially the same argument as for S^2, we can use an equilibrium distribution over $\text{Rem}(s_i)$ to prove that

$$E[\text{Rem}(s_i)^2] \le 2\text{rem}^2_{\max},$$

implying that $E[W_F(s)]$ finite for an arbitrary state of the front s, as desired. □

B SERVERFILLING IN THE MULTISERVER-JOB MODEL

B.1 ServerFilling in the Exponential Setting

LEMMA 7.1. *If jobs requiring at least k servers are present, ServerFilling fills all k servers.*

PROOF. ServerFilling maintains following invariant: whenever jobs requiring v servers are being scheduled, the number of remaining unfilled servers is divisible by v. The invariant is true when scheduling begins, because all server requirements v divide k. At the end of each step of scheduling jobs requiring a given number of servers v, the remaining number of unfilled servers must be divisible by v, because it decreased by a multiple of v, and was initially divisible by v. Then, when ServerFilling moves to schedule jobs requiring the next smaller number of servers v', the number of remaining servers is divisible by v', because $v'|v$. As a result, the only point at which jobs can no longer be added to service is when there are no remaining servers. Because jobs requiring at least k servers are present, ServerFilling must fill all of the servers. □

B.1.1 Improved bounds on $E[T_F]$.

To derive an improved bound on $E[T_F]$ in the exponential setting, we can separate time in front into time in service, which is simply $E[X]$, and time in the front but not in service. Note that if a job requiring v servers is the mth job in the front, then the total number of servers demanded by jobs in the front is at most $k + v - 1$. As a result, there can be at most $v - 1$ jobs in the front but not in service. The fraction of time that a job requiring v servers is the m job in the front, and is therefore in service, can be bounded by the expected time for which jobs requiring v servers are in service in total. This later quantity is equal to $\lambda E[X\mathbb{1}\{V = v\}]$. We can therefore bound the expected number of jobs in the front but not in service, and use Little's law to bound the expected time jobs spend in the front but not in service. This method often gives a significantly better bound on $E[T_F]$ than the naive bound $kE[S]$.

B.2 ServerFilling in the Divisible Setting

In the divisible (but not exponential) setting, ServerFilling uses a fairly conservative policy, for simplicity. ServerFilling tries to fill all k servers using jobs that require the same number of servers v. For a given $v \in \text{supp}(V)$, let $j_v(1), j_v(2), \ldots$ be the jobs in the system requiring exactly v servers. To fill the servers with jobs requiring v servers each, $\frac{k}{v}$ jobs are required. Recall that we assume that $v \mid k$, so all k servers can actually be filled. The newest job scheduled this way would be $j_v(\frac{k}{v})$. Among all v such that at least $\frac{k}{v}$ jobs requiring v servers are present, ServerFilling selects the v such that the job $j_v(\frac{k}{v})$ is as old as possible.

If no v that fills all of the servers exists, ServerFilling picks the v that fills as many servers as possible, ties broken as above.

We can guarantee that all servers will be filled if at least the following number of jobs are present:

$$c(k, \text{supp}(V)) = \left(\sum_{v \in \text{supp}(V)} \frac{k}{v} - 1 \right) + 1.$$

If at least $c(k, \text{supp}(V))$ jobs are present, at least one v will be able to fill the servers.

For an asymptotic understanding of $c(k, \text{supp}(V))$, note that regardless of V,

$$c(k, \text{supp}(V)) \leq \sigma(k) = O(k \log\log k),$$

where $\sigma(\cdot)$ is the sum-of-divisors function [14].

B.2.1 Results. In the divisible setting, the front contains at most the $c(k, \text{supp}(V))$ oldest jobs, as defined in Appendix B.2. As a result, the maximum front size $n = c(k, \text{supp}(V))$.

The front is full if and only if all k servers are occupied. The remaining size of a job j in service falls at rate $\frac{v_j}{k}$. As a result, whenever all k servers are occupied, the system has a total service rate of 1, in which case the system is busy, by on Definition 2.3. The model has bounded expected remaining size, and nonzero minimum service rate whenever a job is present. Because all of the assumptions in Section 2 are satisfied, we can apply our main result, Corollary 3.8.

To apply Corollary 3.8, we must quantify the amounts of work $w_{nonfull}$ and $w_{nonbusy}$, and the service-to-number ratio $r_{\min}$. The system is full if and only if it is busy, so $w_{nonfull} = w_{nonbusy}$. Specifically, $w_{nonfull} \leq (c(k, \text{supp}(V)) - 1)\text{rem}_{\max}$.

If $N(v)$ is the number of jobs requiring v servers present in the system, at least $vN(v)$ servers are filled, for any $v \in \text{supp}(V)$. The server-to-number ratio is minimized when this value is equal for all $v \in \text{supp}(V)$, and when $1 \in \text{supp}(V)$. As a result,

$$r_{\min} \geq \frac{N(1)}{k \sum_{v \in \text{supp}(V)} \frac{N(1)}{v}} = \frac{1}{\sum_{v \in \text{supp}(V)} \frac{k}{v}} \geq \frac{1}{\sigma(k)}.$$

Plugging the requisite values in Corollary 3.8 yields specific bounds on the mean response time $E[T^{MSJ\,Div}]$ in the Multiserver-Job model in the Divisible setting:

$$\begin{aligned}-(c(k, \operatorname{supp}(V)) - 1)\mathrm{rem}_{max} + E[S] &\leq E[T^{MSJ\,Div}] - \frac{\lambda E[S^2]}{2(1-\rho)} \\ &\leq (c(k, \operatorname{supp}(V)) - 1)\mathrm{rem}_{max} + \left(\sum_{v \in \operatorname{supp}(V)} \frac{k}{v} \right) E[S].\end{aligned}$$

These bounds can be improved significantly by focusing on the specific distributions of the random variables V and X, rather than focusing on the just the size S.