Optimal Scheduling in Multiserver Queues

Isaac Grosof

CMU-CS-23-128

July 2023

School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

Thesis Committee:
Mor Harchol-Balter, Chair
Alan Scheller-Wolf
Anupam Gupta
Weina Wang
Michael Mitzenmacher (Harvard)

Submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy.

This research was supported by the National Science Foundation under award numbers CMMI-1538204, XPS-1629444, CSR-1763701, CSR-180341, and CMMI-1938909, and by a Siebel Scholar Award, Class of 2023.

The views and conclusions contained in this document are those of the author and should not be interpreted as representing the official policies, either expressed or implied, of any sponsoring institution, the U.S. government, or any other entity.

Keywords: queueing theory; scheduling; optimal scheduling; response time; multiserver; tails; M/G/1; M/G/k; dispatching; Shortest Remaining Processing Time; SRPT; First-Come First-Served; FCFS; Nudge; Guardrails; work; Multiserver-job; ServerFilling; ServerFilling-SRPT; DivisorFilling; WCFS; Finite Skip; RESET; MARC; Gittins index; heavy-traffic; stochastic improvement; tail probability

To my parents, and to Xinyu.
You helped my dream come true.

Abstract

Scheduling theory is a key tool for reducing latency (i.e. response time) in queueing systems. Scheduling, i.e. choosing the order in which to serve jobs, can reduce response time by an order of magnitude with no additional resources. Scheduling theory is well-developed in single-server systems, where one job is processed at a time. However, little is known about scheduling in multiserver systems, where many jobs are processed at once. Results are especially limited in stochastic multiserver scheduling theory. Today's datacenters have thousands of servers, and scheduling theory is unable to analyze such systems.

This thesis proves the first optimality results and first closed-form bounds on mean response time for scheduling policies in stochastic multiserver models which reflect the behavior of modern computing systems. The thesis explores three themes:

1. I start by studying one-server-per-job multiserver models, and prove the first results on optimal scheduling in that setting. Optimality results are proven for both a central-queue model and a dispatching model. I invent a novel class of dispatching policies, guardrails, to achieve these results.
2. Next, I study the multiserver-job (MSJ) model, where different jobs require different amounts of resources to be served. I prove the first characterization of mean response time for any scheduling policy in the MSJ model, as well as the first optimality results. I invent novel scheduling policies, ServerFilling and ServerFilling-SRPT, to achieve these results.
3. Finally, I study the effects of scheduling on the tail of response time, rather than mean response time. The prior state-of-the-art for scheduling for the tail was First-Come First-Served, which was conjectured to achieve optimal asymptotic tail of response time. I invent a novel scheduling policy, Nudge, which I prove to be the first policy to outperform FCFS's asymptotic tail of response time.

Acknowledgments

I have thoroughly enjoyed my PhD experience, and I will fondly remember everyone I met along the way. I want to thank you all, and I will do my best.

To start with, I want to thank my advisor, Mor Harchol-Balter. When I came to the CMU open house, I didn't know what I wanted to work on. I knew I wanted to study theory, I knew I wanted something new, and I knew I wanted an advisor who I could work closely with and learn from. That day, Mor gave me a copy of her textbook, and I first saw queueing theory. When I came to CMU for the Introductory Course, Mor gave the very first talk, and said "If you want to study math and become a professor, work with me." It's gone well ever since. Mor has been everything I could have wanted, supporting me, guiding me, teaching me, and setting me on the course for my future journey. I couldn't've done it without you, Mor. Thank you, now and always.

I also want to thank my thesis committee. Alan Scheller-Wolf has been a wonderful collaborator, a calm and wise voice in our research meetings. Weina Wang has been an enthusiastic voice of queueing theory excitement ever since the day I saw her give her job talk over five years ago. Anupam Gupta has been a steward of the CMU theory community, through his class and through organizing seminars and teas. Michael Mitzenmacher has been a joy to collaborate with and talk with, taking queueing theory in new directions.

I want to thank my research colleagues. Most of all, I want to thank Ziv Scully. Ziv brings research enthusiasm, wonderful insight, and most of all a joy for discovery to every day we've spent working together. I've loved talking through so many problems together, and I can't wait to talk through more. Some of my fondest memories have been sharing a hotel room and staying up late talking about queueing theory together. And I love Ziv's creativity and enthusiasm for music and programming languages and a host of other subjects.

I want to thank Yige Hong, a diligent and insightful research partner. It's made my heart warm to see Yige become a confident and independent researcher. I want to thank Ben Berg and Daniel Berger, both of whom I wish I'd worked with more, and perhaps we will in the future. Kunhe Yang was a spark of joy and research camaraderie through a long year of COVID. And Jayson Lynch, though our research was separate from my PhD, I can't thank enough for bring about my first collaboration. Naifeng Zhang was a great partner for diving into the novel world of SAT solving, and Marijn Heule was a great mentor for that journey.

Thank you to my postdoc advisors to be, Siva Theja and Srikant. I can't wait to work with each of you.

Thank you to my postdoc advisors who could have been, Rhonda Righter and Adam Wierman. I'm still looking to find some way to collaborate, I think you're both wonderful.

Thank you to Jalani Williams, a wonderful buddy in the CMU queueing theory research journey.

I want to thank the wonderful members of the research community, who I hope to see more and to collaborate with in the future. Thank you to Daniela Hurtado Lange – I always look forward to hugging you, and I'm looking forward to being your colleague. Thank you to Igor Kadota, another wonderful colleague to be. Thank you to Kristy Gardner, my academic sibling and wonderful to see at conferences. Céline Comte, I love talking about research with you.

Thank you to the wider CMU queueing theory research community: Ben Mosely, Gauri Joshi, Kyra Gan, Sara McAllister, Tuhinangshu Chowdhury, and Alec Sun.

Thank you to many more I've met at conferences and visits, in no particular order: Benny van Houdt, Tim Hellemans, Runhuan Xie, Mark Squillante, Jing Dong, Eytan Modiano, Debankhur Mukerjee, Daan Rutten, Sid Banerjee, Jamol Pender, David Goldberg, Abel Souza, Piotr Indyk, Stefan Perko, Doug Down, Augustin Chaintreau, and many, many more.

I want to thank the academic staff who have helped me immeasurably. Nancy Conway, Pat Loring, and Matt McMonagle with the administrative side of things. Deb Cavlovitch and Jenn Landefeld, for keeping the department running smoothly. And Charlotte Yano, with whom I organized many a delightful Open House and IC event.

I want to thank my wonderful friends from throughout my PhD. Myra Dotzel, the teas and the photographs and the warmth, thank you so much. Gaurav Manek, I treasure every pizza. Francisco Maturana, I'm glad every time I see you. Paul Gölz, I'm looking forward to seeing you more. Sol and Yvonne, thank you for all the board games. Michael Rudow, our lunches have been wonderful. Charlie McGuffey, thanks for the D&D. Pallavi Koppol, a great leader. David Kahn, I've loved every board game. Marissa Walter, I've loved every outing we've shared. Katherine Kosaian, I'm looking forward to meeting up more in the future. Jenny Lin, your sweaters are beautiful and I love talking with you. Evan Lohn, I loved our long talks and cute bunnies on the way back from board games. John Clune, I could (and did) listen to you lie for hours, and had a great time. Aditi Kabra, I loved our long conversations. Bingbin Liu, you were a great workout buddy and a great friend. Saranya Vijayakumar, thank you for organizing so many events and making me feel so included. Jay Bosamiya, I will never look at computer security the same. Margarida Ferreira, I've loved our long walks in Schenley park.

Thank you to my friends from the Humanist League, William Chen, Seth Eisenberger, Andres Garcia, Nicole Guccione, Daniel de Angelo, Andy Norman, Nathan Rooney, Melody Tang, Krystal Jackson, Chanel Menocal, Alex Fogelson, Clara Nelson, Jeremy Huang, Jeremy Rich, Illona Altman, Mateo Arevalo, Chay Gruber, Alex Litzenberger, Lovisa, Ariel Tan, Greg Volynsky, and Aiko Kyle.

Thank you to my CMQ+ and TINA friends, Justine Hess, Jack Burgess, Clarice Meffert, Stella, Elizabeth Chang-Davidson, Dani Bork, and Carrie Chisholm.

Thank you to my roommates, Priya, Alex, Octavio, Roger, and Nirav. Thank you to my officemates, Ainesh, Klaas, Arjun, and Tianqin. Thank you to my my mentorship pod, Lucio Derry, Chris van Merwijk, Emanuel Tewolde, NJ, Aviv Bick, and Tianqin Li, and to my mentee Bernardo Subercaseaux.

Thank you to Ray Ware, Cayden Codel, Kevin Pratt, Mark Gillespie, Shir Maimon, Victor Akinwándé, Dorian Chan, Ananaya Joshi, Harrison Grodin, Orestis Chardouvelis, Sophia Roshal, Sherry Sarkar, Heather Newman, Catalina Vajiac, Mik Zlatin, Zoe Wellner, Justin Raizes, Justin Whitehouse, Anup Agarwal, Madgalen Dobson, Edward Chen, Sydney Gibson, Jessie Grosen, Roie Levin, Ben Stoler, Ke Wu, Emre Yoclu, and Giulio Zhou.

I want to thank my family. Eliana, I really appreciate all of the long conversations, and working on the memoirs. Jacob, I love talking about history and philosophy and playing games together. I treasure the advice and stories from Grandma Edith, Grandpa Jim, Grandma Miriam and Grandpa Gene. Thank you to Uncle David, Aunt Jenny, Uncle Victor, and Jae. And thank you so much to Mom and Dad. Mom, our long calls and advice back and forth has meant the world for me. You did so much for me over so many years, raising me to be the person I am now. Thank you, again and again. Dad, you showed me how exciting research could be, and I've loved talking with you about every topic under the sun. Thank you, always.

Thank you, Xinyu. I've never loved anyone the way I love you. Every time we see a cute animal, or lie together on the couch, or talk about the future, I love you even more. You've always supported me no matter what's going on for either of us, and it's meant the world to me. I'm looking forward to our life together. Thank you, forever.

Contents

List of Figures

List of Tables

Part I

Introduction

Chapter 1

Introduction

1.1 Introduction to This Thesis

Queueing theory Queueing theory is an excellent tool for modeling and analyzing computing systems. It has been used to model webservers, operating systems, databases, datacenters, supercomputers, networks, and a variety of other computing systems. Queueing theory provides simple mathematical models to capture the essence of these complex systems. Within these simple models, we can theoretically analyze the performance of these systems through performance metrics such as mean response time[1].

Single-server vs. Multiserver Unfortunately, almost all of queueing theory centers on the single-server queue [40]. Here "server" is an abstraction referring to a unit of computing power that can process one job at a time, such as a CPU core, a GPU, a network switch, etc. By contrast, computing systems are almost all multiserver systems. Computing clusters contain thousands of machines, operating systems and databases manage multiple cores, and networks have hundreds of interlinked switches.

Tractability Single-server queueing systems are studied because they are tractable. A wide variety of techniques have been developed to deeply understand the behavior and performance of single-server queueing models. By contrast, multiserver queueing systems require high-dimensional state spaces, making them far more difficult to understand. Multiserver queueing systems are not understood except for a limited class of settings.

Dichotomy in scheduling Nowhere is the dichotomy between single-server and multiserver research more apparent than in scheduling: deciding in which order to serve the jobs. While scheduling in the single-server setting is well understood, scheduling in multiserver settings is very poorly understood.

Scheduling theory Scheduling theory studies the order in which jobs are served, and how this decision impacts performance. Common scheduling policies include First-Come First-Served (FCFS), which serves jobs in the order they arrive, Shortest Job First (SJF), which prioritizes the job of least duration, and Shortest Remaining Processing Time (SRPT), which goes even further than SJF by allowing the preemption of an ongoing job in order to switch to a newly

[1]The response time of a job is the duration from the job's arrival to its completion. Response time is also known as sojourn time. "Mean" refers to the average across all jobs.

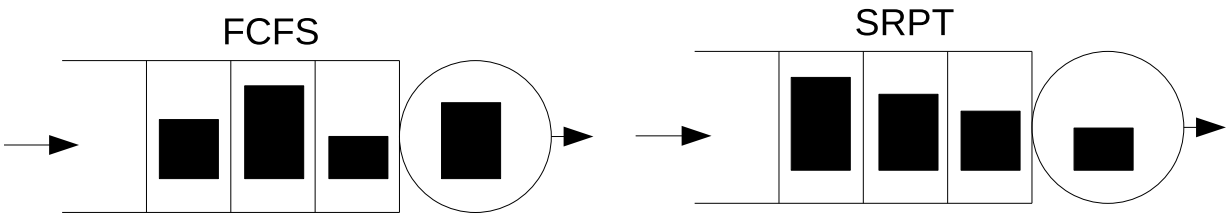


Figure 1.1: The First-Come First-Served (FCFS) and Shortest Remaining Processing Time (SRPT) scheduling policies, in the single-server setting. FCFS orders jobs in arrival order, while SRPT orders jobs by remaining size (remaining service duration).

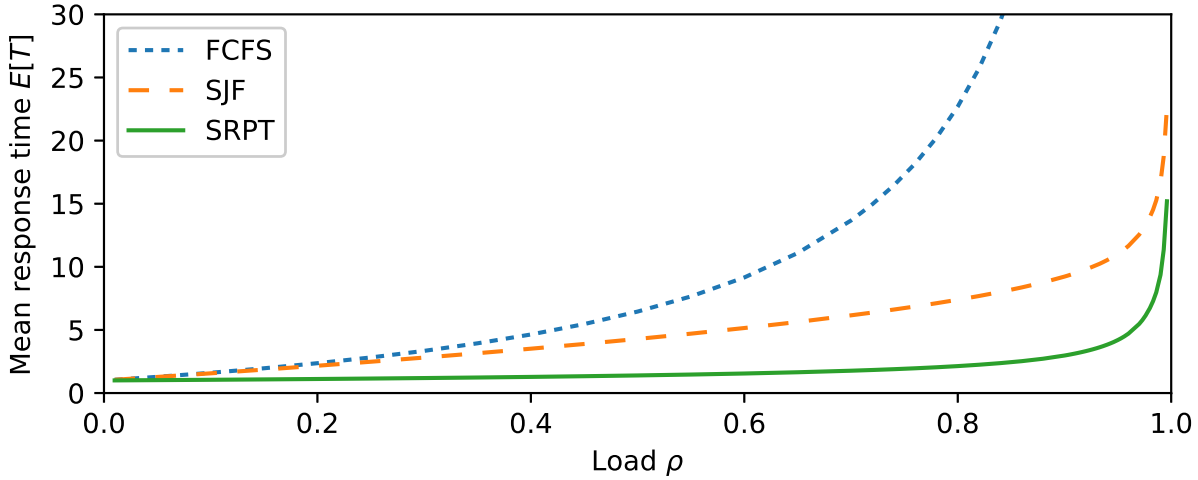


Figure 1.2: Mean response time under the FCFS, SJF, and SRPT scheduling policies in the M/G/1. Job size distribution S is a hyperexponential distribution with squared coefficient of variation $C^2 = 10$, a relatively high-variability workload, mimicking real computing systems. Load $\rho = \lambda\mathbb{E}[S]$ denotes the fraction of time for which the single server is active. For each of these policies, mean response time has been theoretically characterized in the M/G/1 [104, 196].

arrived job of lesser duration. When a job is preempted, it is returned to the queue, where it can be resumed later with no loss of progress. By contrast, SJF does not preempt jobs. To illustrate the FCFS and SRPT scheduling policies, we depict each system in Fig. 1.1.

Single-server scheduling In the single-server setting, the performance of a wide variety of policies has been exactly characterized, and that performance can be highly differentiated, especially in high-variance workloads common in computing settings [42, 101, 218]. Fig. 1.2 demonstrates the power of favoring short jobs over long jobs when trying to minimize mean response time. The setting of Fig. 1.2 is a *stochastic* model called the M/G/1, a single-server model where jobs arrive according to a Poisson process with fixed rate, and job sizes (i.e. durations) are sampled i.i.d. from a general job size distribution. By "stochastic", we mean that the model involves a sequence of random events happening over time. Note that in Fig. 1.2, the SRPT scheduling policy achieves the least mean response time, and therefore best mean response time. In fact, SRPT, the policy which most dramatically favors short jobs, is known to achieve *optimal*

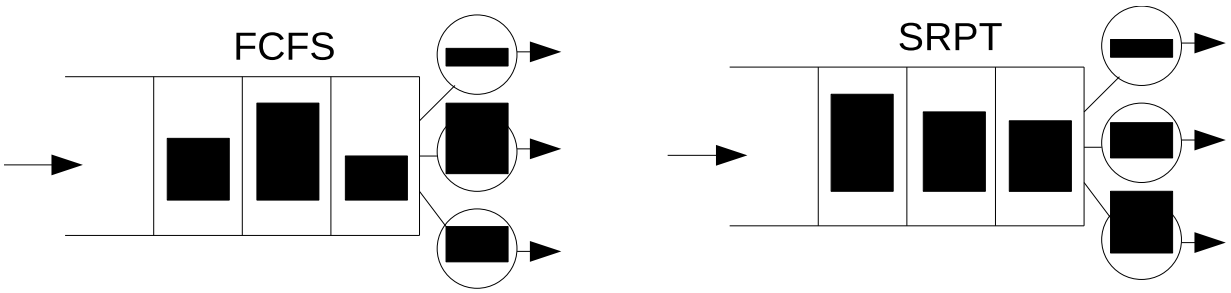


Figure 1.3: The FCFS and SRPT scheduling policies in the multiserver setting. Each policy serves the k "best" jobs according to its ordering. For FCFS, the k best jobs are the k oldest jobs, while for SRPT, the k best jobs are the k jobs with smallest remaining size.

mean response time in the single-server setting [194].

Modern single-server scheduling Even today, there are new research breakthroughs being proven on scheduling in the single-server queue. For instance, the recent SOAP technique massively generalizes the class of policies whose response time can be analyzed in the single-server queue, and specifically in the M/G/1 [201]. This technique enables the analysis of a variety of important scheduling policies in the M/G/1, including Shortest Expected Remaining Processing Time (SERPT) and the Gittins Index policy [73]. Both SERPT and Gittins focus on the setting of unknown job sizes, with Gittins being the optimal scheduling policy in the M/G/1 in that setting [72, 73, 199].

Multiserver scheduling But what about scheduling in a multiserver queue, as depicted in Fig. 1.3? What if we want to understand the benefit of SRPT in a 2-server system? Or a k-server system? A natural model for multiserver settings is the M/G/k model, where jobs are held in a central queue, and the scheduling policy selects which k jobs receive service at a given point in time. The arrival process is the same "M/G" process as the M/G/1: Poisson arrivals, i.i.d. sizes from a general distribution. Natural questions about the M/G/k include: Can we analyze the mean response time of M/G/k scheduling policies? Is the SRPT policy optimal, as it is known to be optimal in the M/G/1? Sadly, almost nothing is known about M/G/k scheduling, even when k is only 2 servers.

Worst-case multiserver scheduling One area of multiserver scheduling in which notable results have been proven is *worst-case* multiserver scheduling, in contrast to the stochastic settings that this thesis focuses on. Worst-case analysis focuses on comparing an online scheduling policy, which must react to jobs as they arrive, to the optimal offline policy, which has access to the entire arrival sequence in advance. This comparison is captured by the competitive ratio of a scheduling policy, which is the worst possible ratio of performance (e.g. mean response time) between the online and offline policies, over all possible arrival sequences. The lower this worst possible ratio, the better the policy. Some scheduling policies are better than others, under this metric, but it has been proven that none can achieve a constant competitive ratio [135, 136]. The best possible competitive ratio is logarithmic in the ratio of largest to smallest job size, which is achieved by multiserver SRPT [135, 136]. This is where the worst-case results end: For every online scheduling policy, there is some arrival sequence for which it performs arbitrarily worse than the optimal offline policy.

Real world is stochastic However, real world computing workloads are far more reminiscent of stochastic models, not worst-case models. The worst-case workloads require precisely timed arrivals of precisely determined sizes, in exact synchrony for huge numbers of arrivals. This is simply not what we see in real workloads. Real-world experimental work has shown that large-scale computing workloads are well approximated by stochastic queueing models [9, 56].

Real workloads typically consist of a variety of unrelated sources of arrivals, interspersed in an essentially uncorrelated fashion. This mixture of flows naturally gives rise to uncorrelated arrival times, modeled by a Poisson process, by the Palm–Kintchine Theorem [122, Page 221-228]. It also gives rise to uncorrelated job sizes, modeled by i.i.d. samples from a size distribution. This real-world pattern is the impetus for the M/G models which are the focus of this paper.

Structure of this thesis This thesis consists of 3 themes: stochastic multiserver scheduling, multiserver-job scheduling, and scheduling for advanced performance metrics which focus on the tail of response time. In the remainder of this chapter, we introduce each of these themes.

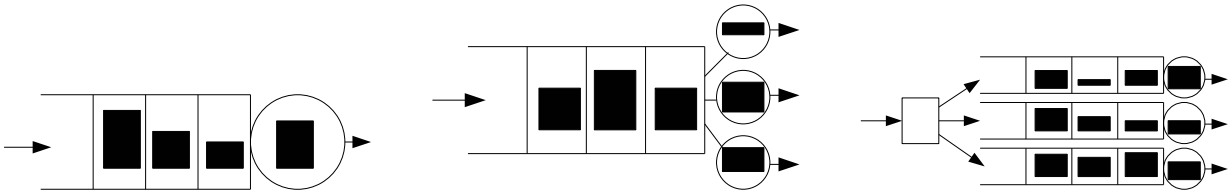
Theme 1: Stochastic multiserver scheduling In Section 1.2, we introduce the first theme, stochastic multiserver scheduling. Prior analysis of stochastic scheduling has primarily focused on single-server models. Prior analysis of multiserver scheduling has primarily focused on worst-case models. In reality, scheduling is vital for large computing systems, which require stochastic multiserver models. In this thesis, we give the first closed-form analytical bounds and optimality results for stochastic multiserver scheduling.

Theme 2: Multiserver-job scheduling In Section 1.3, we introduce the second theme, multiserver-job scheduling. Standard multiserver models, such as the M/G/k, model each job as requiring one server in order to run. In the real world, different jobs typically require different amounts of resources: cores, GPUs, memory, bandwidth, etc. We model these differing resources as jobs requiring different numbers of servers, which we call the "multiserver-job" model. In this thesis, we give the first closed-form analytical bounds on mean response time for stochastic multiserver-job scheduling.

Theme 3: Advanced performance metrics: Tail metrics In Section 1.4, we introduce the third theme, scheduling for performance metrics which capture the tail of response time. Standard queueing analysis focuses on *mean* response time as the primary performance metric. The primary reason for this focus is convenience, rather than importance. Many key queueing techniques, such as Little's law, can only be used to analyze mean response time [104]. However, in practical systems, the most important metrics are often *tail* metrics, such as the probability that a job's response time exceeds some threshold, or the 99th percentile of response time. While tail metrics are far more important than mean response time, much less is known about them. In particular, while it is known that SRPT scheduling optimizes mean response time, essentially nothing is known about scheduling to optimize the tail of response time. In this thesis, we prove the first results on optimizing tail metrics using stochastic scheduling.

1.2 Theme 1: Stochastic Multiserver Scheduling

Large-scale computing We need to understand large-scale computing systems, whose behavior is best captured by stochastic multiserver models. Based on our experience with single-server models, we have the potential to dramatically improve performance if we can find the right



(a) M/G/1 (b) M/G/k (c) Dispatching M/G/k

Figure 1.4: Standard queueing models: In each case, jobs arrive according to a Poisson process, and have i.i.d. sizes. In the M/G/1, the scheduling policy can choose any single job to serve. In the M/G/k, the scheduling policy can choose any k jobs to serve at a time, at equal rates. In the dispatching M/G/k, the dispatching policy assigns each job to a server upon arrival, and the scheduling policy at each server can choose any single job to serve at that server.

scheduling policies. We need to understand how scheduling decisions affect the performance of stochastic multiserver models.

1.2.1 Topic and goals

Topic: Multiserver scheduling This theme focuses on how scheduling affects the performance of systems with more than one server. We consider scheduling in two models, both of which involve scheduling across many servers. We depict these two models in Fig. 1.4, as well as a one-server model for comparison.

Central queue: In the central-queue model, depicted in Fig. 1.4b, jobs arrive to a single queue, and the scheduling policy determines which k jobs are served at a given point in time, where k is the number of servers. A common central-queue model is the M/G/k model. The key question in this model is how the scheduling policy affects the system's performance.

Dispatching: In the dispatching model, depicted in Fig. 1.4c, each job arrives to a front-end dispatcher. Based on the dispatching policy the job is sent to one of the servers, where it waits in the queue. Then, the scheduling policy determines which of the jobs in a particular server's queue is served by that server. Notably, jobs cannot migrate between servers in this setting. In the dispatching setting, we ask how both the dispatching and scheduling policies affect the system's performance.

Goals: Analysis Our first goal is to analyze the performance, specifically the mean response time, of scheduling (and dispatching) policies. In queueing theory, to "analyze" a model's mean response time means to derive a mathematical formula to calculate the mean response time as a function of the model parameters, which in this case are the arrival rate and job size distribution. The best possible analysis would be *exact* analysis, where the exact mean response time is derived, but even closed-form bounds or asymptotic limits would be an important breakthrough in analyzing stochastic multiserver models. We are interested in analyzing both existing scheduling policies, such as SRPT in the central-queue setting, as well as newly-invented policies.

Goals: Optimality Our second, more ambitious goal is to find or invent scheduling policies which achieve optimal mean response time. Finding the exact optimal policy may be overly ambitious, due to negative worst-case results [135], so a more feasible goal is to find *asymptotically*

optimal scheduling policies. A natural asymptotic limit is the "heavy traffic" limit, the limit as the load approaches the capacity of the system. In this limit, the number of jobs increases and scheduling becomes more and more important. Because there are many jobs to choose from, the scheduling policy has many options and more potential for improvement over a naive scheduling policy. Our goal is to find scheduling policies which are asymptotically optimal in the heavy traffic limit.

1.2.2 State of the art

We now discuss prior work on analyzing mean response time in stochastic multiserver models.

FCFS analysis FCFS is the simplest central-queue scheduling policy, and the policy for which the most analysis is known. Note that when no scheduling policy is specified for the M/G/k, FCFS scheduling is assumed by default. Some of this analysis takes the form of approximations [134, 230]. These approximations are not accurate under all conditions [97], but they are tight in the heavy traffic limit. Upper and lower bounds are known, but they are not tight in general [44, 92, 93, 127].

Scheduling analysis Other than FCFS scheduling, there are no *closed-form provable bounds* on mean response time for scheduling policies in the general central-queue M/G/k.

There do exist *non*-closed-form characterizations of mean response time, using computational methods such the matrix-analytic approach. For instance, such methods have been used to analyze the such as the 2-server, 2-branch hyperexponential setting [207]. These computational methods can only handle a restricted set of scheduling policies in restricted workload settings.

There also exist results on *approximations* of mean response time. For instance, the "busy period transitions" technique has been used to approximate the mean response time of a variety of task-assignment-based multiserver scheduling policies [100, 174, 175]. Similarly, the Recursive Dimensionality Reduction technique has been used to approximate the mean response time of a preemptive-priority multiserver scheduling policy [111]. The approximations can be made increasingly accurate with increased computational power, but the approximations are not closed-form bounds.

Provable closed-form bounds on mean response time for general scheduling policies, and for general workloads, are open. Optimal scheduling is completely open.

Dispatching analysis More is known about the analysis of dispatching policies, especially when paired with FCFS scheduling. Many dispatching policies have been analyzed, including Join the Shortest Queue (JSQ) [64], Least Work Left [104], Size Interval Task Assignment (SITA) [108], and many variants of each of these policies. However, because FCFS scheduling is a suboptimal scheduling policy, the overall performance of each of these systems is likewise suboptimal. Essentially nothing is known about dispatching in concert with high-performance scheduling policies, such as SRPT, nor about achieving optimal mean response time in dispatching settings.

Scaling models The focus of this thesis, as well as of the above results, is on settings with a fixed number of servers: The 2-server setting, the 3-server setting, or in general the k-server setting for some fixed k. There has also been considerable work on scaling settings, where the number of servers k grows in synchrony with the load ρ, according to some joint asymptotic

growth function. Most scaling research focuses on FCFS scheduling, both in the central-queue and dispatching settings [57, 99].

1.2.3 Why it's hard

There are many reasons why very little is known about mean response time in stochastic multi-server scheduling settings, some of which we explore here.

Tagged-job method: Single-server A vital tool for single-server scheduling analysis is the *tagged-job* method. The tagged job method involves tagging a generic job as the job of interest, and focusing on its progression through the system. One quantifies the random amount of work the tagged job encounters in the system on arrival, the time the job waits to reach service, and ultimately its overall response time. Here "work" refers to the total size of all of the jobs in the system that the tagged job must wait behind, at a given moment in time. In the single-server setting, each of the random variables can be exactly quantified for many scheduling policies [195, 196]. Because the tagged job is generic, its response time captures the overall response time distribution of the system.

Tagged-job method fails Unfortunately, the tagged-job method does not generalize to the multiserver setting. Key to the tagged-job method is the fact that in the single-server setting, the server completes work at a constant rate, until it empties the system completely. This fact allows exact quantification of the distribution of work in the system. In the multiserver system, a variable number of servers may be active at a time, causing a variable rate of work completion, and hence preventing us from quantifying the work in the system. Without the ability to quantify the amount of work in the system, the tagged-job method cannot proceed.

Lindley recursion The Lindley recursion is a simple equation of random variables whose fixed point gives the steady state amount of work in the system [141]. For the M/G/1 with FCFS scheduling, these random variables are real-valued, and the recursion is relatively simple to solve. In the M/G/k, the random variable becomes a correlated random vector, namely the Kiefer-Wolfowitz workload vector, and the analysis becomes far more difficult [124]. Adding scheduling to that complexity is currently intractable.

Lack of decomposition In single-server settings, a scheduling policy orders the jobs, and service is clean and straightforward: The job with highest priority is served, and all other jobs must wait. This strict prioritization gives rise to decomposition results that allows us to analyze a single priority level at a time. In the multiserver setting, there is not a clean separation by priority: Jobs of a range of priorities are served at the same time, preventing any such decomposition results.

Dispatching: Disrupted arrival processes In the dispatching setting, if the dispatching policy is a static policy, such as the SITA policy [108], or an oblivious randomized policy, each server experiences a Poisson arrival process, allowing one to use well-understood single-server analysis techniques. However, to approach optimality, we want to spread the jobs out across the servers when dispatching. This gives rise to lower-variance arrival processes, which are beneficial for performance, but harder to analysis. Arrivals to a server are correlated with the set of jobs at that server, which cannot be handled by existing single-server analysis.

1.2.4 Our results

We give the first closed-form analytical bounds and optimality results for stochastic multiserver scheduling.

M/G/k/SRPT analysis & optimality In Chapter 2, we give the first closed-form analysis of SRPT in the M/G/k, which we refer to as SRPT-k. In particular, we prove the first closed-form bounds on the mean response time of SRPT-k. These bounds become tight in the heavy-traffic limit, as load approaches capacity. Moreover, we prove that SRPT-k yields asymptotically optimal mean response time in the heavy-traffic limit. This is the first closed-form analytical bound on any M/G/k scheduling policy, and the first scheduling optimality result in the M/G/k. In addition to SRPT-k, we also analyze and prove optimality for several more scheduling policies in the M/G/k, including PSJF-k and FB-k, the latter under a restricted class of job size distributions. Preemptive-Shortest-Job-First (PSJF) preemptively prioritizes the job of least original size, while Foreground-Background (FB) preemptively prioritizes the job of least *age*, namely the least service received so far.

Dispatching M/G/k: Guardrails In Chapter 3, we devise a novel class of dispatching policies for the dispatching M/G/k, which we analyze in concert with SRPT scheduling. In doing so, we give the first analysis of any nontrivial dispatching policy combined with SRPT scheduling. We prove bounds on the mean response time of the combined guardrails/SRPT policy, which become tight in the heavy-traffic limit, as load approaches capacity. We prove that guardrails/SRPT yields asymptotically optimal mean response time in the heavy-traffic limit. This is the first optimality result in the dispatching M/G/k setting.

1.2.5 New analysis techniques

The results in Section 1.2.4 require inventing new analytic techniques, which have been important in subsequent problems. We describe these techniques below.

Resource-pooled SRPT-1 Our first key technique is the resource-pooled SRPT-1 system, which is lower bound that we compare against to prove our optimality results. The *resource pooled* M/G/1 is a system with the same overall capacity as the M/G/k or dispatching M/G/k systems, but which can put all of that capacity into running a single job. Concretely, in the resource-pooled system, 1 job is run at a time instead of k jobs, but that 1 job is run k times faster than in the M/G/k. In the single-server system, it is well known that SRPT scheduling is the optimal policy for minimizing mean response time [194], and its mean response time analysis is also well known [196]. We refer to the resource-pooled M/G/1 with SRPT scheduling as the SRPT-1 system.

SRPT-1 lower bound The resource-pooled single-server system can be thought of as strictly more flexible than the multiserver system. By time-sharing one server between k jobs, a k-server policy can be emulated, and the single-server system has far more options. Thus, the mean response time of SRPT-1 is a lower bound on the optimal multiserver response time, in any multiserver setting. To prove optimality, it therefore suffices to show that our multiserver policies have asymptotically identical mean response time to SRPT-1.

Worst-case and stochastic fusion To analyze and bound the mean response time of SRPT-k and guardrails/SRPT, we analyze the policies relative to SRPT-1. In particular, we prove worst-

case bounds on the gap between the multiserver policies and SRPT-1, in terms of the amount of work in the system, in terms of the wasted capacity that a given job can encounter, and in terms of the server imbalance under the guardrails/SRPT policy. These worst-case bounds, when combined with our exact stochastic understanding of the SRPT-1 system, allow us to prove tight bounds on the multiserver policies. These bounds decompose into a asymptotically dominant term, matching the SRPT-1 system, and asymptotically negligible term, arising out of these worst-case bounds.

1.2.6 Potential impact

Immediate, real-world application Based on our results, the SRPT-k and guardrails/SRPT policies have the potential to dramatically improve performance in a variety of large-scale computing systems. Many large-scale computing systems, such as datacenters, cloud-computing services, and web server-farms, closely resemble the multiserver models considered in this section. Moreover, the sizes of the jobs encountered by these systems are extremely variable [101]. These systems typically do not heavily incorporate size information into their scheduling decisions. Our research shows that the performance of these systems could be dramatically improved by making use of that information.

Hurdles The road from theoretical results to real-world adoption requires overcoming a variety of hurdles. We explore some of these hurdles in the remainder of this section. We discuss these hurdles and provide ideas for overcoming them in Sections 2.10.4, 3.10.3 and 8.2.1.

Extensions of our work Our work makes some assumptions. First, we assume that job sizes are known, which is required for the SRPT scheduling policy. Second, we assume that all jobs are equally important. It is possible to generalize our results to remove these assumptions, thereby increasing the real-world applicability of our results.

In real-world settings, it is rare to know the exact size of a job before it is run. More common is some kind of estimate or partial information. It has long been known that the optimal way to make use of this partial information is via the *Gittins index* scheduling policy [72, 73, 199]. In subsequent work to this thesis, we gave the first analysis of Gittins-k, and proved its asymptotic optimality for mean response time [202]. The Gittins scheduling policy can be thought of as a generalization to the SRPT scheduling policy: In the limit as estimates become perfectly accurate, the Gittins policy simplifies to SRPT. The Gittins-k result can therefore be thought of as a follow-up to this result, increasing its practical impact.

In real-world settings, different jobs often have different importances: some are extremely time-sensitive, while others are more flexible. To theoretically model, it is convenient to give each job a priority, representing the cost incurred for each second waited by the job, and seek to minimize the mean cost. In the single-server system, the well-known $c\mu$-rule generalizes SRPT to achieve optimal mean response time. The $c\mu$-rule prioritizes jobs by their ratio of cost per second to remaining size. Using essentially the same analysis as in Chapters 2 and 3, one can show that the $c\mu$-k policy achieves optimal mean weighted response time, assuming that the range of possible cost-per-second values is bounded. Our subsequent work on the Gittins policy generalizes that result to arbitrary cost-per-second distributions [199].

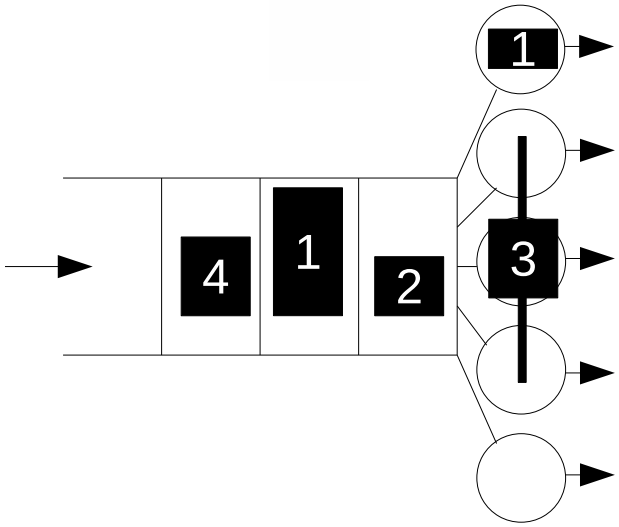


Figure 1.5: Multiserver-job model. Each job has a duration and server need. The duration is the time in service it requires, and the server need is the number of servers it requires in order to run. Visually, a job is represented by a rectangle, the job's duration is its vertical height, and the number inside the rectangle is its server need. For the job in service requiring 3 servers, the thicker rectangle's height represents its duration, while the skinny arms coming off indicate the 3 servers it is occupying. Duration and server need are sampled i.i.d. from some joint distribution, and arrivals are sampled according to a Poisson process.

1.3 Theme 2: Multiserver jobs

Multiserver-jobs in the world In the real world, jobs in large-scale computing systems have a huge spread of resource requirements. For instance, in Google's recently-published trace of their Borg computing-cluster scheduling system, the jobs' CPU requirements vary by a factor of 10^5 [218]. Moreover, some jobs require a large fraction of the entire system's resources, such as large machine-learning training jobs. As a result, it is vitally important to pack the jobs onto the servers in an efficient manner. Poor packing can lead to poor utilization: Only a small fraction of system resources are used. In practice, utilization is often poor [60, 120]. Scheduling has the opportunity to improve utilization, and by extension overall performance.

1.3.1 Topic and goals

Topic: Multiserver-job scheduling This theme focuses on scheduling in systems where different jobs require different amounts of resources. To model varying resource requirements, we consider the *multiserver-job* (MSJ) model, in which each job requires some number of servers, its *server need*. Each job is specified by the pair (server need, duration). The MSJ model is depicted in Fig. 1.5. The concept of "servers" can represent CPUs, GPUs, memory bandwidth, disk IO, network bandwidth, or any other resource that can constrain service. In the MSJ model, each job requires a fixed number of servers throughout its time in service, and any combination of jobs with total server need at most k can be served.

Goals: Analysis Our first goal is to analyze the mean response time of MSJ scheduling policies. As is the case throughout this thesis, exact analysis may be intractable, so characterizing bounds and asymptotic limits are our goals. Closed-form analysis of mean response time is unknown for all MSJ policies, so either analyzing existing scheduling policies, such as MSJ FCFS, or analyzing newly-invented policies, would constitute major breakthroughs.

Goals: Optimize Our second goal is to invent scheduling policies which achieve optimal mean response time. As is the case throughout this thesis, exact optimality is typically intractable, so we focus on finding asymptotically optimal scheduling policies. To understand asymptotic optimality, one must first consider the range of arrival rates which the system can tolerate. This ties into the notion of "stability", which we discuss in Section 1.3.2.

1.3.2 State of the art: Stability

Most prior work on the MSJ model focuses on characterizing its stability region: the range of arrival rates for which the number of jobs in the system does not diverge to infinity. In the MSJ model, unlike the M/G/k, the stability region depends on the scheduling policy, because servers may be left idle even though many jobs are present.

FCFS stability Much of the prior work focuses on the stability region of FCFS scheduling in particular [83, 186–188]. While FCFS stability is well-understood in several special cases, it is still an open problem in general.

Optimal stability Rather than concentrating on the stability region of FCFS, much prior work focuses on finding a scheduling policy with optimal stability region. Under some MSJ workloads, a policy can achieve *full stability*, which means that all servers are occupied all of the time in the limit as the arrival rate approaches the boundary of the stability region. Under other workloads, full stability is unachievable, but there is still some optimal stability threshold, the best possible over all policies. Several scheduling policies have been proven to achieve this optimal stability region:

MaxWeight: Under the MaxWeight policy, the scheduler considers all possible configurations of jobs that can fit on the k servers, and selects the configuration which maximizes a weighting function. The weight on each type of job is equal to the number of jobs of that type present in the system. The MaxWeight policy is proven to achieve the optimal stability region for all MSJ workloads [147, 148]. The MaxWeight policy can be defined for a wide variety of queueing settings, such as the generalized switch, where it is known to achieve even more advantageous properties [212]. No mean response time analysis is known for MaxWeight, owing to the complexity of selecting amongst all possible job configurations.

Randomized Timers: Randomized Timers is a scheduling policy that was invented to achieve optimal MSJ stability region without preemption. MaxWeight preempts jobs constantly, taking them out of service while partially complete and putting them back in the queue. In contrast, Randomized Timers never preempts jobs. Instead, Randomized Timers waits until a job completes. If another job of the same type is available, that job enters service to replace it. Otherwise, Randomized Timers randomly selects between placing a new job into service that fits on the available servers, or leaving the servers empty to free up more servers for other jobs to run. If these probabilities and timers are selected appropriately,

this policy achieves optimal stability region [71, 180, 181]. Again, no mean response time analysis is known, and the empirical mean response time is quite poor, as the policy relies heavily on having many jobs of the same type present.

Notably, both MaxWeight and Randomized Timers achieve the optimal stability region, regardless of whether full stability is achievable or not.

1.3.3 State of the art: Response Time

Little is understood regarding mean response time in MSJ settings.

2-server setting The only MSJ setting where mean response time has been analyzed in closed-form is in the special case of $k = 2$ servers, FCFS scheduling, and a workload in which jobs require an exponential service duration, independent of server need, which can be either 1 or 2 servers. In this setting, the exact z-transform of the steady-state number of jobs in system has been derived and independently rediscovered [31, 63]. Unfortunately, this transform is extremely unwieldy, requiring root-finding from a quartic equation. While analytically derivable, this obstructs analytical understanding.

Scaling models The focus of this thesis, as well as of the above results, is on settings with a fixed number of servers. There has also been recent work on scaling settings, where the number of servers k, the server need distribution, and the arrival rate all grow in synchrony, according to some joint asymptotic growth function. There has been some analysis of mean response time and some optimality results in this setting [113, 115, 226].

1.3.4 Why it's hard

The are many reasons why very little is known about mean response time in MSJ settings, some of which we explore here.

Defining work In the MSJ model, it is not even clear how to define the inherent work of a job. In single-server and one-server-per-job settings, the inherent work of a job, its *size*, is defined based solely on the job's service duration. In the MSJ system, the best definition of work is much less obvious. Within the MSJ model, we define a job's size to be the product of its server need and its duration. By doing so, we ensure that the rate at which work leaves the systems is proportional to the number of servers which are active, just as in the one-server-per-job setting. Just realizing that this is the important property to maintain from prior settings is already highly nontrivial.

Scheduling affects work completion Unlike simpler settings, the number of servers which are active, and hence the rate of work completion, is dependent on the scheduling policy in use. Under some scheduling policies, the jobs might pack poorly, and leave some servers unoccupied. Moreover, the number of servers left unoccupied can vary in hard-to-predict ways, as jobs arrive and complete. These issues make work-based analysis challenging, or even intractable.

Optimization-based policies are inherently complex Optimization-based policies such as MaxWeight are inherently resistant to analysis, when compared to index-based policies such as SRPT. With an index-based policy, from the perspective of a given job, all other jobs are either higher or lower priority, making it possible to quantify which jobs delay the given job, and how much delay those jobs will cause. For an optimization-based policy such as MaxWeight, the set

of jobs in service can only be determined via a global view of the system, and no classification into higher and lower priority is possible.

No candidates for optimality When we turn our eye towards optimal mean response time, no candidate policies present themselves: All known index-based policies do not achieve optimal stability region, and hence cannot achieve optimal mean response time. On the other hand, optimization-based policies are too complex to allow for analysis.

2-server exact analysis cannot be extended The unique mean response time analysis of any MSJ setting is in the special case of $k = 2$ servers, FCFS scheduling, and a workload in which jobs require an exponential service duration, independent of server need, which can be either 1 or 2 servers. This setting was solved via characterizing the exact z-transform of the number of jobs in system. Unfortunately, even solving this simple setting required analytically computing the root of a quartic equation. If the system was made any more complicated, with different duration distributions or more than 2 servers, analysis via this method would become intractable.

1.3.5 Results

We devise the first analysis and optimality results for more than 2 servers in the MSJ setting.

ServerFilling analysis In Chapter 4, we devise and analyze a novel MSJ scheduling policy which we refer to as ServerFilling. In particular, we prove bounds on the mean response time of ServerFilling. We show that ServerFilling's mean response time matches that of "resource-pooled" FCFS, to within an additive constant. By *resource-pooled* FCFS, we refer to a system with a single server running at speed k times the speed of a regular server, and where the scheduling policy is FCFS. This additive constant depends on number of servers k and on the service duration and server need distributions, but crucially does not depend on the load ρ. As a result, these bounds become tight in the heavy-traffic limit, as load ρ approaches 1, the boundary of full stability. This is the first closed-form analysis of any MSJ scheduling policy, and the first resource-pooling result in the MSJ model.

ServerFilling details & generalization In order to make this analysis possible, we focus on the special case of the MSJ system where every job's server need is a power of 2, and the number of servers k is a power of 2. In this setting, the ServerFilling policy is able to achieve a form of work conservation, ensuring that every server is occupied whenever a sufficient number of jobs are present. This work-conservation is key to the resource-pooling result. We also devise and analyze DivisorFilling, a scheduling policy which covers the more general setting where every job's server need evenly divides the total number of servers k. More generally, we specify and analyze an entire class of queueing models, which we refer to as "work-conserving finite-skip" models, all of which we show have a mean response time within an additive constant of resource-pooled FCFS.

ServerFilling-SRPT optimality While ServerFilling achieves full stability region, and we have analyzed its mean response time, this response time is in no way optimal. In order to achieve optimal response time, we need a new scheduling policy which still achieves the full stability region, and yet prioritizes smaller jobs. In Chapter 5, we define such a policy, which we refer to as ServerFilling-SRPT. A key step in defining ServerFilling-SRPT is ordering jobs by their remaining sizes, where size is defined as the product of remaining duration and server

need. We prove bounds on the mean response time of ServerFilling-SRPT which become tight in the heavy-traffic limit, proving that ServerFilling-SRPT converges to resource-pooled SRPT, and is thus heavy-traffic optimal. This is the first optimality result in the MSJ setting. Our results generalize to the unknown-size or estimate-size settings, where we prove that the ServerFilling-Gittins policy is heavy-traffic optimal. The Gittins policy is known to be the optimal single-server scheduling policy in those settings [72, 73, 199].

RESET and MARC: Analyzing MSJ FCFS While ServerFilling-SRPT achieves excellent mean response time, many systems today use FCFS scheduling, or related policies. Understanding the mean response time of MSJ FCFS is therefore an important problem. In Chapter 6, we analyze the FCFS scheduling policy for the MSJ model. We prove bounds on the mean response of MSJ FCFS, bounding its mean response time up to an additive constant not depending on the arrival rate. Because MSJ FCFS is *not* a work-conserving policy, we invent a completely new non-work-based framework of analyze its response time. This is the first closed-form analytic bound on mean response time for any non-work-conserving multiserver scheduling model. Our results generalize to settings of multidimensional resource constraints, and more generally to all "finite-skip" models, with work conservation no longer required.

1.3.6 New analysis techniques

The results in Section 1.3.5 require inventing new analytic techniques, which may be important for future problems. We describe these techniques below.

Scheduling to ensure work conservation Our ServerFilling and ServerFilling-SRPT results rely on a form of work conservation: Whenever at least k jobs are present, all k servers are busy. In simpler multiserver settings such as the M/G/k, work conservation is automatic, not scheduling-policy dependent, and is crucial to our results and previous results in that setting. In the MSJ setting, while work conservation is not automatic, it can be achieved by well-designed scheduling policies in specific server-need settings. This is the key insight behind the design of the ServerFilling policy.

The $\mathbf{W^2}$ method To analyze work in the ServerFilling system, our key insight is to analyze the rate of change of W^2, the square of the amount of work in the system. For any stationary random variable, the rate of increase must balance the rate of decrease, because the overall distribution is not changing. For random variables based on work, increases are due to arrivals, while decreases are due to work completion. Equating the rate of change of W^2 allows us to derive $\mathbb{E}[W]$, the mean work in the system. This characterization of $\mathbb{E}[W]$ is key to our analysis because it is closely related to mean response time.

Waste Via the W^2 analysis, we prove that the key term in bounding mean work $\mathbb{E}[W]$ is $\mathbb{E}[W(1-B)]$, which we call *waste*. Here B represents the fraction of servers that are busy at a given moment in time. The key insight is that waste, namely leaving servers idle ($B < 1$) when there is a significant amount of work in the system, has an enormous impact on mean response time: Our characterizations of $\mathbb{E}[W]$ in Chapters 4 and 5 consist of a resource-pooled term, plus an additive $\mathbb{E}[W(1-B)]/(1-\rho)$ term. As a result, tightly bounding waste is the key to proving resource-pooling results.

Replacing work conservation with constant drift In the MSJ FCFS studied in Chapter 6, work conservation does not hold: no matter how many jobs are present, servers may be left idle.

In particular, as jobs complete and service, the number of idle servers varies in complicated ways. As a result, work does not exit the system at a constant rate. In lieu of work, we need to invent some new random variable that exits the system at constant rate. More specifically, we need to find some random variable that has constant *drift*: The *expected* rate of change of the variable must be constant. The key to this random variable is relative completions, to be defined next.

Relative completions To invent the constant-drift random variable, we start with Q, the number of jobs in the queue. Obviously, this does not have constant drift: In some states, many fast jobs are in service, and the drift of Q is quite negative. In others, only a few slow jobs are in service, and the drift of Q is barely negative. However, this drift is only dependent on the state Y of the jobs in service. To compensate for this irregularity, we invent the *relative completions* function $\Delta(y)$: a function of the states of the jobs in service, which summarizes their impact on the upcoming completions of the system. $\Delta(y)$ smooths out the irregularities of the drift of Q. When combined, $Q - \Delta(Y)$ has constant drift. As a result, we can adapt the W^2 method to analyze $Q - \Delta(Y)$, and hence analyze the MSJ FCFS system.

1.3.7 Potential Impact

Immediate application: New policies Based on our results, ServerFilling, ServerFilling-SRPT, and the rest of the policies that we analyze in Chapters 4 and 5 have the potential to dramatically improve performance in large-scale computing systems, as shown in Fig. 5.4 in Chapter 5. Many such computing systems, such as datacenters and supercomputing centers, closely match the MSJ model. In these systems, packing inefficiency often lowers system utilization dramatically, so to the extent that efficient packing can be guaranteed using our policies, our research indicates that performance could be dramatically improved. However, ServerFilling and DivisorFilling are only applicable under restricted server needs and zero-cost preemption, which limit their impact.

Hurdles The road from theoretical results to real-world adoption requires overcoming a variety of hurdles. We discuss these hurdles and provide ideas for overcoming them in Sections 4.10.4, 5.10.3, 6.13.3 and 8.2.2.

Immediate application: Analysis Our analysis of the MSJ FCFS scheduling policy in Chapter 6 allows us to predict mean response time for a wide variety of more realistic MSJ settings, including settings with general server needs and no preemption. With this better understanding of mean response time, our research allows us to make theoretically validated capacity planning decisions, deciding how many servers are needed to achieve a given mean response time for a given workload.

Load balancing In many real-world MSJ systems, jobs arrive to a dispatcher, which then sends the job to one of several computing clusters. A common dispatching strategy is to send each job to the cluster where it will achieve the lowest response time. Previously, it was not understood how to select that cluster. We can now understand the effect of the duration and server need distributions and the number of servers at cluster on the response time at that cluster, enabling high-quality predictions about response time, to use as the basis for dispatching decisions.

Bandwidth sharing Beyond the setting of datacenters and supercomputing centers, the network scheduling setting bears a resemblance to the MSJ model. A packet flow is analogous to a job. The amount of bandwidth requested by a given flow is analogous to its server need. This is relevant in particular for low-latency network scheduling, where a given flow requires a

certain amount of dedicated network bandwidth. Of course, more work is needed to incorporate the network structure into the decision-making, but the resemblance is present.

1.4 Theme 3: Advanced performance metrics: Tail scheduling

The real-world importance of tail performance Standard queueing theory analysis focuses overwhelmingly on *mean* response time as its primary performance metric. But in the real world, service-level-objectives (SLOs) are not phrased in terms of mean response time. System operators don't monitor their mean response time for signs of degraded performance. Instead, SLOs require that a certain percentage of jobs complete in under a certain time. Operators monitor their 99th percentile of response time, and Amazon and Google have found that users are very sensitive to small delays in website loading times [140]. To address these real-world needs, we must analyze and optimize effects of scheduling on the *tail* of the response time distribution, not just on its mean.

1.4.1 Topic and goals

Topic: Scheduling for tails This theme focuses on the effects of scheduling on the tail of the response time distribution. By "tail of response time", we mean the *tail probability*, namely the probability that response time exceeds some threshold t. Characterizing this effect of scheduling on tail probability is extremely difficult. Queueing theory has not yet characterized how scheduling affects the tail probability, even in the single-server setting, with the multiserver setting completely out of reach for the time being.

Focus: Single-server tail scheduling We focus on studying the effects of scheduling in the single-server setting, and specifically in the M/G/1. There are two important tail scheduling regimes to study:

Asymptotic tail: To study the asymptotic tail of response time, we consider the behavior of the tail probability function. Letting T denote the response time distribution, the asymptotic tail regime studies the asymptotic decay rate of $\mathbb{P}\{T > t\}$ as t grows without bound, under a variety of scheduling policies.

Non-asymptotic tail The non-asymptotic tail of response time studies the behavior of $\mathbb{P}\{T > t\}$ at concrete t, under different scheduling policies. One can study the behavior of $\mathbb{P}\{T > t\}$, for fixed, concrete values of t, under a variety of scheduling policies.

The asymptotic tail regime is generally theoretically cleaner, and more is known about it, as we discuss in Section 1.4.2. The non-asymptotic tail regime is more resistant to analysis, but is more applicable to real-world objectives.

Focus: Dethroning FCFS Prior work has proven that the simple FCFS scheduling policy achieves *weakly optimal* asymptotic tail probability [27], and a long-standing conjecture claims that it is *strongly optimal* [232], terms we define in Section 1.4.2. Intuitively, this seems wrong: FCFS does not use any size information, so surely a better policy must exist. However, common

size-based scheduling policies such as SRPT are known to achieve atrociously poor asymptotic tail probability, so FCFS is currently the best-known policy. We are interested in studying scheduling policies in relation to FCFS, which is the previous best-known policy.

Goal: Asymptotic improvement on FCFS Our first goal is to improve upon FCFS in the asymptotic regime. We want to design a scheduling policy which makes use of size information to achieve better asymptotic tail probability than FCFS, contrary to standing conjecture [232].

Goal: Non-asymptotic improvement on FCFS Our second, more ambitious goal is to improve upon FCFS in the non-asymptotic regime. While empirically some policies outperform FCFS for some tail probability regimes, there is a dearth of analytical understanding of non-asymptotic regimes. We hope to prove that, for some policy and some threshold t, our policy is guaranteed to achieve better $\mathbb{P}\{T > t\}$ than a baseline policy such as FCFS. Our dream is to prove *stochastic dominance*: Designing a policy which achieves better $\mathbb{P}\{T > t\}$ than FCFS for all thresholds t, simultaneously.

1.4.2 State of the art

Prior work on the effect of scheduling on the tail of response time focuses on the Laplace-Stieltjes transform of response time, on asymptotic analysis, and on worst-case results.

Transform analysis The Laplace-Stieltjes transform is an important tool in studying the response time distribution of various scheduling policies. The transform function $\widetilde{T}(s)$ is defined as:

$$\widetilde{T}(s) = \mathbb{E}[e^{-sT}] = \mathbb{P}\{T < Exp(s)\}$$

By combining information across all values of s, the transform completely summarizes and uniquely determines the response time distribution. Moreover, for a wide variety of scheduling policies, the transform function is exactly known in closed form [104, 201]. The asymptotic tail probability behavior can often be analytically extracted from the transform expression.

Asymptotic tail probability The asymptotic tail probability behavior is well understood for a wide variety of scheduling policies, including FCFS, SRPT, FB, and Processor Sharing (PS) [27]. The PS policy splits the capacity of the server equally among all of the jobs present. For a large class of job size distributions known as *light-tailed* job size distributions, each of these policies are known to have asymptotic tail probability of the form

$$\mathbb{P}\{T > t\} = Ce^{-\alpha t} + o(e^{-\alpha t}), \tag{1.1}$$

for some constants α, C dependent on the scheduling policy and the job size distribution. Light-tailed job size distributions include bounded, exponential, hyperexponential, and phase-type distributions.

FCFS: Weakly asymptotically optimal For any light-tailed job size distribution, FCFS is known to achieve the optimum (maximum) possible constant α, across all scheduling policies [27]. As a result, FCFS achieves *weakly optimal* asymptotic tail probability. In contrast, SRPT, PS, and FB are all known to achieve the pessimal (minimum) possible constant α for light-tailed job size distributions [27]. As FCFS achieves the best possible exponential constant α, the only

remaining question is whether it achieves the best possible multiplicative constant C. A long-standing conjecture claims that FCFS does achieve the best possible C [232], which would make FCFS *strongly optimal* if the conjecture holds.

Sample-path comparison Non-asymptotic comparison between scheduling policies is far more limited. A couple of limited results have been proven, constructing size-based scheduling policies which outperform non-size-based scheduling policies in a "sample-path" manner. Sample-path improvement refers to achieving a lower response time for every job under an arbitrary arrival sequence of jobs, making this a worst-case result. Sample-path improvement is an even stronger result than the "stochastic dominance" discussed in Section 1.4.1. Such improvement is known relative to PS and FB, both of which are non-size-based policies which time-share between different jobs [66, 67, 172]. In general, it is possible to achieve sample-path improvement upon any time-sharing policy by using size information.

FCFS: Best maximum In the worst-case setting, one can show that for any finite arrival sequence of jobs, the FCFS policy minimizes the maximum response time among those jobs. More specifically, FCFS "lexicographically minimizes" the ordered sequence of response times: It achieves the optimal maximum response time, and among policies which achieve the optimal maximum, it achieves the optimal second-to-maximum response time, and so forth. Due to this optimality property, no policy can achieve sample-path improvement upon FCFS.

1.4.3 Why it's hard

There are may reasons why little is known about the tail of response time even in single-server scheduling settings, some of which we discuss here.

Little's law Little's law is a key tool in queueing theory, which can only be used to analyze *mean* response time. Denoting the number of jobs in system by the random variable N, response time by T, and arrival rate by λ, Little's law states that $\mathbb{E}[N] = \lambda\mathbb{E}[T]$ [142]. Little's law holds under an arbitrary scheduling policy. Little's law is invaluable in queueing analysis, because the number of jobs in the system N, being an instantaneous property of the system, is often much easier to characterize than response time T, and Little's law can be used to bridge the gap. Unfortunately, no equivalent of Little's law exists for the tail of response time $\mathbb{P}\{T > t\}$.

Recursive transform: Non-asymptotic While the transform of response time $\widetilde{T}(s)$ has been exactly characterized for a variety of scheduling policies, this transform is extremely difficult to reason about. For all but the simplest combinations of scheduling policy and job size distribution, the expression for the transform is a complicated, recursively defined function. Extracting properties of interest, such as $\mathbb{P}\{T > t\}$, while possible in principle, is intractable in actuality.

Scheduling using time-in-system While a wide variety of scheduling policies have been analyzed, existing analyzed policies almost always essentially ignore time-in-system information. Time-in-system is critical in making good scheduling decisions towards the goal of optimizing tail probability. For instance, while the acclaimed SOAP technique enables the analysis of many important policies, these policies cannot use time-in-system information, except through the narrow window of FCFS tiebreaking between otherwise equal-priority jobs. There are only a tiny handful of analyzed scheduling policies that use time-in-system in a more substantive way [211].

This inability to analyze most such policies obstructs our ability to design and analyze policies optimized for tail performance.

1.4.4 Our results

We design and analyze the first scheduling policy to achieve superior asymptotic and non-asymptotic tail performance to FCFS.

Nudge: Better asymptotic response time In Chapter 7, we design and analyze a novel scheduling policy which we refer to as Nudge. We characterize Nudge's asymptotic tail of response time, proving that, under light-tailed job sizes, it achieves the same optimal exponential decay rate α as FCFS and a superior multiplicative constant C to FCFS, where C and α are defined in (1.1). We thus show that Nudge is the first scheduling policy with superior asymptotic tail performance to FCFS, and the first policy to use size information to improve tail performance. In particular, we overturn the long-standing conjecture of FCFS's strong optimality.

Nudge: Stochastic dominance Furthermore, we prove that Nudge *stochastically dominates* FCFS, achieving smaller tail probability $\mathbb{P}\{T > t\}$ for every threshold t simultaneously. This is the first stochastic dominance scheduling result that is specific to the stochastic setting: All previous stochastic dominance results were worst-case results about time-sharing policies. This is a strong non-asymptotic tail performance result, proving improvement on metrics that practitioners care about, such as the fraction of jobs that complete in under a second, or the 99$^{\text{th}}$ percentile of response time.

1.4.5 New analysis techniques

The results in Section 1.4.4 require inventing new analytic techniques, which may be important in the future, and which we describe below.

Relative analysis Our Nudge policy is only a small deviation from FCFS: A small 'nudge' in the right direction. Our analysis, rather than studying Nudge in isolation, characterizes the difference in response time distribution between FCFS and Nudge. This leverages decades of work characterizing the tail of response time of FCFS as the starting point for our analysis.

Ordering-based scheduling Like FCFS, Nudge makes heavy use of arrival order information when deciding how to scheduling jobs. Arrival ordering information encodes time-in-system information in a way that is more amenable to theoretical analysis than working directly with the numerical time-in-system. In particular, time-in-system changes while a job waits in the queue, introducing complication into the analysis, while arrival ordering is fixed from the moment the job arrives. In this way, our analysis avoids the technical challenges that have impeded the analysis of more general time-in-system-based scheduling policies.

1.4.6 Potential Impact

Immediate, real-world application Based on our results, Nudge can serve as a drop-in replacement for FCFS in essentially any queueing system, regardless of whether that system's objectives focus more on mean response time or the tail of response time. Like FCFS, Nudge is a nonpreemptive policy, and is therefore practical to use in the same settings as FCFS Many

system operators are hesitant to deploy more aggressive scheduling policies such as SRPT due to its detrimental effects on the tail of response time, but Nudge has no such downside.

While Nudge is defined to use exact size information, it can easily accommodate approximate or estimated size information. Nudge only uses size information to classify jobs into one of two size categories, and approximate classification is sufficient.

Hurdles Beyond approximate size information, the road from theoretical results to real-world adoption requires overcoming a variety of hurdles. We discuss these hurdles and provide ideas for overcoming them in Sections 7.14.5 and 8.2.3.

Extensions of our work Nudge is merely the first step into using size information to improve tail scheduling. Researchers are already studying generalizations of Nudge, and characterizing the situations under which such generalizations can achieve even better tail performance than Nudge [222].

While we analyzed Nudge in the single-server model, the multiserver model more accurately resembles many real-world applications. As even the analysis of FCFS in the multiserver setting is much less well-developed, significant challenges stand in the way of transferring our results to the multiserver setting. However, the core of our analysis should transfer to the multiserver setting, giving hope for such results.

1.5 How to read this thesis

This thesis contains 8 chapters: The introduction, 6 content-focused chapters comprising the 3 themes of the thesis, and the conclusion. The introduction (Chapter 1) and conclusion (Chapter 8) are new content, written fresh for this thesis. Each of the 6 content-focused chapters primarily consists of a paper that I have written over the course of my PhD. In each content-focused chapter, I have added additional introductory and concluding sections, fresh for this thesis. The rest of of each content-focused chapter is a reiteration of an existing paper, with better formatting in some cases due to a lack of length restrictions. I have also fixed inaccuracies in a couple places.

In each case, the material that is written fresh for this thesis is aimed at a more general audience, such as people who may be new to queueing theory, while the preexisting material (the papers) is aimed at a more technical audience.

This thesis does not need to be read end-to-end. The thesis consists of 3 themes, each of which can be read independently. When reading a theme, first read the corresponding material in the introduction, then the corresponding content-focused chapter(s), then the corresponding material in the conclusion.

If you've read parts of this thesis and want to get in touch, either to ask questions, make a connection, or for whatever reason, feel free to reach out. My website at the time of writing this is `https://isaacg1.github.io`, which should list ways of contacting me. If it doesn't exist anymore, or is out of date, just search for my name – I'm the only person with my name. If you want to prove you've read this far, you can include the phrase "Special system, more applied" in your email or contact message. Those used to be first words on the 50th, 100th, 150th, and 250th pages of this thesis, rearranged. The 200th page used to be blank, and there used to be no 300th page. I think the phrase does a pretty good job of communicating what this thesis is about, for a pile of randomly selected words.

Part II

Multiserver Queues

Chapter 2

Optimal Central-Queue Multiserver Scheduling: SRPT-k

This chapter is based on the paper "SRPT for Multiserver Systems", published in IFIP Performance 2018, written with my coauthors Ziv Scully and Mor Harchol-Balter [81].

2.1 General Introduction

Modeling modern computing systems Modern computing systems typically contain huge numbers of servers, allowing them to process many jobs at once. In these multiserver systems, the *scheduling* decisions, deciding what order in which to process the jobs, has a major impact on the performance of these systems. Improving the scheduling policy can improve mean response time by an order of magnitude, without any additional resources.

To capture the behavior of these computing systems, we can use a multiserver queueing model, depicted in Fig. 2.1. Jobs arrive randomly over time, and wait in a central queue. Each job requires one of the k servers in order to run, so the scheduling policy can select any k jobs to run at a time. Two important scheduling policies, which we depict in Fig. 2.1, are First-Come First-Served (FCFS) and Shortest Remaining Processing Time (SRPT). FCFS always serves the k oldest jobs in the system. SRPT always preemptively serves the k jobs of least remaining *size*

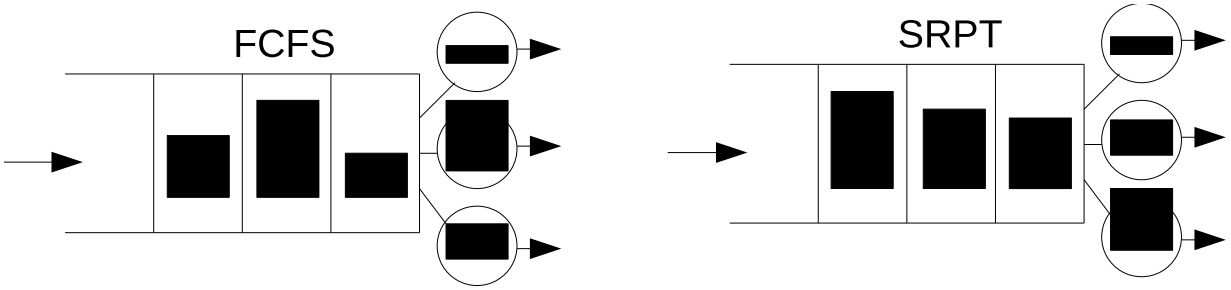


Figure 2.1: The FCFS and SRPT scheduling policies in the multiserver setting. Each policy serves the k "best" jobs according to its ordering. For FCFS, the k best jobs are the k oldest jobs, while for SRPT, the k best jobs are the k jobs with smallest remaining size.

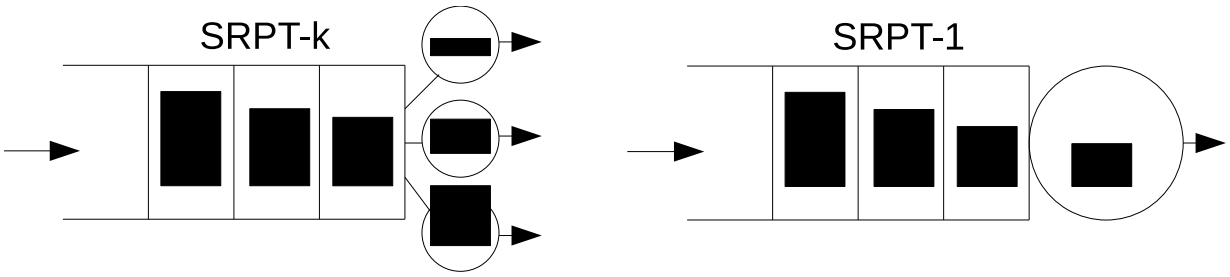


Figure 2.2: The SRPT-k system, showing the SRPT policy in a multiserver setting, and the SRPT-1 system, showing the SRPT policy in a resource-pooled setting, with one gigantic server.

in the system, where a job's size is its service duration. *Preemption* refers to pausing a job in service, and returning to it later. We are interested in the performance effects of these scheduling policies. A natural measure of a policy's performance is its *mean response time*, the mean time from when a job arrives to when it completes.

Multiserver scheduling not understood Unfortunately, from a theoretical perspective, little is known about stochastic multiserver scheduling. No closed-form formula for mean response time is known, nor are any closed-form bounds on mean response time. The optimal scheduling policy is likewise unknown.

In contrast, in the single-server scheduling setting, far more is understood. The scheduling policy that minimizes mean response time is the SRPT policy [194]. Moreover, the mean response time of SRPT has been theoretically analyzed in the M/G/1, a common single-server setting: An exact formula for mean response time is known [196]. The M/G/1 is a stochastic queueing model, with arrival times distributed according to a Poisson process and job sizes sampled i.i.d. from a general distribution.

Is multiserver SRPT optimal? Because SRPT achieves optimal mean response time in the M/G/1, we ask: Does SRPT achieve optimal mean response time in the M/G/k? The M/G/k is the multiserver analogue of the M/G/1: It has the same job arrival process. The M/G/k has the structure shown in Fig. 2.1: Jobs wait in a central queue, and the scheduling policy selects any k jobs to serve at a time. Each job occupies a single server. In addition to examining SRPT's optimality, we are also interested in analysis: theoretical characterization of SRPT's mean response time, either exact characterization or upper or lower bounds.

Key idea: Resource pooling Our key technique is to compare SRPT in the M/G/k, which we call SRPT-k, against a resource-pooled system where all k servers have been combined into one gigantic server. This gigantic server can serve any job at k times the speed of the original k servers. The resource-pooled system is a huge upgrade over the original M/G/k: One could split the gigantic server's efforts k ways, to emulate the original system, but one also has far more options. As a result, the resource-pooled system forms a lower bound that we can compare SRPT-k against. Specifically, we compare against the SRPT policy for the resource-pooled system, which we refer to as SRPT-1. We depict the SRPT-k and SRPT-1 systems in Fig. 2.2.

2.2 Technical Introduction

The Shortest Remaining Processing Time (SRPT) scheduling policy and variants thereof have been deployed in many computer systems, including web servers [110], networks [149], databases [89], operating systems [32] and FPGA layout systems [36]. SRPT has also long been a topic of fascination for queueing theorists due to its optimality properties. In 1966, the mean response time for SRPT was first derived [196], and in 1968 SRPT was shown to minimize mean response time both in a stochastic sense and in a worst-case sense [194]. However, these beautiful optimality results and the analysis of SRPT are only known for *single-server* systems. Almost nothing is known for *multiserver* systems, such as the M/G/k, even for the case of just $k = 2$ servers.

The SRPT policy for the M/G/k is defined as follows: at all times, the k jobs with smallest remaining processing time receive service, preempting jobs in service if necessary.

We assume a central queue, meaning any job can be dispatched or migrated to any server at any time, and a preempt-resume model, meaning preemption incurs no cost or loss of work.

It seems believable that SRPT should minimize mean response time in multiserver systems because it gives priority to the jobs which will finish soonest, which seems like it should minimize the number of jobs in the system. However, it was shown in 1997 that SRPT is not optimal for multiserver systems in the worst case [135, 136]. That is, one can come up with an adversarial arrival sequence for which the mean response time under SRPT is larger that the optimal mean response time. In fact, the ratio by which SRPT's mean response time exceeds the optimal mean response time can be arbitrarily large [135, 136].

The fact that multiserver SRPT is not optimal in the worst case provokes a natural question about the *stochastic* case.

Is SRPT optimal or near-optimal for minimizing mean response time in the M/G/k?

Unfortunately, this question is entirely open. Not only is it not known whether SRPT is optimal, but multiserver SRPT has also eluded stochastic analysis.

What is the mean response time for the M/G/k under SRPT?

The purpose of this chapter is to answer both of these questions in the high-load setting. Under low load, response time is dominated by service time, which is not affected by the scheduling policy. In contrast, under high load, response time is dominated by queueing time, which can vary wildly under different scheduling policies. We thus focus on the high-load setting, and specifically on the *heavy-traffic limit* as load approaches capacity.

Our main result is that, under mild assumptions on the service requirement distribution,

> *SRPT is an optimal multiserver policy for minimizing mean response time in the M/G/k in the heavy-traffic limit.*

We also give the *first mean response time bound for the M/G/k under SRPT*. The bound is valid for all loads and is tight for load near capacity.

In addition to SRPT, we give the *first mean response time bounds for the M/G/k with three other scheduling policies*, specifically Preemptive Shortest Job First (PSJF) [231], Remaining Size Times Original Size (RS) [117, 233], and Foreground-Background (FB) [171]. Our bounds imply that in the heavy-traffic limit, under the same mild assumptions as for SRPT above,

- multiserver PSJF and RS are also optimal multiserver scheduling policies; and
- multiserver FB is optimal in the same setting where single-server FB is optimal [185],

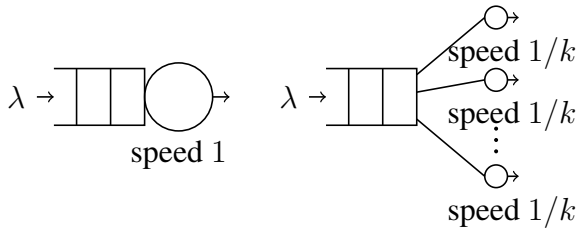


Figure 2.3: Single-server and k-server systems.

which is when the service requirement distribution has decreasing hazard rate and the scheduler does not have access to job sizes.

Our approach to analyzing SRPT on k servers is to compare its performance to that of SRPT on a single server which is k times as fast, where both systems have the same arrival rate λ and service requirement distribution S. Specifically, let *SRPT-k* be the policy which uses multiserver SRPT on k servers of speed $1/k$, as shown in Figure 2.3. Ordinary SRPT on a single server is simply SRPT-1. The *system load* $\rho = \lambda \mathbb{E}[S]$ is the average rate at which work enters the system. The maximal total rate at which the k servers can do work is 1, so the system is stable for $\rho < 1$, which we assume throughout.

Our main result is that in the $\rho \to 1$ limit, the mean response time under SRPT-k, $\mathbb{E}\left[T^{\text{SRPT-}k}\right]$, approaches the mean response time under SRPT-1, $\mathbb{E}\left[T^{\text{SRPT-1}}\right]$. Because SRPT-1 minimizes response time among all scheduling policies, this means that SRPT-k is asymptotically optimal among k-server policies. In particular, let OPT-k be the optimal k-server policy. Then

$$\mathbb{E}\left[T^{\text{SRPT-1}}\right] \leq \mathbb{E}\left[T^{\text{OPT-}k}\right] \leq \mathbb{E}\left[T^{\text{SRPT-}k}\right],$$

so showing that $\mathbb{E}\left[T^{\text{SRPT-}k}\right]$ approaches $\mathbb{E}\left[T^{\text{SRPT-1}}\right]$ as $\rho \to 1$ also shows that $\mathbb{E}\left[T^{\text{SRPT-}k}\right]$ approaches $\mathbb{E}\left[T^{\text{OPT-}k}\right]$ as $\rho \to 1$.

Specifically, we prove the following sequence of theorems.

Our first theorem is an upper bound on the mean response time of a job of size x under SRPT-k, written $\mathbb{E}\left[T^{\text{SRPT-}k}(x)\right]$. As in the classic SRPT-1 analysis [196], the response time of a job of size x depends on the system load contributed by jobs of size at most x, written $\rho_{\leq x}$ (see Definition 2.5.3).

Theorem 2.6.2. In an M/G/k, the mean response time of a job of size x under SRPT-k is bounded by

$$\mathbb{E}\left[T^{\text{SRPT-}k}(x)\right] \leq \frac{\int_0^x \lambda t^2 f_S(t)\, dt}{2(1-\rho_{\leq x})^2} + \frac{k\rho_{\leq x} x}{1-\rho_{\leq x}} + \int_0^x \frac{k}{1-\rho_{\leq t}}\, dt,$$

where $f_S(\cdot)$ is the probability density function of the service requirement distribution S.

The bound given in Theorem 2.6.2 holds for any load ρ and any service requirement distribution S. We use this bound to prove that, under mild conditions on S, the performance of SRPT-k approaches that of SRPT-1 in the $\rho \to 1$ limit, which implies asymptotic optimality of SRPT-k.

Theorem 2.7.1. In an M/G/k with any service requirement distribution S such that $\mathbb{E}[S^2(\log S)^+]$

is finite,

$$\lim_{\rho \to 1} \frac{\mathbb{E}\left[T^{\text{SRPT-}k}\right]}{\mathbb{E}\left[T^{\text{SRPT-1}}\right]} = 1.$$

The technique by which we bound response time under SRPT-k is widely generalizable. We also use it to give mean response time bounds and optimality results for PSJF-k, RS-k, and FB-k (see Section 2.8).

Our approach is inspired by two very different worlds: the stochastic world and the adversarial worst-case world. Purely stochastic approaches are difficult to generalize to the M/G/k for many reasons, including the fact that multiserver systems are not work-conserving. Purely adversarial worst-case analysis is easier but leads to weak bounds when directly applied to the stochastic setting. For instance, Leonardi and Raz [135, 136] show that for an adversarial arrival sequence, SRPT-k has worse mean response time than the optimal offline k-server policy by a factor of $\Omega(\log(\min(n/k, P))$, where n is the total number of jobs in the arrival sequence and P is the ratio of the largest job size to the smallest job size. This factor can be arbitrarily large in the context of the M/G/k, because $n \to \infty$ if the arrival sequence is an infinite Poisson process, and $P \to \infty$ if the service requirement distribution is unbounded or allows for arbitrarily small jobs.

What makes our analysis work is a strategic combination of the stochastic and worst-case techniques. We use the more powerful stochastic tools where possible and use worst-case techniques to bound variables for which exact stochastic analysis is intractable.

2.3 Prior Work

Countless papers have been published on the stochastic analysis of the SRPT policy in the single-server model over the last 52 years, beginning in 1966 with Schrage and Miller's response time analysis of the M/G/1 queue under SRPT [196], which was followed shortly by the proof of SRPT's optimality [194]. SRPT remains a major topic of study today. There have been beautiful works on analyzing the tail of response time [25–27], the fairness of SRPT [17, 231] and SRPT in different models, such as energy-aware control [70].

However, all of these works analyze *single-server* SRPT. We give the first analysis of multiserver SRPT. While single-server SRPT minimizes mean response time, multiserver SRPT does not[1] [135, 136]. We show that multiserver SRPT approaches optimality in heavy traffic.

2.3.1 Single-Server SRPT in Heavy Traffic

While the exact mean response time analysis of single-server SRPT is known, it is in the form of a triply nested integral. Therefore, it is useful to have a simpler formula for mean response time. Many papers have derived such a formula under heavy traffic [15, 16, 52, 138].

Heavy traffic analysis describes the behavior of a queueing system in the limit as load approaches capacity. The most general heavy-traffic analysis of the mean response time of single-

[1] It has been claimed that multiserver SRPT is optimal under the additional assumption that all servers are busy at all times [51, Theorem 2.1]. However, the proof has an error. See Section 2.3.5.

server SRPT is due to Lin et al. [138], who characterize the asymptotic behavior of mean response time for general service requirement distributions.

In particular, we show in Lemma 2.7.1 that their results imply that for any job size distribution S for which $\mathbb{E}\left[S^2(\log S)^+\right]$ is finite, the mean response time of single-server SRPT is $\omega(\log(1/(1-\rho)))$. We focus on the finite $\mathbb{E}\left[S^2(\log S)^+\right]$ setting, which is roughly equivalent to finite variance.

We build on the work of Lin et al. [138] to give the first heavy-traffic analysis of multiserver SRPT. In particular, we demonstrate that in the heavy-traffic limit, the mean response time of SRPT in a multiserver system with k servers approaches that of SRPT in a single-server system which runs k times faster (see Figure 2.3).

2.3.2 The Multiserver Priority Queue

While there is no existing stochastic analysis of multiserver SRPT, there is some analysis of multiserver priority queues. In a multiserver priority queue, it is assumed that there are finitely many classes of jobs (typically two) with exponential or phase-type service requirement distributions. Thus, the system can be modeled as a multidimensional Markov chain. Mitrani and King [154] give an exact analysis of the two class multiserver system with preemptive priority between the job classes and exponential service times within each class. Sleptchenko et al. [207] extend this analysis to hyperexponential service requirement distributions, and Harchol-Balter et al. [111] extend it further still to support phase-type service requirement distributions and any constant number of preemptive priority classes. However, the solutions found through these extensions can take a very long time to calculate, requiring more time with every added server, priority class, or state in the phase-type distribution.

Our analysis goes beyond the multiclass setting by handling an arbitrary service requirement distribution and a policy, namely SRPT-k, with an infinite set of priorities. Furthermore, our analysis produces a *closed-form* result, in contrast to the numerical results of these prior works.

2.3.3 Multiserver SRPT in the Worst Case

While stochastic analysis of multiserver SRPT is open, multiserver SRPT has been well studied in the worst-case setting. Worst-case analysis considers an *adversarially chosen* sequence of job arrival times and service requirements. An online policy (which does not know the arrival sequence) such as SRPT-k is typically compared to the optimal offline policy (which knows the arrival sequence). In the worst-case setting, a policy is a c-approximation if its mean response time is at most c times the mean response time of the offline optimal policy on any arrival sequence.

Leonardi and Raz [135, 136] analyze SRPT-k in the worst-case setting under the assumptions that (1) there are n jobs in the arrival sequence and (2) the ratio of the largest and smallest service requirements in the arrival sequence is P. They show that SRPT-k is an $O(\log(\min(n/k, P)))$-approximation for mean response time, where n is the total number of jobs. They also show that any online policy is at least an $\Omega(\log(\min(n/k, P)))$-approximation. This shows that no online policy has a better approximation ratio than SRPT-k by more than a constant factor.

Unfortunately, directly applying the $O(\log(\min(n/k, P)))$ bound on SRPT-k to the M/G/k is not helpful for two reasons. First, the arrival process is an infinite Poisson process, so $n \to \infty$. Second, often the maximum job size is unbounded or the minimum job size is arbitrarily small, so $P \to \infty$ as well.

SRPT has also been considered in other multiserver models. For example, Avrahami and Azar [12] analyze the immediate dispatch setting, in which each server has a queue and jobs are dispatched to these queues on arrival. Each server can only serve the jobs in its queue, and jobs cannot migrate between queues. Within each queue, jobs are served according to SRPT. Avrahami and Azar [12] give a dispatch policy called IMD which achieves the same $O(\log(\min(n/k, P)))$-approximation as SRPT-k, even when compared to the optimal offline policy with migrations. Again, directly applying this to the M/G/k is problematic because $n \to \infty$ and $P \to \infty$.

In contrast with these worst-case results, we show that in the stochastic setting, SRPT-k is asymptotically optimal policy for mean response time in the heavy-traffic limit. Our result holds for an extremely general class of service requirement distributions, including distributions which are unbounded and/or have arbitrarily small jobs.

2.3.4 Other Prior Work

Gong and Williamson [76] propose a *single-server* policy called K-SRPT which is superficially similar to our SRPT-k. Specifically, K-SRPT shares the processor between the k jobs in the system with least remaining time. That is, K-SRPT is a hybrid of processor sharing (PS) and SRPT. Crucially, when fewer than k jobs are in the system, K-SRPT allows each job to receive an increased share of the maximum service rate, ensuring work conservation. In contrast, our SRPT-k model never allows a job to receive more than $1/k$ of the maximum service rate of the system, since a job cannot run on more than one server at once. This means SRPT-k is not work-conserving, which makes it difficult to analyze.

2.3.5 Flawed Interchange Arguments

Down and Wu [51, Theorem 2.1] claim that SRPT-k is optimal in the sense of minimizing the completion time of the nth job for all n, under the additional assumption that all servers are busy at all times. Unfortunately, this claim is false. The proof attempts to use an interchange argument, mimicking the classic proof of the optimality of SRPT-1 [194]. However, the specified interchange can result in the same job running on two servers simultaneously, which is of course not possible.

A concrete counterexample is the following: let $k = 2$, and let jobs of size $1, 1, 2$ and 2 arrive at time 0. Recall that a job of size x must be in service for kx time to complete. SRPT-k completes its third job at time 6, while a policy which serves a job of size 2 over the interval $[0, 4)$ and jobs of size 1 over the intervals $[0, 2)$ and $[2, 4)$ would finish its third job at time 4. Moreover, more complicated counterexamples exist which show that multiserver SRPT does not minimize mean response time even if all servers are busy at all times.

A similar error occurs in a claim by Wu and Down [236, Theorem 2.1] that FB-k is optimal among policies that do not have access to job size information when the service requirement

distribution has decreasing hazard rate. Again the proof given is an interchange argument, and again the specified interchange can result in the same job running on two servers simultaneously.

2.4 Model

We study scheduling policies for the M/G/k queue. We write λ for the arrival rate, S for the service requirement distribution, and k for the number of servers. The rate at which any given server completes work is $1/k$. That is, a job with a service requirement, or *size*, of x needs to be served for time kx to complete. The k servers all together have total service rate 1.

The *load* of the M/G/k system, namely the average rate at which work arrives, is

$$\rho = \lambda \mathbb{E}[S].$$

That is, jobs arrive at rate λ jobs per second, each contributing $\mathbb{E}[S]$ work in expectation. We can view $\mathbb{E}[S] = 1/(k\mu)$, where $1/\mu$ is the expected amount of time a job needs to be served to complete. We assume a stable system, meaning $\rho < 1$, and a preempt-resume model, meaning that preemption incurs no cost or loss of work.

We will analyze systems in the *heavy-traffic limit*, which is the limit as $\rho \to 1$. More precisely, this is the limit as $\lambda \to 1/\mathbb{E}[S]$ for fixed S.

We analyze and compare systems with $k = 1$ and general k. An example of each is shown in Figure 2.3. Note that in our model, the M/G/1 and M/G/k systems have the same load ρ.

The primary policy we study is the SRPT-k policy, which is the Shortest Remaining Processing Time policy on k servers. At every moment in time, SRPT-k serves the k jobs with smallest remaining processing time. If there are fewer than k jobs in the system, every job receives service, which leaves some servers idle. Note SRPT-1 is the usual single-server SRPT policy.

2.5 Background and Challenges

Our approach to analyzing response time under SRPT-k is to compare it with SRPT-1. As such, we begin this section by briefly reviewing the analysis of SRPT-1, specifically focusing on the definitions and formulations that will come up in the SRPT-k analysis. We then outline why the SRPT-1 analysis does not easily generalize to SRPT-k with $k > 1$ servers.

2.5.1 SRPT-1 Tagged Job Tutorial

We now review the technique used by Schrage and Miller [196] to analyze SRPT-1. Consider a particular "tagged" job j, of size x, arriving to a *random system state* drawn from the system's steady-state distribution. We denote j's response time by $T^{\text{SRPT-1}}(x)$. Of course, $T^{\text{SRPT-1}}(x)$ is a random variable which depends on both the random arrivals that occur after j and the random queue state that j observes upon its own arrival.

We split the analysis of $T^{\text{SRPT-1}}(x)$ into two parts, shown in Figure 2.4:

- *waiting time* $W^{\text{SRPT-1}}(x)$, the time between j's arrival and the moment j first enters service; and

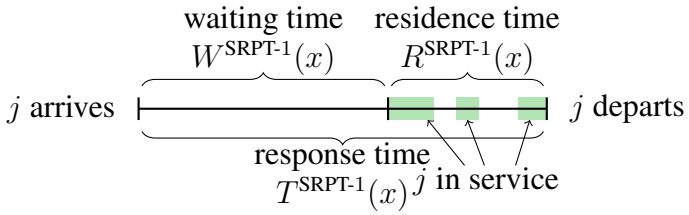


Figure 2.4: Response time of the tagged job j of size x is the sum of waiting time and residence time.

- *residence time* $R^{\text{SRPT-1}}(x)$, the time between the moment j first enters service and j's departure.

Given waiting time and residence time, response time is simply

$$T^{\text{SRPT-1}}(x) = W^{\text{SRPT-1}}(x) + R^{\text{SRPT-1}}(x).$$

Under SRPT-1, j has priority over all jobs with larger remaining size than itself, so such jobs do not impact j's response time.

Definition 2.5.1. *Suppose job j has remaining size x. A job ℓ is* relevant *to job j if ℓ has remaining size at most x. Otherwise ℓ is* irrelevant *to j.*

In particular, we will often consider which jobs are relevant to the tagged job j. We will simply call jobs "relevant" and "irrelevant" when the comparison is clear from context. For the purpose of analyzing j's response time, we can ignore all jobs which are irrelevant to j.

During j's waiting time, the server is only doing *relevant work*, namely work that is due to a relevant job. The total amount of work done is the sum of

- relevant work due to relevant jobs that were in the system when j arrived and
- relevant work due to relevant jobs that arrived after j.

To analyze j's waiting time, we make use of a concept called a "busy period".

Definition 2.5.2. *A* busy period *started by (possibly random) amount of work V, written $B(V)$, is the amount of time it takes for a single-server work-conserving system that starts with V work to become empty.*

Busy periods are very useful because their length depends only on the initial amount of work and the arrival process, not on the service policy or the number of jobs in the system.

In the SRPT-1 system, we do not have to wait for the system to become completely empty for j to start receiving service. We only have to wait for the system to become empty of relevant work. We capture this with the concept of a "relevant busy period".

Definition 2.5.3. *A* relevant busy period *for a job of size x started by (possibly random) amount of work V, written $B_{\leq x}(V)$, is the amount of time it takes for a work-conserving system that starts with V work to become empty, where only arrivals of size at most x, the relevant arrivals, are admitted to the system. A relevant busy period has expectation*

$$\mathbb{E}\left[B_{\leq x}(V)\right] = \frac{\mathbb{E}\left[V\right]}{1 - \rho_{\leq x}}.$$

Above, $\rho_{\leq x}$ is the relevant load *for a job of size x, which is the total load due to relevant jobs. Its value is*

$$\rho_{\leq x} = \lambda \mathbb{E}\left[S \mathbb{1}(S \leq x)\right],$$

where $\mathbb{1}(\cdot)$ is the indicator function.

This means j's waiting time is a relevant busy period started by the amount of relevant work that the tagged job j sees on arrival. By the PASTA property (Poisson Arrivals See Time Averages) [235], the distribution for the amount of relevant work j sees is the steady-state distribution.

Definition 2.5.4. *The* steady-state relevant work *for a job of size x under SRPT-1, written $\texttt{RelWork}^{\text{SRPT-1}}_{\leq x}$, is the sum of remaining sizes of all jobs with remaining size at most x observed at a random point in time. (An analogous definition applies to SRPT-k.)*

By the above discussion, j's waiting time is

$$W^{\text{SRPT-1}}(x) = B_{\leq x}\big(\texttt{RelWork}^{\text{SRPT-1}}_{\leq x}\big).$$

The analysis of $\texttt{RelWork}^{\text{SRPT-1}}_{\leq x}$ is known [196] but outside the scope of this tutorial.

The residence time of j can be analyzed in a similar way. At the start of j's residence time, the SRPT-1 policy serves j, so j, which has remaining x, must be the job with the smallest remaining size in the system. This means the system is effectively empty from j's perspective, because all work relevant to j is gone.

The only work that will be done from this point until j completes is work on j itself and relevant arrivals. Because j's residence time starts with its own work x and ends when that work is done, we can stochastically upper bound j's residence time as a relevant busy period:

$$R^{\text{SRPT-1}}(x) \leq_{\text{st}} B_{\leq x}(x).$$

The reason this bound is not tight is because j's remaining size decreases during service, which changes the cutoff for relevant jobs. An exact analysis of $R^{\text{SRPT-1}}(x)$ is known [196] but outside the scope of this tutorial.

2.5.2 Why the Tagged Job Analysis is Hard for SRPT-k

Having summarized the analysis of SRPT-1, it is natural to ask: why does a similar strategy not work for SRPT-k? The primary difficulty is that multiserver systems are *not work-conserving*, which manifests in two ways.

First, analyzing *busy periods* relies on work conservation, namely the fact that the server is doing work at rate 1 whenever the system is not empty. This allows for many simplifications. For instance, in Definition 2.5.3, we define busy periods as being started as a total amount of work, without worrying exactly how that work is divided among jobs. In a k-server system, work is only done at rate 1 if there are k or more jobs in the system. Thus, the exact rate at which work is done varies over time depending on the number of jobs in the system, making it difficult to analyze.

Second, analyzing the *steady-state relevant work* relies on work conservation. The analysis of $\texttt{RelWork}^{\text{SRPT-1}}_{\leq x}$ by Schrage and Miller [196] relies on being able to equate $\texttt{RelWork}^{\text{SRPT-1}}_{\leq x}$ to the total work in a simpler first-come-first-served system. Equality of remaining work only holds if both systems are work-conserving. The fact that SRPT-k is not work-conserving means that we can't make such an argument.

2.6 Analysis of SRPT-k

As explained in Section 2.5.2, traditional tagged job analysis cannot be applied to SRPT-k because SRPT-k is not work-conserving. Our approach is to find a way to make SRPT-k appear work-conserving while the tagged job j is in the system. We do this by introducing the new concept of *virtual work*. Virtual work encapsulates all of the time that the servers spend either idle or working on irrelevant jobs while j is in the system. By thinking of these times as "virtual work", the system appears to be work-conserving while j is in the system, allowing us to bound the response time of j.

Consider a tagged job j of size x. Recall from Definition 2.5.1 that only jobs of remaining size at most x are *relevant* to j when j arrives. We will bound j's response time by bounding the *total amount of server activity* between j's arrival and departure. Between j's arrival and departure, each server can be doing one of four categories of work.

- *Tagged work:* serving j.
- *Old work:* serving a job which is relevant to j that was in the system upon j's arrival.
- *New work:* serving a job which is relevant to j that arrived after j.
- *Virtual work:* either idling or serving an job which is irrelevant to j.

The response time of j is exactly the total of tagged, old, new, and virtual work. The main idea behind our analysis is to bound this total by a single (work-conserving) relevant busy period (see Definition 2.5.3). The busy period is still defined with regards to a single server system, making the analysis straightforward.

We already know a few facts about the four categories of work.

- Tagged work is j's size x.
- Old work is equal to the amount of relevant work seen by j upon arrival.[2] By the PASTA property [235], this is $\texttt{RelWork}^{\text{SRPT-}k}_{\leq x}$, the steady state amount of relevant work for a job of size x (see Definition 2.5.4).
- New work is bounded by all jobs which are relevant to a job of remaining size x that arrive during a relevant busy period $B_{\leq x}(\cdot)$ started by tagged, old, and virtual work.[3] This is only an upper bound because we ignore the fact that j's remaining size decreases as j is served, which changes the size cutoff for relevant jobs.
- Virtual work is as of yet unknown. We denote with the random variable $\texttt{VirtWork}^{\text{SRPT-}k}(x)$ the amount of virtual work done while j is in the system.

Taken together, these yield the bound

$$T^{\text{SRPT-}k}(x) \leq_{\text{st}} B_{\leq x}\Big(x + \texttt{RelWork}^{\text{SRPT-}k}_{\leq x} + \texttt{VirtWork}^{\text{SRPT-}k}(x)\Big). \tag{2.1}$$

Our task in the remainder of this section is to bound $\texttt{RelWork}^{\text{SRPT-}k}_{\leq x}$ and $\texttt{VirtWork}^{\text{SRPT-}k}(x)$ as tightly as we can. We use worst-case methods to bound $\texttt{VirtWork}^{\text{SRPT-}k}(x)$ and a combination of stochastic and worst-case methods to bound $\texttt{RelWork}^{\text{SRPT-}k}_{\leq x}$.

[2] One might worry that an *old job* that is irrelevant when j arrives could later become relevant to j, and therefore be part of old work, but this does not occur under SRPT-k.

[3] One might worry that a *new job* that is irrelevant when it arrives could later become relevant to j, and therefore be part of new work, but this does not occur under SRPT-k.

2.6.1 Virtual Work

We start by bounding $\texttt{VirtWork}^{\text{SRPT-}k}(x)$, the virtual work done while j is in the system. A purely stochastic analysis of virtual work would be very difficult. Fortunately, a simple worst-case bound suffices for our purposes. The key is that a server can do virtual work *only while j is in service at a different server*. This is because SRPT-k never allows an irrelevant job to have priority over j.

Lemma 2.6.1. *The virtual work is bounded by*

$$\texttt{VirtWork}^{\text{SRPT-}k}(x) \leq (k-1)x.$$

Proof. Virtual work only occurs while j is in service. The maximum possible virtual work is achieved by all $k-1$ other servers doing virtual work whenever j is in service. Each server does work at rate $1/k$. This means j is in service for time kx, during which virtual work is done at rate at most $(k-1)/k$. □

2.6.2 Relevant Work

Our next task is to bound $\texttt{RelWork}_{\leq x}^{\text{SRPT-}k}$, the steady state amount of relevant work for a job of size x under SRPT-k. As with virtual work, a purely stochastic analysis of relevant work would be very difficult. We therefore take the following hybrid approach. We consider a pair of systems, one using SRPT-1 and the other using SRPT-k, experiencing the same arrival sequence. We compare the amounts of relevant work in each system, giving a *worst-case bound for the difference*. This allows us to use the previously known *stochastic analysis* of $\texttt{RelWork}_{\leq x}^{\text{SRPT-1}}$ to give a stochastic bound for $\texttt{RelWork}_{\leq x}^{\text{SRPT-}k}$.

Consider running a pair of systems under the same job arrival sequence:

- *System* 1, which schedules using SRPT-1; and
- *System* k, which schedules using SRPT-k.

For any time t, let $\texttt{RelWork}_{\leq x}^{(1)}(t)$ be the amount of relevant work in System 1 at t, and similarly for $\texttt{RelWork}_{\leq x}^{(k)}(t)$. Our goal is to give a worst-case bound for the difference in relevant work between Systems 1 and k,

$$\Delta_{\leq x}(t) = \texttt{RelWork}_{\leq x}^{(k)}(t) - \texttt{RelWork}_{\leq x}^{(1)}(t).$$

To bound $\Delta_{\leq x}(t)$, we split times t into two types of intervals:

- *few-jobs intervals*, during which there are fewer than k relevant jobs at a time in System k; and
- *many-jobs intervals*, during which there are at least k relevant jobs at a time in System k.

A similar type of splitting was used by Leonardi and Raz [135, 136].

As a reminder, a job is *relevant* if its remaining size is at most x and *irrelevant* otherwise (see Definition 2.5.1). Note that many-jobs intervals are defined only in terms of System k, so System 1 may or may not have relevant jobs during a many-jobs interval.

Lemma 2.6.2. *For any arrival sequence and at any time t, the difference between the relevant work in System 1 and the relevant work in System k is bounded by*

$$\Delta_{\leq x}(t) \leq kx.$$

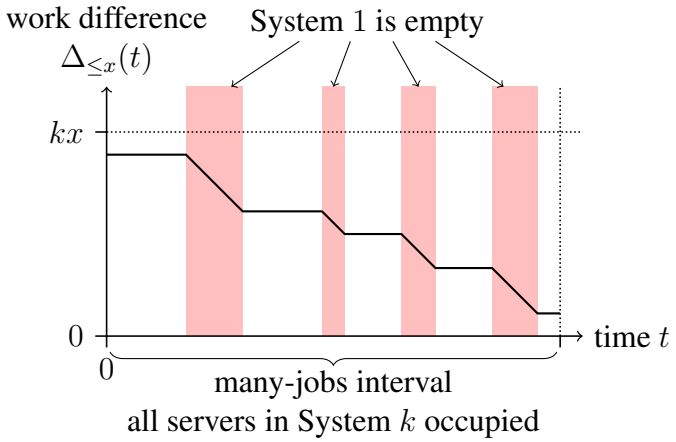


Figure 2.5: Relevant work difference is nonincreasing during many-jobs intervals.

Lemma 2.6.2 shows that $\Delta_{\le x}(t)$ is bounded at all times. We can summarize the proof of Lemma 2.6.2 as follows. In a few-jobs interval, $\Delta_{\le x}(t)$ is bounded because there are few relevant jobs in System 1 and each contributes a bounded amount of relevant work. In a many-jobs interval, $\Delta_{\le x}(t)$ is nonincreasing, and hence bounded.

One might intuitively expect $\Delta_{\le x}(t)$ to be constant during a many-jobs interval. However, $\Delta_{\le x}(t)$ can decrease during a many-jobs interval, namely when System 1 is empty, as shown in Figure 2.5.

Since the few-jobs and many-jobs intervals cover all possible times, Lemma 2.6.2 always holds.

Proof of Lemma 2.6.2. Any time t is in either a few-jobs interval or a many-jobs interval. The case where t is in a few-jobs interval is simple: there are at most $k-1$ relevant jobs in System k at time t, each of remaining size at most x, so

$$\Delta_{\le x}(t) \le \texttt{RelWork}^{(k)}_{\le x}(t) \le (k-1)x.$$

Suppose instead that t is in a many-jobs interval. Let time s be the start of the many-jobs interval containing t. We will show

$$\Delta_{\le x}(t) \le \Delta_{\le x}(s) \le kx.$$

We first show that $\Delta_{\le x}(t) \le \Delta_{\le x}(s)$. Let

$$D^{(1)} = \texttt{RelWork}^{(1)}_{\le x}(t) - \texttt{RelWork}^{(1)}_{\le x}(s)$$
$$D^{(k)} = \texttt{RelWork}^{(k)}_{\le x}(t) - \texttt{RelWork}^{(k)}_{\le x}(s)$$

be the change in relevant work from s to t in Systems 1 and k, respectively. Because

$$\Delta_{\le x}(t) - \Delta_{\le x}(s) = D^{(k)} - D^{(1)},$$

it suffices to show $D^{(k)} \le D^{(1)}$.

We can write $D^{(1)}$ as a sum of three components,

$$D^{(1)} = \texttt{Arrivals}^{(1)} + \texttt{NewlyRelevant}^{(1)} - \texttt{Served}^{(1)},$$

which are defined as follows.

- $\texttt{Arrivals}^{(1)}$ is the relevant work added during $[s,t]$ due to relevant new arrivals.
- $\texttt{NewlyRelevant}^{(1)}$ is the relevant work added during $[s,t]$ due to the server serving irrelevant jobs until they reach remaining size x, at which point they become relevant. For our purposes, all that matters is that $\texttt{NewlyRelevant}^{(1)} \geq 0$.
- $\texttt{Served}^{(1)}$ is the amount of relevant work done by the server during $[s,t]$. System 1 does relevant work at rate 1 if it has any relevant jobs and rate 0 otherwise, so $\texttt{Served}^{(1)} \leq t-s$.

We define analogous quantities for System k and compare them to their System 1 counterparts.

- $\texttt{Arrivals}^{(k)} = \texttt{Arrivals}^{(1)}$ because the two systems experience the same arrivals.
- $\texttt{NewlyRelevant}^{(k)} = 0$ because $[s,t]$ is within a many-jobs interval, during which System k has at least k relevant jobs. Therefore, there is never an opportunity for an irrelevant job to be served and become relevant. In particular,

$$\texttt{NewlyRelevant}^{(k)} \leq \texttt{NewlyRelevant}^{(1)}.$$

- $\texttt{Served}^{(k)} = t-s$ because $[s,t]$ is within a many-jobs interval, during which System k has at least k relevant jobs. Therefore, its servers do relevant work at combined rate 1 during all of $[s,t]$. In particular,

$$\texttt{Served}^{(k)} \geq \texttt{Served}^{(1)}.$$

The three comparisons above imply $D^{(k)} \leq D^{(1)}$, as desired.

All that remains is to show $\Delta_{\leq x}(s) \leq kx$. Recall that s is the start of a many-jobs interval. There are two ways to enter a many-jobs interval. In both cases, we show that $\Delta_{\leq x}(s) \leq kx$.

One way a many-jobs interval can start is when *a relevant job arrives while System k has $k-1$ relevant jobs*. The same arrival occurs in System 1, so $\Delta_{\leq x}(s) = \Delta_{\leq x}(s^-)$, where s^- is the instant before the arrival. But s^- is the end of a few-jobs interval, during which System k has at most $k-1$ relevant jobs, so

$$\Delta_{\leq x}(s) = \Delta_{\leq x}(s^-) \leq \texttt{RelWork}^{(k)}_{\leq x}(s^-) \leq (k-1)x.$$

The other way a many-jobs interval can start is for *irrelevant jobs already in System k to become relevant*. For this to happen, System k must be serving $i \geq 1$ irrelevant jobs at s^-. Because relevant jobs have priority over irrelevant jobs, all relevant jobs must also be in service at s^-. There are i irrelevant jobs in service at s^-, so there are at most $k-i$ relevant jobs at s^-. At time s, at most i irrelevant jobs become relevant, so there are at most k relevant jobs at s. Each relevant job has size at most x, so

$$\Delta_{\leq x}(s) \leq \texttt{RelWork}^{(k)}_{\leq x}(s) \leq kx. \qquad \square$$

2.6.3 Response Time Bound

Theorem 2.6.1. *In an M/G/k, the response time of a job of size x under SRPT-k is bounded by*

$$T^{\text{SRPT-}k}(x) \leq_{\text{st}} W^{\text{SRPT-1}}(x) + B_{\leq x}(2kx),$$

where $W^{\text{SRPT-1}}(x)$ denotes the waiting time of a job of size x under SRPT-1.

Proof. From (2.1), we know that

$$T^{\text{SRPT-}k}(x) \leq_{\text{st}} B_{\leq x}\Big(x + \texttt{RelWork}^{\text{SRPT-}k}_{\leq x} + \texttt{VirtWork}^{\text{SRPT-}k}(x)\Big).$$

By plugging in Lemmas 2.6.1 and 2.6.2, we find that

$$\begin{aligned} T^{\text{SRPT-}k}(x) &\leq_{\text{st}} B_{\leq x}\big(\texttt{RelWork}^{\text{SRPT-1}}_{\leq x} + 2kx\big) \\ &= B_{\leq x}\big(\texttt{RelWork}^{\text{SRPT-1}}_{\leq x}\big) + B_{\leq x}(2kx). \end{aligned}$$

Recall from Section 2.5.1 that the waiting time in SRPT-1 is

$$W^{\text{SRPT-1}}(x) = B_{\leq x}\big(\texttt{RelWork}^{\text{SRPT-1}}_{\leq x}\big),$$

giving the desired bound. □

While Theorem 2.6.1 gives a good bound on the response time under SRPT-k, we can tighten the bound further by making use of three ideas.

- As the tagged job j is served, its remaining size decreases. This decreases the size cutoff for new arrivals to be relevant, so not as many arriving jobs contribute to new work. Our current bounds do not account for this effect.
- In Lemma 2.6.2, we bound the difference $\Delta_{\leq x}(t)$ between relevant work in System 1, which uses SRPT-1, and relevant work in System k, which uses SRPT-k. It turns out that the same proof holds when System 1 uses PSJF-1, the preemptive shortest job first policy, instead of SRPT-1. This improves the bound because waiting time under PSJF-1 is smaller than waiting time under SRPT-1 [233].
- Even after replacing SRPT-1 with PSJF-1, Lemma 2.6.2 is not tight. In particular, $\Delta_{\leq x}(t)$ is at most x times the number of servers serving relevant jobs at time t, and there are not always k such servers.

These ideas allow us to prove the following bound on mean response time, which is strictly tighter than the one given by Theorem 2.6.1.

Theorem 2.6.2. *In an M/G/k, the mean response time of a job of size x under SRPT-k is bounded by*

$$\mathbb{E}\left[T^{\text{SRPT-}k}(x)\right] \leq \frac{\int_0^x \lambda t^2 f_S(t)\,dt}{2(1-\rho_{\leq x})^2} + \frac{k\rho_{\leq x}x}{1-\rho_{\leq x}} + \int_0^x \frac{k}{1-\rho_{\leq t}}\,dt,$$

where $f_S(\cdot)$ is the probability density function of the service requirement distribution S.

Proof. We will prove Theorem 2.6.2 by proving improved versions of (2.1) and Lemma 2.6.2.

A key element of our analysis is bounding the amount of new work done while the tagged job j of size x is in the system. In (2.1), we bound this quantity by a relevant busy period with size cutoff x. However, in reality, the size cutoff decreases as j receives service. We can use this to give a tighter bound on the amount of new work performed.

Let r_j be the amount of relevant work seen by j on arrival. Note that r_j is also the amount of old work that will be done while j is in the system.

Starting from the time of j's arrival, after at most $B_{\leq x}(r_j)$ time, j must enter service. During this busy period, an amount of work is performed equal to r_j plus all relevant arrivals during this busy period.

More generally, for any amount of time $s \leq x$, after at most a relevant busy period started by $r_j + ks$ work, j must have received s service. This holds because even if the servers finish all the old work and all the new work that has arrived so far, the servers must still complete ks combined tagged and virtual work. Of this tagged and virtual work, at least s must be tagged work, namely serving j. This means that the first dt service of j must be completed by time

$$B_{\leq x}(r_j) + B_{\leq x}(k \cdot dt).$$

The next dt service of j must be completed by time

$$B_{\leq x}(r_j) + B_{\leq x}(k \cdot dt) + B_{\leq x-dt}(k \cdot dt),$$

because the cutoff for entering the relevant busy period decreases as j receives service. Similarly, the following dt service of j must be completed by time

$$B_{\leq x}(r_j) + B_{\leq x}(k \cdot dt) + B_{\leq x-dt}(k \cdot dt) + B_{\leq x-2\,dt}(k \cdot dt).$$

This pattern continues as j receives service. The descending size cutoff yields the same sort of relevant busy period as in the traditional tagged job analysis of SRPT-1 [196]. Recalling that r_j is drawn from the distribution $\texttt{RelWork}_{\leq x}^{\text{SRPT-}k}$ yields the following bound on the mean response time of j:

$$T^{\text{SRPT-}k}(x) \leq B_{\leq x}\big(\texttt{RelWork}_{\leq x}^{\text{SRPT-}k}\big) + \int_0^x B_{\leq t}(k \cdot dt). \tag{2.2}$$

With (2.2), we have improved upon (2.1).

Next, we will improve upon Lemma 2.6.2. We consider a pair of systems experiencing the same arrival sequence: System 1, which uses PSJF-1, and System k, which uses SRPT-k.

Recall from Section 2.8.1 that under PSJF-1, a job ℓ is relevant to j if ℓ has *original* size at most x. In contrast, under SRPT-k, a job ℓ is relevant to j if ℓ has *remaining* size at most x.

We define $\Delta'_{\leq x}(t)$ to be the difference between the amounts of relevant work in the two systems at time t. Using Lemma 2.6.3 (proof deferred), we obtain a bound on $\Delta'_{\leq x}(t)$ tighter than the analogous bound in Lemma 2.6.2.

Lemma 2.6.3. *The difference in relevant work between Systems 1 and k is bounded by*

$$\Delta'_{\leq x}(t) \leq x \cdot \texttt{RelBusy}_{\leq x}^{(k)}(t)$$

where $\texttt{RelBusy}_{\leq x}^{(k)}(t)$ *is the number of servers in System k which are busy with relevant work at time t.*

Proof. We define few-jobs intervals and many-jobs intervals as in Section 2.6.2. Note that $\texttt{RelBusy}^{(k)}_{\leq x}(t) = k$ during a many-jobs interval, and that $\texttt{RelBusy}^{(k)}_{\leq x}(t)$ is the number of jobs in the system during a few-jobs interval.

The case where t is in a few-jobs interval is simple: there are exactly $\texttt{RelBusy}^{(k)}_{\leq x}(t)$ jobs in System k at time t, each of remaining size at most x, so

$$\Delta'_{\leq x}(t) \leq x \cdot \texttt{RelBusy}^{(k)}_{\leq x}(t).$$

Suppose instead that t is in a many-jobs interval, in which case $\texttt{RelBusy}^{(k)}_{\leq x}(t) = k$. Let time s be the start of the many-jobs interval containing t. Over the interval $[s, t]$, the same amount of relevant work arrives in both systems, because relevant arrivals are the same under SRPT and PSJF. Upon arrival a job's original and remaining sizes are equal. The other two categories of relevant work over the interval follow the same arguments as in the proof of Lemma 2.6.2. Thus,

$$\Delta'_{\leq x}(t) \leq \Delta'_{\leq x}(s).$$

It therefore suffices to show $\Delta'_{\leq x}(s) \leq kx$. As in Lemma 2.6.2, a many-jobs interval can begin due to the arrival of a relevant job, or due an irrelevant job in System k becoming relevant. In the case of an arrival, the same arrival occurs in System 1, and must be relevant in System 1, so

$$\Delta'_{\leq x}(s) = \Delta_{\leq x}(s^-) \leq (k-1)x,$$

because s^-, the instant before s, is in a few-jobs interval. In the case of an irrelevant job in System k becoming relevant, by the same argument as in the proof of Lemma 2.6.2,

$$\Delta'_{\leq x}(s) \leq \texttt{RelWork}^{(k)}_{\leq x}(s) \leq kx. \qquad \square$$

Continuing the proof of Theorem 2.6.2, we are now ready to prove the stronger bound. From (2.2), we know

$$T^{\text{SRPT-}k}(x) \leq B_{\leq x}\big(\texttt{RelWork}^{\text{SRPT-}k}_{\leq x}\big) + \int_0^x B_{\leq t}(k \cdot dt).$$

By plugging in Lemma 2.6.1 and Lemma 2.6.3, we find that

$$\begin{aligned}
&T^{\text{SRPT-}k}(x) \\
&\leq B_{\leq x}\big(\texttt{RelWork}^{\text{PSJF-1}}_{\leq x} + x \cdot \texttt{RelBusy}^{\text{SRPT-}k}_{\leq x}\big) + \int_0^x B_{\leq t}(k \cdot dt) \\
&= B_{\leq x}\big(\texttt{RelWork}^{\text{PSJF-1}}_{\leq x}\big) + B_{\leq x}\big(x \cdot \texttt{RelBusy}^{\text{SRPT-}k}_{\leq x}\big) + \int_0^x B_{\leq t}(k \cdot dt) \\
&= W^{\text{PSJF-1}}(x) + B_{\leq x}\big(x \cdot \texttt{RelBusy}^{\text{SRPT-}k}_{\leq x}\big) + \int_0^x B_{\leq t}(k \cdot dt),
\end{aligned}$$

where $\texttt{RelBusy}^{\text{SRPT-}k}_{\leq x}$ is the steady state number of servers which are busy with relevant jobs under SRPT-k. Taking expectations yields

$$\mathbb{E}\left[T^{\text{SRPT-}k}(x)\right] \leq \mathbb{E}\left[W^{\text{PSJF-1}}(x)\right] + \mathbb{E}\left[B_{\leq x}\big(x \cdot \texttt{RelBusy}^{\text{SRPT-}k}_{\leq x}\big)\right] + \int_0^x \mathbb{E}\left[B_{\leq t}(k \cdot dt)\right].$$

From the literature [233], we know that

$$\mathbb{E}\left[W^{\text{PSJF-1}}(x)\right] = \frac{\int_0^x \lambda t^2 f_S(t)\, dt}{2(1-\rho_{\le x})^2}.$$

By the expectation of a relevant busy period, from Definition 2.5.3,

$$\int_0^x \mathbb{E}\left[B_{\le t}(k \cdot dt)\right] = \int_0^x \frac{k}{1-\rho_{\le t}}\, dt.$$

Similarly,

$$\mathbb{E}\left[B_{\le x}\left(x \cdot \texttt{RelBusy}_{\le x}^{\text{SRPT-}k}\right)\right] = \frac{\mathbb{E}\left[x \cdot \texttt{RelBusy}_{\le x}^{\text{SRPT-}k}\right]}{1-\rho_{\le x}}.$$

The average rate at which the SRPT-k system performs relevant work is $\mathbb{E}\left[\texttt{RelBusy}_{\le x}^{\text{SRPT-}k}\right]/k$, since each busy server does work at rate $1/k$. Because the system is stable, the rate at which relevant work is done must equal the rate at which relevant work enters the system, namely $\rho_{\le x}$. Thus, $\mathbb{E}\left[\texttt{RelBusy}_{\le x}^{\text{SRPT-}k}\right] = k\rho_{\le x}$, so

$$\mathbb{E}\left[B_{\le x}\left(x \cdot \texttt{RelBusy}_{\le x}^{\text{SRPT-}k}\right)\right] = \frac{k\rho_{\le x}x}{1-\rho_{\le x}},$$

yielding the desired bound. □

Note that the first term of Theorem 2.6.2's upper bound is the mean waiting time of a job of size x under PSJF-1.

2.7 Optimality of SRPT-k in Heavy Traffic

With the bound derived in Theorem 2.6.1, we can prove our main result on the optimality of SRPT-k in the heavy-traffic limit. Theorem 2.7.1 will refer to $\mathbb{E}\left[T^{\text{SRPT-}k}\right]$, which is derived from Theorem 2.6.1 by taking the expectation over possible sizes x.

Theorem 2.7.1. *In an M/G/k with any service requirement distribution S such that $\mathbb{E}[S^2(\log S)^+]$ is finite,*

$$\lim_{\rho\to 1} \frac{\mathbb{E}\left[T^{\text{SRPT-}k}\right]}{\mathbb{E}\left[T^{\text{SRPT-1}}\right]} = 1.$$

To prove Theorem 2.7.1, we start with a result from the literature on the performance of SRPT-1 in the heavy-traffic limit [138].

Lemma 2.7.1. *In an M/G/1 with any service requirement distribution S such that $\mathbb{E}[S^2(\log S)^+]$ is finite,*

$$\lim_{\rho\to 1} \frac{\log\left(\frac{1}{1-\rho}\right)}{\mathbb{E}\left[T^{\text{SRPT-1}}\right]} = 0.$$

Proof. This proof was originally published in our 2020 paper [202, Theorem 1.3].

Lin et al. [138] analyze the mean response time of SRPT in the M/G/1. A result of theirs implies [138, Lemma 5]

$$\mathbb{E}[T_1] = \Omega\left(\frac{1}{(1-\rho)\,G^{-1}(\rho)}\right),$$

where $G(r) = \mathbb{E}[S\mathbb{1}S \le r]/\mathbb{E}[S]$. We therefore want to show

$$\frac{1}{G^{-1}(\rho)} = \omega\left((1-\rho)\log\frac{1}{1-\rho}\right),$$

which amounts to showing that in the $r \to \infty$ limit,

$$(1-G(r))\log\frac{1}{1-G(r)} = o\left(\frac{1}{r}\right).$$

It actually suffices to show $1 - G(r) = o(1/(r\log r))$, as then

$$(1-G(r))\log\frac{1}{1-G(r)} = o\left(\frac{\log r + \log\log r}{r\log r}\right) = o\left(\frac{1}{r}\right).$$

We now use the $\mathbb{E}[S^2(\log S)^+] < \infty$ assumption to prove $1 - G(r) = o(1/(r\log r))$. The first step is to express $1 - G(r)$ in terms of S and S_e:

$$1 - G(r) = \frac{\mathbb{E}[S\mathbb{1}\{S > r\}]}{\mathbb{E}[S]} = \frac{\mathbb{E}[(S-r)^+]}{\mathbb{E}[S]} = \frac{r}{\mathbb{E}[S]}\mathbb{P}\{S > r\} + \mathbb{P}\{S_e > r\}.$$

We show that both terms are $o(1/(r\log r))$. For the first term, finiteness of $\mathbb{E}[S^2(\log S)^+]$ implies

$$\begin{aligned} r^2(\log r)^+\mathbb{P}\{S > r\} &\le \mathbb{E}\left[S^2(\log S)^+\mathbb{1}\{S > r\}\right] \\ &= \mathbb{E}\left[S^2(\log S)^+\right] - \mathbb{E}\left[S^2(\log S)^+\mathbb{1}\{S \le r\}\right] = o(1). \end{aligned}$$

We can bound the second term similarly because

$$\begin{aligned} \mathbb{E}[S]\,\mathbb{E}\left[S_e(\log S_e)^+\right] &= \mathbb{E}\left[\mathbb{1}\{S > 1\}\int_1^S s\log s\,ds\right] \\ &= \mathbb{E}\left[\mathbb{1}\{S > 1\}\left(\frac{1}{2}S^2\log S - \frac{1}{4}S^2 + \frac{1}{4}\right)\right] < \infty. \end{aligned}$$

□

The next step in proving Theorem 2.7.1, is to use the bound on $T^{\text{SRPT-}k}(x)$ provided by Theorem 2.6.1. Let $H(x)$ be the bound on $\mathbb{E}\left[T^{\text{SRPT-}k}(x)\right]$,

$$H(x) = \mathbb{E}\left[W^{\text{SRPT-1}}(x) + B_{\le x}(2kx)\right]. \tag{2.3}$$

By taking the expectation of drawing size x from the service requirement distribution S, Theorem 2.6.1 implies $\mathbb{E}\left[T^{\text{SRPT-}k}\right] \le \mathbb{E}[H(S)]$. The following lemma shows that $\mathbb{E}[H(S)]$ approaches $\mathbb{E}\left[T^{\text{SRPT-1}}\right]$ in the heavy-traffic limit.

Lemma 2.7.2. *In an M/G/k with any service requirement distribution S such that $\mathbb{E}[S^2(\log S)^+]$ is finite,*

$$\lim_{\rho\to 1} \frac{\mathbb{E}\left[H(S)\right]}{\mathbb{E}\left[T^{\text{SRPT-1}}\right]} = 1.$$

Proof. We know $\mathbb{E}\left[H(S)\right] \geq \mathbb{E}\left[T^{\text{SRPT-}k}\right]$ by Theorem 2.6.1, and we know $\mathbb{E}\left[T^{\text{SRPT-}k}\right] \geq \mathbb{E}\left[T^{\text{SRPT-1}}\right]$ by optimality of SRPT-1, so

$$\frac{\mathbb{E}\left[H(S)\right]}{\mathbb{E}\left[T^{\text{SRPT-1}}\right]} \geq 1.$$

We thus only need to show

$$\lim_{\rho\to 1} \frac{\mathbb{E}\left[H(S)\right]}{\mathbb{E}\left[T^{\text{SRPT-1}}\right]} \leq 1.$$

Because $W^{\text{SRPT-1}} \leq T^{\text{SRPT-1}}$, by (2.3) it suffices to show

$$\lim_{\rho\to 1} \frac{\mathbb{E}\left[B_{\leq S}(2kS)\right]}{\mathbb{E}\left[T^{\text{SRPT-1}}\right]} = 0. \tag{2.4}$$

Applying standard results for busy periods [104],

$$\mathbb{E}\left[B_{\leq S}(2kS)\right] = 2k\mathbb{E}\left[\frac{S}{1-\rho_{\leq S}}\right] = 2k\int_0^\infty \frac{x f_S(x)}{1-\rho_{\leq x}}\,dx,$$

where $f_S(\cdot)$ is the probability density function of S. To compute the integral, we make a change of variables from x to $\rho_{\leq x}$ (see Definition 2.5.3), which uses the following facts:

$$\begin{aligned}
\rho_{\leq x} &= \lambda\mathbb{E}\left[S\mathbb{1}(S<x)\right] = \int_0^x \lambda t f_S(t)\,dt \\
\frac{d\rho_{\leq x}}{dx} &= \lambda x f_S(x) \\
\rho_{\leq 0} &= 0 \\
\lim_{x\to\infty} \rho_{\leq x} &= \rho.
\end{aligned}$$

Given this change of variables, we compute

$$\begin{aligned}
\mathbb{E}\left[\frac{S}{1-\rho_{\leq S}}\right] &= \int_0^\infty \frac{x f_S(x)}{1-\rho_{\leq x}}\,dx \\
&= \int_0^\rho \frac{1}{\lambda(1-\rho_{\leq x})}\,d\rho_{\leq x} \\
&= \frac{1}{\lambda}\ln\left(\frac{1}{1-\rho}\right) \\
&= \Theta\left(\log\left(\frac{1}{1-\rho}\right)\right).
\end{aligned}$$

This means $\mathbb{E}\left[B_{\leq S}(2kS)\right] = \Theta(\log(1/(1-\rho)))$, so (2.4) follows from Lemma 2.7.1. □

Armed with Theorem 2.6.1 and Lemma 2.7.2, we are now prepared to prove our main result, Theorem 2.7.1.

Proof of Theorem 2.7.1. Because SRPT-1 minimizes mean response time, it suffices to show that

$$\lim_{\rho\to 1} \frac{\mathbb{E}\left[T^{\text{SRPT-}k}\right]}{\mathbb{E}\left[T^{\text{SRPT-1}}\right]} \leq 1,$$

which follows immediately from Theorem 2.6.1 and Lemma 2.7.2. □

Theorem 2.7.1 and the optimality of SRPT-1 imply that SRPT-k is optimal in the heavy-traffic limit.

Corollary 2.7.1. *In an M/G/k with any service requirement distribution S such that $\mathbb{E}[S^2(\log S)^+]$ is finite,*

$$\limsup_{\rho\to 1} \frac{\mathbb{E}\left[T^{\text{SRPT-}k}\right]}{\mathbb{E}\left[T^{P}\right]} \leq 1$$

for any scheduling policy P.

Recall from (2.3) that Theorem 2.6.1 implies $\mathbb{E}\left[T^{\text{SRPT-}k}\right] \leq \mathbb{E}\left[H(S)\right]$. Similarly, letting

$$I(x) = \frac{\int_0^x \lambda t^2 f_S(t)\,dt}{2(1-\rho_{\leq x})^2} + \frac{k\rho_{\leq x}x}{1-\rho_{\leq x}} + \int_0^x \frac{k}{1-\rho_{\leq t}}\,dt,$$

Theorem 2.6.2 implies $\mathbb{E}\left[T^{\text{SRPT-}k}\right] \leq \mathbb{E}\left[I(S)\right]$. Lemma 2.7.2 and the optimality of SRPT-1 imply that these bounds on SRPT-k's mean response time are tight as $\rho \to 1$.

Corollary 2.7.2. *In an M/G/k with any service requirement distribution S such that $\mathbb{E}[S^2(\log S)^+]$ is finite,*

$$\lim_{\rho\to 1} \frac{\mathbb{E}\left[H(S)\right]}{\mathbb{E}\left[T^{\text{SRPT-}k}\right]} = \lim_{\rho\to 1} \frac{\mathbb{E}\left[I(S)\right]}{\mathbb{E}\left[T^{\text{SRPT-}k}\right]} = 1.$$

Proof. After applying Theorem 2.6.1, Lemma 2.7.2, and the optimality of SRPT-1, we know that

$$\lim_{\rho\to 1} \frac{\mathbb{E}\left[H(S)\right]}{\mathbb{E}\left[T^{\text{SRPT-}k}\right]} = 1.$$

All that remains is to show $I(x) \leq H(x)$. This holds because

$$\frac{\int_0^x \lambda t^2 f_S(t)\,dt}{2(1-\rho_{\leq x})^2} \leq \mathbb{E}\left[W^{\text{SRPT-1}}(x)\right]$$

by the standard analysis of $W^{\text{SRPT-1}}(x)$ [196], and

$$\frac{k\rho_{\leq x}x}{1-\rho_{\leq x}} + \int_0^x \frac{k}{1-\rho_{\leq t}}\,dt \leq \frac{2kx}{1-\rho_{\leq x}} = \mathbb{E}\left[B_{\leq x}(2kx)\right].$$ □

As an illustration of the optimality of SRPT-k, we plot the ratio $\mathbb{E}\left[T^{\text{SRPT-}k}\right]/\mathbb{E}\left[T^{\text{SRPT-1}}\right]$ in Figure 2.6. The solid orange lines show simulation results for this ratio. For the dashed blue lines, we used our analysis from Theorem 2.6.2 as an upper bound on $\mathbb{E}\left[T^{\text{SRPT-}k}\right]$, and divided by the known results for $\mathbb{E}\left[T^{\text{SRPT-1}}\right]$. The important feature to notice in Figure 2.6 is that as system load ρ approaches 1, both our analytic bound and the simulation converge to 1.

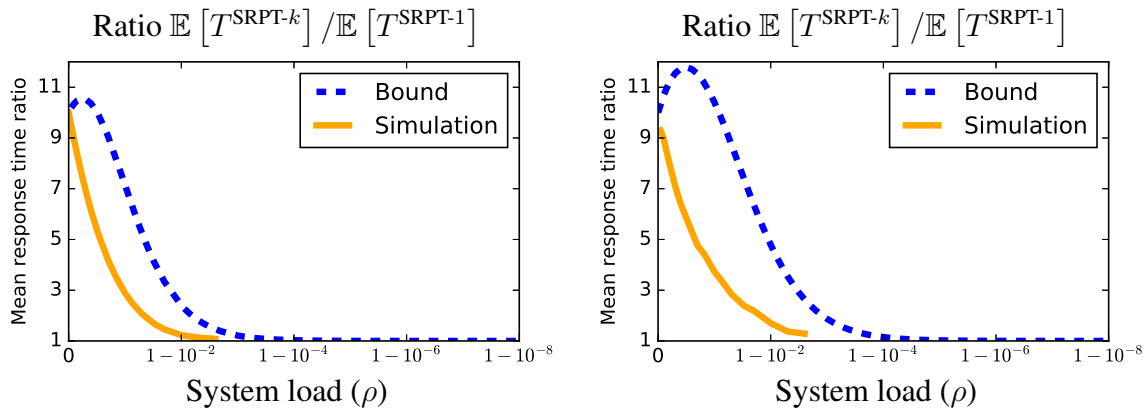


Figure 2.6: The plots above show the ratio $\mathbb{E}\left[T^{\text{SRPT-}k}\right]/\mathbb{E}\left[T^{\text{SRPT-1}}\right]$. Observe that as $\rho \to 1$, both our bound and the simulation converge to a ratio of 1. Our simulations of this ratio are the solid orange curves. Our analytic upper bounds derived in Theorem 2.6.2 are the dashed blue curves. We use $k = 10$ servers. The service requirement distribution S is $\text{Uniform}(0, 2)$ in the left plot and a Hyperexponential distribution with $\mathbb{E}[S] = 1$ and $C^2 = 10$ in the right plot. We only simulate up to $\rho = 0.9975$ due to long convergence times.

2.8 Other Scheduling Policies

We generalize our analysis to give the first response time bounds on several additional multiserver scheduling policies. Using the bounds, we prove optimality results for each policy as $\rho \to 1$. For each policy P, we generalize the usual M/G/1 policy, written P-1, to a multiserver policy for the M/G/k, written P-k, by preemptively serving the k jobs with highest priority at any time.

- *Preemptive Shortest Job First* (PSJF) prioritizes the jobs with smallest *original* size. PSJF-1 achieves performance comparable to SRPT despite not tracking every job's age [104].
- *Remaining Size Times Original Size* (RS) prioritizes the jobs with the smallest product of original size and remaining size. RS is also known as Size Processing Time Product (SPTP). RS-1 is optimal for minimizing *mean slowdown* [117].
- *Foreground-Background* (FB) prioritizes the jobs with smallest *age*, meaning the jobs that have been served the least so far. FB is also known as Least Attained Service (LAS). When the service requirement distribution has decreasing hazard rate, FB-1 minimizes mean response time among all scheduling policies that do not have access to job sizes [185].

We give the first response time bounds for PSJF-k, RS-k and FB-k. We then use these bounds to prove the following optimality results, under mild assumptions on the service requirement distribution:

- In the $\rho \to 1$ limit, PSJF-k and RS-k minimize mean response time among all scheduling policies (see Theorems 2.8.2 and 2.8.4).
- In the $\rho \to 1$ limit, FB-k minimizes mean response time under the same conditions as FB-1 (see Theorem 2.8.6).

Our analyses follow the same steps as in Section 2.6.

- Use the four categories of work to bound the response time of the tagged job j in terms of virtual work and steady-state relevant work.
- Bound virtual work.
- Bound steady-state relevant work.

Because different scheduling policies prioritize jobs differently, we use a different definition of "relevant jobs" for each policy. Under PSJF-k and RS-k, the definition of relevant jobs is very similar to that for SRPT-k, allowing us to use familiar tools such as relevant busy periods $B_{\leq x}(\cdot)$. However, FB-k uses a somewhat different definition of relevant jobs, resulting in a few changes to the analysis.

Finally, in Section 2.8.4, we discuss why our technique *does not* generalize to the First-Come, First-Served (FCFS) scheduling policy.

2.8.1 Preemptive Shortest Job First (PSJF-k)

As usual, we consider a tagged job j of size x. Under PSJF-k, another job ℓ is *relevant* to j if ℓ has *original* size at most x. With this definition of relevance, we divide work into the same four categories as in Section 2.6, namely tagged, old, new, and virtual. This bounds the response time of j by

$$T^{\text{PSJF-}k} \leq_{\text{st}} B_{\leq x}\Big(x + \texttt{RelWork}^{\text{PSJF-}k}_{\leq x} + \texttt{VirtWork}^{\text{PSJF-}k}(x)\Big). \tag{2.5}$$

The proof of Lemma 2.6.1 works nearly verbatim for PSJF-k, so

$$\texttt{VirtWork}^{\text{PSJF-}k}(x) \leq (k-1)x. \tag{2.6}$$

The analysis of steady-state relevant work is similar to that in Section 2.6.2. We consider a pair of systems experiencing the same arrival sequence: System 1, which uses PSJF-1, and System k, which uses PSJF-k. We define $\Delta^{\text{PSJF-}k}_{\leq x}(t)$ to be the difference between the amounts of relevant work in the two systems at time t. We then bound $\Delta^{\text{PSJF-}k}_{\leq x}(t)$.

Lemma 2.8.1. *The difference in relevant work between Systems* 1 *and* k *is bounded by*

$$\Delta^{\text{PSJF-}k}_{\leq x}(t) \leq (k-1)x.$$

Proof. We define few-jobs intervals and many-jobs intervals as in Section 2.6.2. The case where t is in a few-jobs interval is simple: there are at most $k-1$ relevant jobs in System k at time t, each of remaining size at most x, so

$$\Delta^{\text{PSJF-}k}_{\leq x}(t) \leq (k-1)x.$$

Suppose instead that t is in a many-jobs interval. Let time s be the start of the many-jobs interval containing t. By essentially the same argument as in the proof of Lemma 2.6.2,[4]

$$\Delta^{\text{PSJF-}k}_{\leq x}(t) \leq \Delta^{\text{PSJF-}k}_{\leq x}(s).$$

[4]In fact, the argument for PSJF is slightly simpler than that for SRPT, because irrelevant jobs never become relevant under PSJF.

It thus suffices to show $\Delta_{\leq x}^{\text{PSJF-}k}(s) \leq (k-1)x$. The only way a many-jobs interval can start under PSJF-k is for a relevant job to arrive while System k has $k-1$ relevant jobs. The same arrival occurs in System 1, so

$$\Delta_{\leq x}^{\text{PSJF-}k}(s) = \Delta_{\leq x}^{\text{PSJF-}k}(s^-) \leq (k-1)x$$

because s^-, the instant before s, is in a few-jobs interval. □

Theorem 2.8.1. *In an M/G/k, the response time of a job of size x under PSJF-k is bounded by*

$$T^{\text{PSJF-}k}(x) \leq_{\text{st}} W^{\text{PSJF-1}}(x) + B_{\leq x}((2k-1)x).$$

Proof. By (2.5), (2.6), and Lemma 2.8.1,

$$\begin{aligned} T^{\text{PSJF-}k}(x) &\leq_{\text{st}} B_{\leq x}\big(\texttt{RelWork}_{\leq x}^{\text{PSJF-1}} + (2k-1)x\big) \\ &= B_{\leq x}\big(\texttt{RelWork}_{\leq x}^{\text{PSJF-1}}\big) + B_{\leq x}((2k-1)x). \end{aligned}$$

The waiting time in PSJF-1 is

$$W^{\text{PSJF-1}}(x) = B_{\leq x}\big(\texttt{RelWork}_{\leq x}^{\text{PSJF-1}}\big),$$

giving the desired bound. □

With the bound derived in Theorem 2.8.1, we can prove that PSJF-k also minimizes mean response time in the heavy-traffic limit.

Theorem 2.8.2. *In an M/G/k with any service requirement distribution S such that $\mathbb{E}[S^2(\log S)^+]$ is finite,*

$$\lim_{\rho \to 1} \frac{\mathbb{E}\left[T^{\text{PSJF-}k}\right]}{\mathbb{E}\left[T^{\text{SRPT-1}}\right]} = 1.$$

Proof. From Theorem 2.8.1, we know that

$$T^{\text{PSJF-}k}(x) \leq_{\text{st}} W^{\text{PSJF-1}}(x) + B_{\leq x}((2k-1)x)$$

However, $W^{\text{PSJF-1}}(x) \leq_{\text{st}} W^{\text{SRPT-1}}(x)$ [233]. Therefore,

$$\begin{aligned} T^{\text{PSJF-}k}(x) &\leq_{\text{st}} W^{\text{SRPT-1}}(x) + B_{\leq x}((2k-1)x) \\ &\leq_{\text{st}} W^{\text{SRPT-1}}(x) + B_{\leq x}(2kx) \end{aligned}$$

This bound on $T^{\text{PSJF-}k}(x)$ is the same as the bound on $T^{\text{SRPT-}k}(x)$ given in Theorem 2.6.1. The rest of the proof proceeds as in the proof of Theorem 2.7.1. □

As in Corollary 2.7.1, Theorem 2.8.2 and the optimality of SRPT-1 imply that PSJF-k is optimal in the heavy-traffic limit.

2.8.2 Remaining Size Times Original Size (RS-k)

As usual, we consider a tagged job j of size x. When j has remaining size y, another job ℓ is *relevant* to j if the *product of ℓ's original size and remaining size* is at most xy. In particular, if ℓ is relevant to j, then ℓ's remaining size is at most x. With this definition of relevance, we divide work into the same four categories as in Section 2.6, namely tagged, old, new, and virtual. This bounds the response time of j by

$$T^{\text{RS-}k} \leq_{\text{st}} B_{\leq x}\Big(x + \texttt{RelWork}^{\text{RS-}k}_{\leq x} + \texttt{VirtWork}^{\text{RS-}k}(x)\Big). \tag{2.7}$$

The proof of Lemma 2.6.1 works nearly verbatim for RS-k, so

$$\texttt{VirtWork}^{\text{RS-}k}(x) \leq (k-1)x. \tag{2.8}$$

The analysis of steady-state relevant work is similar to that in Section 2.6.2. We consider a pair of systems experiencing the same arrival sequence: System 1, which uses RS-1, and System k, which uses RS-k. We define $\Delta^{\text{RS-}k}_{\leq x}(t)$ to be the difference between the amounts of relevant work in the two systems at time t. We then bound $\Delta^{\text{RS-}k}_{\leq x}(t)$.

Lemma 2.8.2. *The difference in relevant work between Systems* 1 *and* k *is bounded by*

$$\Delta^{\text{RS-}k}_{\leq x}(t) \leq kx.$$

Proof. Even though RS uses a definition of relevant jobs different from SRPT's, the proof is analogous to that of Lemma 2.6.2. □

Theorem 2.8.3. *In an M/G/k, the response time of a job of size x under RS-k is bounded by*

$$T^{\text{RS-}k}(x) \leq_{\text{st}} W^{\text{RS-1}}(x) + B_{\leq x}(2kx).$$

Proof. By (2.7), (2.8), and Lemma 2.8.2,

$$\begin{aligned} T^{\text{RS-}k}(x) &\leq_{\text{st}} B_{\leq x}\big(\texttt{RelWork}^{\text{RS-1}}_{\leq x} + 2kx\big) \\ &\leq_{\text{st}} B_{\leq x}\big(\texttt{RelWork}^{\text{RS-1}}_{\leq x}\big) + B_{\leq x}(2kx). \end{aligned}$$

The waiting time in RS-1 is

$$W^{\text{RS-1}}(x) = B_{\leq x}\big(\texttt{RelWork}^{\text{RS-1}}_{\leq x}\big),$$

giving the desired bound. □

With the bound derived in Theorem 2.8.3, we can prove that RS-k also minimizes mean response time in the heavy-traffic limit.

Theorem 2.8.4. *In an M/G/k with any service requirement distribution S such that $\mathbb{E}[S^2(\log S)^+]$ is finite,*

$$\lim_{\rho\to 1} \frac{\mathbb{E}\left[T^{\text{RS-}k}\right]}{\mathbb{E}\left[T^{\text{SRPT-1}}\right]} = 1.$$

Proof. From Theorem 2.8.3, we know that

$$T^{\text{RS-}k}(x) \leq_{\text{st}} W^{\text{RS-1}}(x) + B_{\leq x}(2kx)$$

However, $W^{\text{RS-1}}(x) \leq_{\text{st}} W^{\text{SRPT-1}}(x)$ [233]. Therefore,

$$T^{\text{RS-}k}(x) \leq_{\text{st}} W^{\text{SRPT-1}}(x) + B_{\leq x}(2kx)$$

This bound on $T^{\text{RS-}k}(x)$ is the same as the bound on $T^{\text{SRPT-}k}(x)$ given in Theorem 2.6.1. The rest of the proof proceeds as in the proof of Theorem 2.7.1. □

As in Corollary 2.7.1, Theorem 2.8.4 and the optimality of SRPT-1 imply that RS-k is optimal in the heavy-traffic limit.

We have so far shown response time bounds for SRPT-k, PSJF-k, and RS-k that are strong enough to prove asymptotic optimality in heavy traffic. We conjecture that similar bounds and optimality results hold for multiserver variants of any policy in the SMART class [233], which includes SRPT, PSJF, and RS.

2.8.3 Foreground-Background (FB-k)

The analysis of FB-k proceeds similarly to the analysis of SRPT-k but with a few more changes than were needed for PSJF-k and RS-k. To analyze PSJF-k and RS-k, we followed the same outline as Section 2.6 with a small change to the definition of relevant jobs. In particular, we reused the notion of relevant busy periods $B_{\leq x}(\cdot)$ from Definition 2.5.3. In contrast, as we will see shortly, FB-k has a significantly different definition of relevant jobs, so the definition of relevant busy periods will also change.

As usual, we consider a tagged job j of size x. Recall that FB prioritizes the jobs of smallest *age*, or attained service. When j arrives, its age is 0, so it has priority over all other jobs in the system. However, as j is served, its age increases and its priority gets worse. The key to the usual single-server analysis of FB is that to define relevant work, we have to look at j's *worst future priority* [104, 195, 201]. This worst priority occurs when j has age x, an instant before completion, giving us the following definition of relevant jobs.

Definition 2.8.1. *Suppose job j has original size x. Under FB-k, a job ℓ is* relevant *to job j if ℓ has age at most x. Otherwise ℓ is* irrelevant *to j.*

There is an important difference between the notions of relevance for SRPT-k and FB-k. Under SRPT-k, each arriving job starts as either relevant or irrelevant to j and remains that way for j's entire time in the system. In contrast, under FB-k, *every new arrival is at least temporarily relevant to j*. Specifically, if a new arrival ℓ has size at most x, then ℓ is relevant to j for its entire time in the system. If ℓ instead has size greater than x, then ℓ is relevant to j only until it reaches age x, at which point it becomes irrelevant. This observation motivates the definition of relevant busy periods for FB-k.

Definition 2.8.2. *Under FB-k, a* relevant busy period *for a job of size x started by (possibly random) amount of work V, written $B_{\bar{x}}(V)$, is the amount of time it takes for a work-conserving*

system that starts with V work to become empty, where every arrival's service is truncated at age x. *A relevant busy period has expectation*

$$\mathbb{E}\left[B_{\bar{x}}(V)\right] = \frac{\mathbb{E}\left[V\right]}{1-\rho_{\bar{x}}}.$$

Above, $\rho_{\bar{x}}$ is the relevant load *for a job of size x, which is the total load due to relevant jobs. Its value is*

$$\rho_{\bar{x}} = \lambda \mathbb{E}\left[\min(S, x)\right],$$

because each arrival is relevant only until it reaches age x.

We make a similar modification to the definition of steady-state relevant work.

Definition 2.8.3. *The* steady-state relevant work *for a job of size x under FB-k, written* $\texttt{RelWork}_{\bar{x}}^{\text{FB-}k}$, *is the sum of* remaining truncated sizes *of all jobs observed at a random point in time. A job's remaining truncated size is the amount of time until it either completes or reaches age x.*

Armed with Definitions 2.8.1, 2.8.2, and 2.8.3, we divide work into the same four categories as in Section 2.6, namely tagged, old, new, and virtual. This bounds the response time of j by

$$T^{\text{FB-}k} \leq_{\text{st}} B_{\bar{x}}\Big(x + \texttt{RelWork}_{\bar{x}}^{\text{FB-}k} + \texttt{VirtWork}^{\text{FB-}k}(x)\Big). \tag{2.9}$$

The proof of Lemma 2.6.1 works nearly verbatim for FB-k, so

$$\texttt{VirtWork}^{\text{FB-}k}(x) \leq (k-1)x. \tag{2.10}$$

The analysis of steady-state relevant work is similar to that in Section 2.6.2. We consider a pair of systems experiencing the same arrival sequence: System 1, which uses FB-1, and System k, which uses FB-k. We define $\Delta_{\bar{x}}^{\text{FB-}k}(t)$ to be the difference between the amounts of relevant work in the two systems at time t. We then bound $\Delta_{\bar{x}}^{\text{FB-}k}(t)$.

Lemma 2.8.3. *The difference in relevant work between Systems 1 and k is bounded by*

$$\Delta_{\bar{x}}^{\text{FB-}k}(t) \leq (k-1)x.$$

Proof. Even though FB uses a definition of relevant jobs different from PSJF's,[5] the proof is analogous to that of Lemma 2.8.1. □

Theorem 2.8.5. *In an M/G/k, the response time of a job of size x under FB-k is bounded by*

$$T^{\text{FB-}k}(x) \leq_{\text{st}} B_{\bar{x}}\big(\texttt{RelWork}_{\bar{x}}^{\text{FB-}k} + (2k-1)x\big).$$

Proof. Combining (2.9), (2.10), and Lemma 2.8.3 yields the desired bound. □

Note that the waiting time under FB-1 is always zero, as a new job immediately receives service, so we do not phrase the bound in terms of waiting time.

[5] We draw an analogy with PSJF rather than SRPT because under both FB and PSJF, irrelevant jobs never become relevant.

With the bound derived in Theorem 2.8.3, we can prove that the mean response time of FB-k approaches that of FB-1 in the heavy-traffic limit. We make use of prior work on the mean response time of FB in heavy traffic [121]. Let

$$W(x) = \mathbb{E}\left[B_{\bar{x}}(\texttt{RelWork}_{\bar{x}}^{\text{FB-}k})\right]$$
$$R(x) = \mathbb{E}\left[B_{\bar{x}}(x)\right].$$

$W(x)$ and $R(x)$ are not the mean waiting and residence times of a job of size x under FB because waiting time is always zero, but they play roughly analogous roles in the standard analysis of FB [201, Section 5].

Lemma 2.8.4. *In an M/G/1 with any service requirement distribution S which is unbounded with tail function of upper Matuszewska index less than -2,*

$$\lim_{\rho \to 1} \frac{\mathbb{E}\left[R(S)\right]}{\mathbb{E}\left[T^{\text{FB-1}}\right]} = 0.$$

Proof. Recall that

$$W(x) = \mathbb{E}\left[B_{\bar{x}}(\texttt{RelWork}_{\bar{x}}^{\text{FB-}k})\right]$$
$$R(x) = \mathbb{E}\left[B_{\bar{x}}(x)\right].$$

The standard analysis of FB-1 [104, 195] shows

$$\mathbb{E}\left[T^{\text{FB-1}}\right] = \mathbb{E}\left[W(S)\right] + \mathbb{E}\left[R(S)\right].$$

Kamphorst and Zwart [121, Equation (4.3)] decompose $\mathbb{E}\left[T^{\text{FB-1}}\right]$ into a sum of three functions of the load ρ,

$$\mathbb{E}\left[T^{\text{FB-1}}\right] = X(\rho) + Y(\rho) + Z(\rho),$$

such that

$$\mathbb{E}\left[W(S)\right] = Z(\rho) + \frac{1}{2}Y(\rho)$$
$$\mathbb{E}\left[R(S)\right] = X(\rho) + \frac{1}{2}Y(\rho).$$

Kamphorst and Zwart [121, Section 4.1.1] then show that

$$\lim_{\rho \to 1} \frac{X(\rho)}{Z(\rho)} = \lim_{\rho \to 1} \frac{Y(\rho)}{Z(\rho)} = 0,$$

which implies the desired limit

$$\lim_{\rho \to 1} \frac{\mathbb{E}\left[R(S)\right]}{\mathbb{E}\left[T^{\text{FB-1}}\right]} = \lim_{\rho \to 1} \frac{X(\rho) + \frac{1}{2}Y(\rho)}{X(\rho) + Y(\rho) + Z(\rho)} = 0.$$ □

Theorem 2.8.6. *In an M/G/k with any service requirement distribution S which is unbounded with tail function of upper Matuszewska index less than -2,*

$$\lim_{\rho \to 1} \frac{\mathbb{E}\left[T^{\text{FB-}k}\right]}{\mathbb{E}\left[T^{\text{FB-1}}\right]} = 1.$$

Proof. The standard analysis of FB-1 [104, 195] shows

$$\mathbb{E}\left[T^{\text{FB-1}}\right] = \mathbb{E}\left[W(S)\right] + \mathbb{E}\left[R(S)\right],$$

whereas Theorem 2.8.5 implies

$$\mathbb{E}\left[T^{\text{FB-}k}\right] \leq \mathbb{E}\left[W(S)\right] + (2k-1)\mathbb{E}\left[R(S)\right],$$

so the result follows by Lemma 2.8.4. □

Righter and Shanthikumar [185] show that when the job size distribution S has decreasing hazard rate, FB-1 is optimal for minimizing response time among all scheduling policies that do not have access to job sizes. Theorem 2.8.6 implies that in the heavy-traffic limit, FB-k is optimal in the same setting.[6]

Corollary 2.8.1. *In an M/G/k with any service requirement distribution S which (a) is unbounded, (b) has decreasing hazard rate, and (c) has tail function of upper Matuszewska index less than -2,*

$$\lim_{\rho \to 1} \frac{\mathbb{E}\left[T^{\text{FB-}k}\right]}{\mathbb{E}\left[T^{P}\right]} \leq 1$$

for any scheduling policy P that does not have access to job sizes.

2.8.4 What about First-Come, First-Served?

Having seen the success of our modified tagged job analysis for a variety of policies, it is natural to ask: does a similar analysis work for the multiserver First-Come, First-Served policy (FCFS-k)?

Unfortunately, our technique does not work for FCFS-k. To see why, let us take a look at what our analyses of SRPT-k, PSJF-k, RS-k, and FB-k have in common. A central component of all four analyses is bounding the *difference in relevant work* between two systems experiencing the same arrival sequence, one using a single-server policy P-1 and another using its k-server variant P-k. These bounds are given in Lemmas 2.6.2, 2.8.1, 2.8.2, and 2.8.3. All four lemmas have similar two-step proofs.

- First, they bound the *number of relevant jobs* both during few-jobs intervals and at the start of many-jobs intervals. For all four policies, this bound is at most k.
- Second, they bound the *relevant work contributed by each relevant job*. For all four policies, this bound is x.

When we try to prove analogous bounds for FCFS-k, we can still bound the number of relevant jobs by k, but *the relevant work contributed by each relevant job is unbounded*.

The definition of relevant jobs is the crucial difference between FCFS-k and the policies we analyze. Consider the jobs relevant to a tagged job j of size x.

- Under SRPT-k, PSJF-k, and RS-k, only *some* jobs are relevant to j, and all such jobs have size at most x.

[6] It has been claimed that FB-k is optimal for arbitrary arrival sequences when the service requirement distribution has decreasing hazard rate [236, Theorem 2.1]. However, the proof has an error. See Section 2.3.5.

- Under FB-k, while all jobs might be relevant to j, they are only *temporarily* relevant, each contributing at most x relevant work.
- However, under FCFS-k, *all* jobs in the system when j arrives are *permanently* relevant to j.

This means that if the service requirement distribution S is unbounded, our worst-case technique is insufficient for bounding the difference in relevant work between FCFS-1 and FCFS-k.

2.9 Technical Conclusion

We give the first stochastic bound on the response time of SRPT-k (see Section 2.6). Using this bound, we show that SRPT-k has asymptotically optimal mean response time in the heavy-traffic limit (see Section 2.7). We generalize our analysis to give the first stochastic bounds on the response times of the PSJF-k, RS-k and FB-k policies, and we use these bounds to prove asymptotic optimality results for all three policies (see Section 2.8).

To achieve these results, we strategically combine stochastic and worst-case techniques. Specifically, we obtain our bounds using a modified tagged job analysis. Traditional tagged job analyses for single-server systems rely on properties that do not hold in multiserver systems, notably work conservation. To make tagged job analysis work for multiple servers, we use two key insights.

- We introduce the concept of *virtual work* (see Section 2.6), which makes the system appear work-conserving while the tagged job is in the system. We give a worst-case bound for virtual work.
- We compare the multiserver system with a *single-server system of the same service capacity*. We show that even in the worst case, the steady state amount of relevant work under SRPT-k is close to the steady state amount of relevant work under SRPT-1.

Applying these two insights to the tagged job analysis gives a *stochastic* expression bounding response time.

One direction for future work is to apply our technique to a broader range of scheduling policies. In particular, we conjecture that out results generalize to the SMART class of policies [233], which includes SRPT, PSJF, and RS. Another direction is to improve our response time bounds under low system load. While our bounds are valid for all loads, they are only tight for load near capacity.

2.10 General Conclusion

2.10.1 Summary

We start by summarizing the results proven in this chapter, as well as the key techniques behind these results.

Results: Multiserver SRPT is optimal We prove that the multiserver SRPT scheduling policy achieves the best possible mean response time when the load on the system is high and queue lengths become long (See Theorem 2.7.1).

We also prove an upper bound on the mean response time of multiserver SRPT, in the form of a clean mathematical formula (See Theorem 2.6.2). This upper bound proves that multiserver SRPT achieves similar mean response time to that of resource-pooled SRPT, in which all k servers are combined into a single ultra-fast server, which runs single-server SRPT. This bound becomes tight as load becomes high. The bound is looser at moderate traffic. Further research will be needed to more tightly bound multiserver SRPT's mean response time under moderate traffic.

Simulation result: SRPT is near-optimal under moderate traffic While our results focus on heavy traffic, we show via simulation in Fig. 2.6 that SRPT-k is also very good for moderate traffic. In particular, we see that the ratio between SRPT-k and our lower bound of SRPT-1 is near 1 even at moderate loads. This implies that SRPT-k is near optimal even at moderate loads.

Results: Other scheduling policies We prove similar optimality results and upper bounds for several more multiserver scheduling policies in Section 2.8: Preemptive Shortest Job First (PSJF), which preemptively prioritizes the job with least *original* size, Remaining Size Times Original Sizes (RS), which is a halfway point between PSJF and SRPT, and Foreground Background (FB), which preemptively prioritizes the job with least service already completed. FB is useful in settings where job sizes are unknown to the server [182, 183], in contrast to the known-size setting that is the focus of this chapter. We prove that each of these policies achieves optimal mean response time when load becomes high, in each policy's respective setting. We also prove upper bounds on each policy's mean response time, proving that each policy resembles its respective resource-pooled variant. These bounds also become tight as load becomes high.

Key techniques To understand the response time of a specific job, we start by understanding work in the system is *relevant* to the specific job: what work the specific job must wait behind, before it can reach service. The work that is relevant to a specific job of size x consists of other jobs with remaining size smaller than x. Our first proof step, in Section 2.6.2, focuses on bounding the random distribution of relevant work in the system.

To bound relevant work, we need a baseline to compare against. For that baseline, we use a resource-pooled system, where all k original servers are combined into one gigantic server which runs k times as fast. In the resource-pooled system, we consider the SRPT scheduling policy. We show that multiserver SRPT can nearly keep up with resource-pooled SRPT, and has not much more relevant work than resource-pooled SRPT.

Coupling analysis To compare the multiserver and resource-pooled systems, we set up a coupling: We send the same stochastic arrival process to both systems, and analyze the amount of relevant work in both systems. To perform this analysis, we split up time into two kinds of intervals: Periods where there are fewer than k relevant jobs, and periods where there are at least k relevant jobs. We show that in each period of time, the difference in relevant work between the coupled multiserver SRPT and resource-pooled SRPT systems is small: No more than k job's worth of work.

Finally, we use this bound on relevant work to show that multiserver SRPT has nearly the same mean response time as resource-pooled SRPT (See Theorem 2.6.1).

2.10.2 Subsequent results

The techniques discussed in Section 2.10.1 have been extended to a wide variety of multiserver scheduling systems:

Unknown sizes and size estimates: While this chapter focuses on the setting where job sizes are known to the scheduler, one can apply the same proof structure to a setting where job sizes are unknown to the scheduler, or where only size estimates are known. A natural scheduling policy for this setting is the Gittins index policy, which is known to achieve optimal mean response time in the M/G/1 [72, 73, 199]. We first use this proof structure to analyze the *monotonic* Gittins policy, a close variant of Gittins, in the M/G/k with unknown sizes [204]. We prove the first heavy-traffic optimality results in those settings. We next use this proof structure, with a variety of new techniques, to analyze the Gittins policy in the M/G/k, handling unknown sizes, size estimates, and a variety of other scheduler-information settings, proving yet more heavy-traffic optimality results [202].

General interarrival times, setup times: Building off of the multiserver Gittins results, Hong and Scully [114] then extended the analysis to handle general i.i.d. interarrival time distributions, as well as general setup times, where servers take time to reactivate after going idle.

Dispatching: This chapter focuses on the central-queue setting, where every job is available to be served at any server. Another important multiserver model is the *dispatching* model, where each arriving job must be immediately sent to the server where it will eventually be served. The dispatching model is important for capturing the behavior of webservers, cloud computing systems, and other large-scale computing systems. Building on the techniques in this chapter, we prove the first results on optimal scheduling in the dispatching model in Chapter 3.

Multiserver-jobs: This chapter focuses on a setting where each job requires one server. In many real-world computing systems, such as supercomputers and datacenters, different jobs may require dramatically different amounts of resources, modeled as different numbers of servers. The resulting model, the *multiserver-job* model, is the focus of Chapters 4 to 6, with Chapter 5 most directly building off of this chapter's results to prove the first optimal scheduling results in the multiserver-job model.

2.10.3 Future directions

Improving upon SRPT-k This chapter proves that SRPT-k achieves optimal mean response time in heavy traffic. At the opposite extreme, with no arrivals, SRPT-k is likewise known to achieve optimal mean response time [77, Section 4.4.1] [152]. For moderate arrival rates, no such optimality results are known. As we discuss in Section 8.3.1, SRPT-k does *not* achieve optimal mean response time in the M/G/k, under specific combinations of arrival rate and job size distribution.

In particular, we have found alternative scheduling policies which outperform SRPT-k by up to 1% under specific combinations of job size distribution and load. SRPT-k appears to make suboptimal decisions when there are just over k jobs in the system. In such situations, SRPT-k

will rapidly complete jobs, bringing the total number of jobs remaining below k, and wasting some server capacity. When jobs later arrive, there is more work in the system than necessary, potentially hurting mean response time.

We therefore ask:

> Can we devise a scheduling policy with lower mean response time than SRPT-k, across all arrival rates and job size distributions?

This lower arrival rate will be asymptotically negligible, but could be significant at intermediate loads.

Stronger lower bonds At intermediate loads, outside of heavy traffic and light traffic, we know little about optimal scheduling in the M/G/k: The upper bounds on SRPT-k proven in this chapter are not tight. There are two natural lower bounds: SRPT-1, from the resource-pooled system, and $k\mathbb{E}[S]$, the mean service time. At intermediate loads, around $\rho = 1 - 1/k$, neither of these bounds are tight. Proving stronger lower bounds on all policies, not just SRPT-k, is of particular interest. We discuss the problem of proving stronger lower bounds in the M/G/k in Section 8.3.2.

2.10.4 Potential impact

We now explore potential directions in which the multiserver SRPT policies could be applied, and discuss ways of adapting the multiserver SRPT policy to real-world environments.

Adopting SRPT into Multiserver Computing Systems

Our analysis of SRPT in multiserver systems shows that the SRPT policy can dramatically lower response times compared to other scheduling policies, and that SRPT is in fact optimal for mean response time under sufficiently heavy load. Our hope is that our results will lead computing system operators to adopt SRPT scheduling in their systems.

However, the road from theoretical results to adoption requires overcoming several hurdles. These include:

1. Verifying that multiserver SRPT achieves low mean response time under the moderate loads that real computing systems operate under. In [78], we show that multiserver SRPT empirically achieves low mean response time under moderate load in simulation. Proving that this behavior always holds is a major open problem. For practical applications, trace-based simulation may be most useful.
2. Ensuring that SRPT won't "starve" large jobs by dramatically increasing their mean response times. While SRPT will remain stable, and jobs won't literally take forever to complete, SRPT may dramatically increase the mean response time of the largest jobs, under certain job size distributions. This question has been investigating in the single-server setting, both in theory and practice. In the single-server setting, this behavior primarily occurs under *low variability* job size distributions. Under the high variability job size distributions commonly found in real computing systems, every size of job can benefit from being prioritized ahead of even larger jobs, resulting in mean response time improvement at every size [17]. This theoretical result has been replicated in practice in the setting

of webserver scheduling [41, 110]. More research is needed to determine whether these results transfer from the single-server setting to the multiserver setting.

3. Dealing with unknown sizes. Often only approximate size information is known, rather than the exact size information utilized by SRPT. If the estimate quality distribution is known, the Gittins index policy can be applied, as discussed in Section 2.10.2. In the more common situation where approximation quality is unknown, the SRPT-B policy is a good option. Under the SRPT-B policy, a job's priority improves until it has received its estimated service duration, after which the priority reduces back down again. We introduced and analyzed SRPT-B in the single-server setting with approximations of unknown quality [205]. We believe its advantageous properties likely carry over to the multiserver setting.
4. Handling jobs with differing importance. In the real world, some jobs may be far more latency-sensitive than others. This is naturally modeled by introducing a holding cost per second waited for each job. The natural generalization of SRPT to this setting is to prioritize jobs according to the ratio of holding cost to remaining size. The results of this chapter generalize to prove optimality in that setting as well. Specifically, if the ratio of largest to smallest holding cost is bounded, the techniques of this chapter suffice, while Gittins-based techniques along the lines of our Gittins-k result may be helpful to handle the setting of general holding costs [202].

SRPT for Scheduling People at Multiple Servers

One might think that if SRPT is so good at reducing mean response time for computing systems, it should also be applied to scheduling people. For example, if two people show up to a post office, and one needs to mail a letter while the other needs to fill out some complicated shipping paperwork, we should first serve the "short job", the person who won't take as long.

Challenge: Fairness However, challenges arise when serving people. People have strong opinions about fairness: If everyone is put in the same line, and then certain people are pulled for out for favored treatment, people will get upset. However, if one makes the scheduling structure transparent to the people involved, it will help quell that anger and incentivize people towards more helpful behavior. For instance, there could be two lines, one for quick interactions, and one for slow interactions: Few people would begrudge someone in a different line being served before them.

Another measure of fairness is the mean response time of jobs of a specific size, such as the mean response time of large jobs. In the single-server scheduling setting, research has shown that SRPT gives jobs of size x a mean response time approximately proportional to x [17, 109]. This proportional response time is also described as *constant slowdown*, where a job's slowdown is the ratio between its response time and its size. Constant slowdown is considered a desirable goal from the perspective of fairness. We prove in Theorems 2.6.1 and 2.6.2 that under multiserver SRPT, a job of size x achieves similar mean response time as under resource-pooled SRPT. Thus, multiserver SRPT achieves the same approximately constant slowdown as singe-server SRPT.

Challenge: Preemption Preemption must be considered more carefully as well. Unlike computer jobs, it doesn't work to preempt people unpredictably. Instead, one should only preempt jobs that are taking a long time, and only when the reason for doing so is clear, such as

when a large line has built up. Long jobs holding up long queues is the only situation where preemption is very important to mean response time.

In addition to limiting the amount of preemption performed, when scheduling people, one must consider overheads due to preemption. While computer jobs can often be paused and resumed with negligible overhead, people are not so simple. Often, some time may be required to pick up where an interaction left off. Sometimes, one might need to restart the interaction entirely. Research is ongoing into characterizing when preemption is worthwhile in the presence of overheads [177, 200]. Under sufficiently high-variability conditions, even preemption involving a complete restart can lower mean response times [103].

Chapter 3

Optimal Dispatching and Scheduling: Guardrails

This chapter is based on the paper "Load Balancing Guardrails: Keeping Your Heavy Traffic on the Road to Low Response Times", published in ACM SIGMETRICS 2019, written with my coauthors Ziv Scully and Mor Harchol-Balter [82].

3.1 General Introduction

Modeling dispatching in computer systems Modern computing systems typically contain huge numbers of servers, allowing them to process many jobs at once. At this large scale, holding jobs in a central queue, as studied in Chapter 2, can become a bottleneck. Instead, these multiserver systems often *dispatch* an arriving job to a specific server, immediately upon arrival. Dispatching is often performed in webservers, remote-procedure-call (RPC) systems, and many other large computing systems.

Dispatching model To capture the behavior of these computing systems, we can use an immediate-dispatch queueing model, depicted in Fig. 3.1. New jobs arrive at the dispatcher, which sends that job to a queue at a server. At the server, a scheduling policy decides which

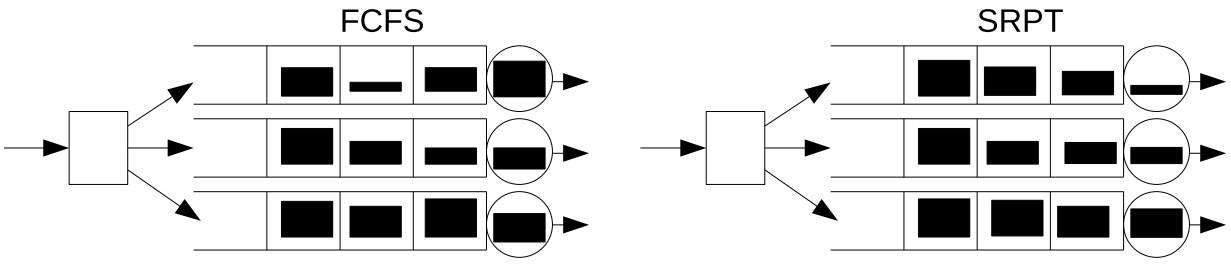


Figure 3.1: The multiserver dispatching model, with FCFS and SRPT scheduling. Each job arrive to the dispatcher, which sends the job to one of the k servers. The scheduling policy determines which job to serve, out of the jobs present at a given server. FCFS serves the oldest job, which SRPT servers the job with smallest remaining size.

of the jobs in the local queue to run. Both the dispatching policy and the scheduling policy are important to the performance of the system. Two important scheduling policies, which we depict in Fig. 3.1, are First-Come First-Served (FCFS) and Shortest Remaining Processing Time (SRPT). FCFS serves the oldest job in the queue, while SRPT preemptively serves the job of least remaining size in the queue. A job's *size* is its service duration. *Preemption* refers to pausing a job in service, putting it back in the same queue it came from, and returning to it later. We are interested in the performance effects of these dispatching and scheduling policies. A natural measure of the combined performance of a dispatching policy and a scheduling policy is the system's *mean response time*, the mean time from when a job arrives to when it completes.

Prior focus: Dispatch to FCFS The are many important dispatching policies which have been theoretically analyzed in combination with FCFS scheduling at each server. Such policies include Join-Shortest-Queue (JSQ), which sends each job to to the server with the fewest jobs at its queue, and Least-Work-Left (LWL), which sends each job to the server with the least *work*, where the work at a server is the total remaining size of all jobs at the server. LWL dispatching to FCFS queues achieves the same response time distribution as the central-queue FCFS policy. Certain optimality results are known for JSQ [227, 234] and LWL [102] dispatching, but only under the assumptions that FCFS scheduling is used at the servers, and that the dispatcher does not know the size of the arriving job. These policies emphasize balancing the load of arriving jobs across all of the servers, ensuring that all servers are busy as often as possible.

If the size of the arriving job is known to the dispatcher, intentional load *imbalancing* can be beneficial [107, 197]. The Size-Interval-Task-Assignment (SITA) policy employs such load imbalancing. SITA statically routes jobs according to their sizes, sending smaller jobs to the servers with less total load, ensuring better response time for those small jobs, with the goal of improving overall response time [108]. SITA can outperform even LWL dispatching under certain loads and job size distributions [112]. Optimal dispatching to FCFS queues using size information is still an area of active research [237].

Prior work: Dispatch to Processor Sharing A practical setting that is far less studied in the literature is dispatching jobs to servers running Processor Sharing (PS). Under Processor Sharing, the server is split equally among all jobs in the queue. The combination of JSQ dispatching and PS service was studied in [96], and further studied in [94]. Interestingly, it has been shown that the mean response time under this combination is insensitive to the job size distribution [94].

Optimal dispatching + scheduling Mean response time is only well-understood in dispatching systems where the scheduling policy is either FCFS or PS. However, it is straightforward to show that SRPT scheduling is the optimal scheduling policy for minimizing mean response time, in combination with any dispatching policy. This optimality follows from the worst-case optimality of SRPT [194]. Surprisingly, policies with great performance under FCFS scheduling, such as LWL and SITA, can have terrible performance under SRPT scheduling, as shown in Fig. 3.4. Thus, the focus of this chapter is:

> What dispatching policy minimizes mean response time, given SRPT scheduling at each server?

Key idea: Same mix of jobs To achieve optimal mean response time, we invent a novel class of dispatching policies, "guardrails" policies. The key idea behind guardrails policies is to balance not just the *total* amount of work at each server as in LWL, but also the amount of

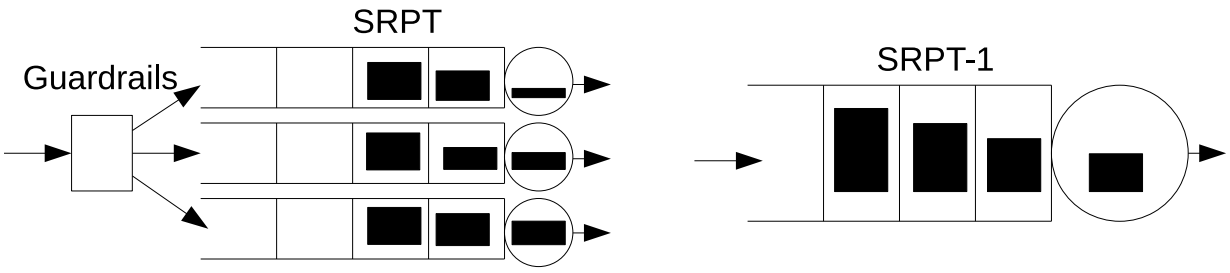


Figure 3.2: The Guardrails/SRPT system, showing the combination of a generic guardrails dispatching policy and the SRPT scheduling policy, and the SRPT-1 system. In the SRPT-1 system, the SRPT policy is used in a resource-pooled setting, with one gigantic server which runs as fast as all of the original servers combined.

work of *each kind of job* at each server. Imagine labeling every job as small, medium, or large, based on its size. Guardrails policies ensure that each server has the same amount of work of small jobs, the same amount of work of medium jobs, and so forth. As a result, if any small jobs are present in the system, all servers will work on small jobs, improving response times. This property is at the heart of our optimality results for guardrails policies.

Key idea: Resource pooling Our key proof technique is to compare the guardrails/SRPT combination against a resource-pooled system where all k servers have been combined into one gigantic server. This gigantic server can serve any job at k times the speed of the original k servers. This resource-pooled system is strictly more capable than the original k-server dispatching system: One could split the gigantic server's efforts k ways, to emulate the original system, but one also has far more options. The resource-pooled system forms a lower bound that we can compare the guardrails/SRPT combination against. Specifically, we compare against the SRPT policy for the resource-pooled system, which we call the SRPT-1 system. We show both systems in Fig. 3.2. Using the fact that each server has the same mix of jobs as each other server, we show that each server in the dispatching system operates like a miniature version of the SRPT-1 system. This forms the core of our proof of the optimality of the guardrails/SRPT system.

3.2 Technical Introduction

Load balancers are ubiquitous throughout computer systems. They act as a front-end to web server farms, distributing HTTP requests to different servers. They likewise act as a front-end to data centers and cloud computing pools, where they distribute requests among servers and virtual machines.

In this chapter, we consider the immediate dispatch load balancing model, where each arriving job is immediately dispatched to a server, as shown in Figure 3.3. The system has two decision points:

(1) A *dispatching policy* decides how to distribute jobs across the servers.

(2) A *scheduling policy* at each server decides which job to serve among those at that server.

We ask:

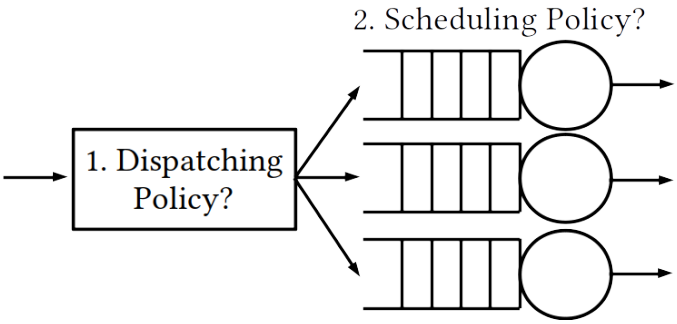


Figure 3.3: Two decision points within a load balancing system: (1) Pick the dispatching policy. (2) Pick the scheduling policy for the servers.

> What (1) dispatching policy and (2) scheduling policy should we use to *minimize mean response time* of jobs?

We assume that the job arrival process is Poisson and that job sizes are i.i.d. from a general size distribution. We assume jobs are preemptible with no loss of work. Finally, we assume that job sizes are known at the time the job arrives in the system.

With these assumptions, the scheduling question turns out to be easy to answer: use Shortest-Remaining-Processing-Time (SRPT) at the servers. No matter what dispatching decisions are made, if we consider the sequence of jobs dispatched to a particular server, the policy which minimizes mean response time for that server must be to schedule those jobs in SRPT order. This follows from from the optimality of SRPT for arbitrary arrival sequences [194]. SRPT scheduling is in fact already used in backend servers [110, 160]. Thus, in the remainder of this chapter we assume SRPT is used at the servers.

The question remains: What dispatching policy minimizes mean response time given SRPT service at the servers? While many dispatching policies have been considered in the literature, they have mostly been considered in the context of First-Come-First-Served (FCFS) or Processor-Sharing (PS) scheduling at the servers. Popular dispatching policies include Random routing [108, 156], Least-Work-Left (LWL) [29, 108, 112], Join-Shortest-Queue (JSQ) [24, 96, 227, 234], JSQ-d [29, 137, 156, 162], Size-Interval-Task-Assignment (SITA) [14, 62, 108], Round-Robin (RR) [108, 143], and many more [7, 23, 46, 244]. However, only the simplest of these policies, such as Random and RR, have been studied for SRPT servers [51, 104].

One might hope that the same policies that yield low mean response time when servers use FCFS scheduling would also perform well when servers use SRPT scheduling. Unfortunately, this does not always hold. For example, when the servers use FCFS, it is well-known that LWL dispatching, which sends each job to the server with the least remaining work, outperforms Random dispatching, which sends each job to a randomly chosen server. (We write this as LWL/FCFS outperforms Random/FCFS.) However, the opposite can happen when the servers use SRPT: as shown in the scenario in Figure 3.4, Random/SRPT can outperform LWL/SRPT. Moreover, the performance difference is highly significant: Random/SRPT outperforms LWL/SRPT by a factor of 5 or more under heavy load. This means that LWL is making serious mistakes in dispatching decisions. We can therefore see that the heuristics that served us well for FCFS servers can steer us awry when we use SRPT servers.

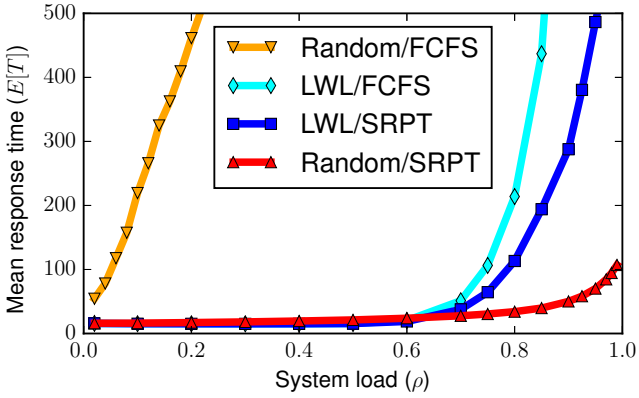


Figure 3.4: Two dispatching policies: Random and LWL. Two scheduling policies: FCFS and SRPT. FCFS scheduling at the servers yields higher mean response time as a function of load, compared with SRPT scheduling at the servers. Random dispatching is worse than LWL dispatching under FCFS scheduling at the servers, but Random dispatching is better than LWL dispatching under SRPT scheduling at the servers. Simulation uses $k = 10$ servers. Size distribution shown is Bimodal with jobs of size 1 with probability 99.95% and jobs of size 1000 with probability 0.05%.

In this chapter, we introduce *guardrails*, a new technique for creating dispatching policies. Given an arbitrary dispatching policy P, applying guardrails results in an improved policy Guarded-P (G-P). We prove that the improved policy G-P has asymptotically optimal mean response time in the heavy traffic limit, no matter what the initial policy P is. We also show empirically that adding guardrails to a policy almost always decreases its mean response time (and never significantly increases it), even outside the heavy-traffic regime.

As an example of the power of guardrails, Figure 3.5 shows the performance of guarded versions of LWL and Random, namely G-LWL and G-Random. The guardrails stop LWL from making serious mistakes and dramatically improve its performance. Random dispatching also benefits from guardrails. Moreover, the guarded policies have a theoretical guarantee: In the limit as load $\rho \to 1$, G-Random/SRPT and G-LWL/SRPT converge to the optimal mean response time. In contrast, unguarded Random/SRPT is a factor of k worse than optimal in the $\rho \to 1$ limit, where k is the number of servers.

This chapter makes the following contributions:

- In Section 3.3, we introduce guardrails, a technique for improving any dispatching policy.
- In Section 3.5, we bound the mean response time of any guarded dispatching policy when paired with SRPT scheduling at the servers. Using that bound, we prove that any guarded policy has asymptotically optimal mean response time as load $\rho \to 1$, subject to a technical condition on the job size distribution roughly equivalent to finite variance.
- In Section 3.6, we consider a wide variety of common dispatching policies. We empirically show that guardrails improve most of these at all loads.
- In Section 3.8, we discuss practical considerations and extensions of guardrails, including

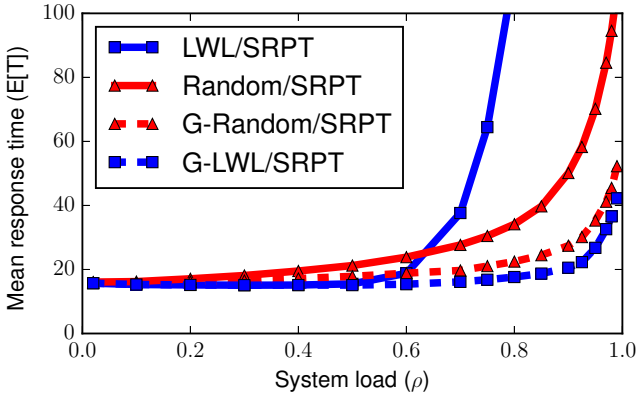


Figure 3.5: Adding guardrails to LWL yields much lower mean response time as a function of load. Guardrails yield a factor of 3 improvement even at $\rho = 0.8$, and a factor of 7 improvement at $\rho = 0.9$. Adding guardrails to Random also yields significantly lower mean response time as a function of load. Simulation uses $k = 10$ servers. Size distribution shown is Bimodal with jobs of size 1 with probability 99.95% and jobs of size 1000 with probability 0.05%. The guardrails have tightness $g = 2$.

- guardrails for large systems, which may have multiple dispatchers and network delays;
- guardrails for scheduling policies other than SRPT; and
- guardrails for heterogeneous servers.

We give a more technical summary of our theoretical results and review related work in Section 3.4.

3.3 Load Balancing Guardrails

3.3.1 What are Guardrails?

Traditional dispatching policies aim to equalize load at each server. However, minimizing mean response time requires more than balancing load: we also need to find a way to favor small jobs. Given that every server uses SRPT scheduling, if we can spread out the small jobs across the servers, then we ensure that the maximum possible number of servers are working on the smallest jobs available. Our idea is to take any dispatching policy and add "guardrails" that force it to spread out small jobs across the servers.

In the discussion above, "small" is a relative term. Whatever the size of the smallest jobs currently in the system, we would like to spread out jobs near that size across the servers. To do this, we define the *rank* of a job of size x to be

$$r = \lfloor \log_c x \rfloor, \tag{3.1}$$

where $c > 1$ is a constant called the *guardrail rank width* (see Section 3.3.1). The idea of

guardrails is to spread out the jobs within a rank r across the servers, doing so separately for each rank r. To do so, for each rank r and each server s, the dispatcher stores a *guardrail work counter* G_s^r. When dispatching a job of size x to server s, the dispatcher increases G_s^r by x, with r given by (3.1).[1] Guardrails are a set of constraints which ensure that no two rank r work counters are ever too far apart.

Definition 3.3.1. *A dispatching policy* satisfies guardrails with tightness g *if at all times*

$$|G_s^r - G_{s'}^r| \leq gc^{r+1}$$

for all ranks r and all pairs of servers s and s', where $c > 1$ is the same constant as in (3.1)*. The tightness can be any constant $g \geq 1$.*

We sometimes say that a particular dispatching decision satisfies (respectively, violates) guardrails if it satisfies (respectively, violates) the constraints imposed by Definition 3.3.1.

Choosing the Guardrail Rank Width c

The choice of c in (3.1) heavily affects the performance of policies satisfying guardrails.

- If c is too large, then guardrails may not differentiate between jobs of different sizes.
- If c is too small, then guardrails may misguidedly differentiate between jobs of similar sizes. This could allow one server to receive multiple small jobs of different ranks while another receives none.

To balance this tradeoff, we set c to be a function of load ρ:

$$c = 1 + \frac{1}{1 + \ln \frac{1}{1-\rho}}. \tag{3.2}$$

This particular value of c is chosen to enable the heavy-traffic optimality proof for any dispatching policy satisfying guardrails.

3.3.2 Guarded Policies: How to Augment Dispatching Policies with Guardrails

Guardrails as described in Definition 3.3.1 are a set of constraints on dispatching policies that we will use to guarantee bounds on mean response time (see Section 3.5). However, the constraints alone do not give a complete dispatching policy. To define a concrete dispatching policy satisfying guardrails, we start with an arbitrary dispatching policy P and augment it to create a new policy, called *Guarded-P* (G-P), which satisfies guardrails.

Roughly speaking, G-P tries to dispatch according to P, but if dispatching to P's favorite server would violate guardrails, G-P considers P's second-favorite server, and so on. Below are guarded versions of some common dispatching policies:

- G-Random dispatches to a random server among those which satisfy guardrails.
- G-LWL dispatches to the server with the least remaining work among those which satisfy guardrails.

[1] The dispatcher also occasionally decreases work counters, as explained in Section 3.3.3.

- Round-Robin (RR) can be seen as always dispatching to the server that has least recently received a job, so G-RR dispatches to the server that has least recently received a job among those which satisfy guardrails.

Given an arbitrary dispatching policy P, Algorithm 1 formally defines G-P. We assume that P is specified by procedure Dispatch$^{\text{P}}$ which, when passed a job of size x and a set of servers $\mathcal{S}$, returns a server in $\mathcal{S}$ to which P would dispatch a job of size x. The key to Algorithm 1 is that instead of calling Dispatch$^{\text{P}}$ with the set of all servers, we pass it a restricted set of servers $\mathcal{S}_{\text{safe}} \subseteq \mathcal{S}$ such that dispatching to any server in $\mathcal{S}_{\text{safe}}$ will satisfy guardrails. $\mathcal{S}_{\text{safe}}$ is never empty because $x \leq gc^{r+1}$, so $\mathcal{S}_{\text{safe}}$ will always contain the server s' of minimal $G^r_{s'}$.

Algorithm 1. The Guarded-P (G-P) policies is defined as follows:

Given a dispatching policy P, tightness $g \geq 1$, set of servers $\mathcal{S}$, and rank width $c = 1 + \frac{1}{1+\ln\frac{1}{1-\rho}}$,

Initialize$^{\text{G-P}}$(): When the system starts, for each rank $r \in \mathbb{Z}$ and each server $s \in \mathcal{S}$, initialize the counter $G^r_s = 0$.

Dispatch$^{\text{G-P}}$(x): Whenever a job of size x arrives,

- Let the job's rank r be $\lfloor \log_c x \rfloor$.
- Let the minimal counter value $G_{\min}$ be the minimum over servers $s' \in \mathcal{S}$ of $G^r_{s'}$.
- Let the safe set $\mathcal{S}_{\text{safe}}$ be the set of servers $\{s' \in \mathcal{S} \mid G^r_{s'} + x \leq G_{\min} + gc^{r+1}\}$.
- Dispatch to the server s chosen by dispatch policy P: $s = \text{Dispatch}^{\text{P}}(x, \mathcal{S}_{\text{safe}})$
- Increment the counter: $G^r_s \leftarrow G^r_s + x$.

Reset$^{\text{G-P}}$(s): Whenever a server s becomes empty, for every rank $r \in \mathbb{Z}$, set the counter $G^r_s = \min_{s' \in \mathcal{S}} G^r_{s'}$, the minimum rank-$r$ counter value in the system.

Algorithm 1 is phrased in terms of for loops over all ranks r. While there are infinitely many ranks in theory, it is simple to represent all of the work counters in finite space by representing most of them implicitly.

3.3.3 Resets

Algorithm 1 includes a procedure, Reset$^{\text{G-P}}$, which we have not yet explained. As defined so far, guardrails effectively spread out small jobs across the servers, but they have an unfortunate side effect: they sometimes prevent dispatches to empty servers. This is because the work counters G^r_s as defined so far depend only on the dispatching history, not the current server state.

Because dispatching to empty servers is desirable, we would like to ensure that dispatching to an empty server never violates guardrails. We accomplish this by having servers *reset* whenever they becomes empty. When a server s resets, for each rank r, we decrease G^r_s to match the minimum among all rank r work counters. Because all rank r jobs have size less than gc^{r+1}, by Definition 3.3.1, dispatching to a server that has just reset will never violate guardrails.

3.4 Technical Summary

3.4.1 System Model

We will study a k-server load balancing system with Poisson arrivals at rate λ jobs per second and job size distribution X. Our optimality results (Theorem 3.4.2) assume that $\mathbb{E}\left[X^2 \log^2 X \mathbb{1}\{X > 1\}\right]$ is finite. This assumption is roughly equivalent to finite variance. We adopt the convention that each of the k servers serves jobs at speed $1/k$. As a result, a job of size x requires kx service time to complete. We have chosen to define the speed of a server this way because we will later compare the k-server system with a single server system of speed 1, and this convention allows us to directly apply standard results on single server systems. We define the system load ρ for both a single-server system and the k-server system by

$$\rho = \lambda \mathbb{E}[X] < 1.$$

Load does not depend on k because the total service rate of all k servers combined is 1.

Throughout, we assume that the dispatching policy is a guarded policy, as defined in Algorithm 1. We consider two different scheduling policies that might be used at the servers:

SRPT The policy that serves the job of least remaining size.

Priority-*c* The preemptive class-based priority policy in which a job's class is its *rank*, as defined by (3.1). That is, a job of size x has rank $r = \lfloor \log_c x \rfloor$, and Priority-$c$ serves the job of minimal rank. Within each rank, jobs are served FCFS.

3.4.2 Theorem Overview

Our overall goal is to prove, for any dispatching policy P, the asymptotic optimality of the policy Guarded-P (G-P) with respect to mean response time, given SRPT scheduling at the servers. We refer to this joint dispatch/scheduling policy as G-P/SRPT.

Rather than studying G-P/SRPT directly, we instead bound mean response time under G-P/Priority-c. By the optimality of SRPT scheduling [194], the mean response time under G-P/Priority-c gives an upper bound on the mean response time under G-P/SRPT.

Theorem 3.4.1. *For any dispatching policy P, consider the policy G-P with tightness g. The expected response time for a job of size x under G-P/Priority-c is bounded by*

$$\mathbb{E}[T(x)]^{\text{G-P/Priority-}c} \leq \frac{\frac{\lambda}{2}\int_0^{c^{r+1}} t^2 f_X(t)dt}{(1-\rho_{c^r})(1-\rho_{c^{r+1}})} + \frac{(4c+2)gk\frac{c^{r+1}}{c-1} + kx}{(1-\rho_{c^r})},$$

where

- *$f_X(\cdot)$ is the probability density function of X,*
- *c is the guardrail rank width*
- *$r = \lfloor \log_c x \rfloor$ is the rank of a job of size x, and*
- *$\rho_y = \lambda \int_0^y t f_X(t)\, dt$ is the load due to jobs of size $\leq y$.*

We prove Theorem 3.4.1 in Section 3.5.3.

Using the bound in Theorem 3.4.1, we show that the mean response time of the G-P/Priority-c system converges to that of a single-server SRPT system in the heavy-traffic limit.

Theorem 3.4.2. *Consider a single-server SRPT system whose single server is k times as fast as each server in the load balancing system. For any dispatching policy P, consider the policy G-P with any constant tightness. Then for any size distribution X such that $\mathbb{E}\left[X^2 \log^2 X \mathbb{1}\{X > 1\}\right] < \infty$, the mean response times of G-P/SRPT, G-P/Priority-c, and (single-server) SRPT converge as load approaches capacity:*

$$\lim_{\rho \to 1} \frac{\mathbb{E}\left[T\right]^{\text{G-P/SRPT}}}{\mathbb{E}\left[T\right]^{\text{SRPT}}} = \frac{\mathbb{E}\left[T\right]^{\text{G-P/Priority-}c}}{\mathbb{E}\left[T\right]^{\text{SRPT}}} = 1.$$

We prove Theorem 3.4.2 in Section 3.5.4.

Theorem 3.4.2 relates the mean response times of G-P/SRPT and single-server SRPT, which has the optimal mean response time among all single-server policies [194]. But a single-server system can simulate a load balancing system running any joint dispatching/scheduling policy P′/S′. As a result, the mean response time under single-server SRPT is a lower bound for the mean response time under P′/S′.

Using that bound, Theorem 3.4.2 implies the following relationship between the mean response times of G-P/SRPT and P′/S′.

Corollary 3.4.1. *For any dispatching policy P consider the policy G-P with any constant tightness. Consider any joint dispatching/scheduling policy P′/S′. Then for any size distribution X such that $\mathbb{E}\left[X^2 \log^2 X \mathbb{1}\{X > 1\}\right] < \infty$, the mean response times of G-P/SRPT and G-P/Priority-c are at least as small as the mean response time of P′/S′ as load approaches capacity:*

$$\lim_{\rho \to 1} \frac{\mathbb{E}\left[T\right]^{\text{G-P/SRPT}}}{\mathbb{E}\left[T\right]^{\text{P′/S′}}} = \frac{\mathbb{E}\left[T\right]^{\text{G-P/Priority-}c}}{\mathbb{E}\left[T\right]^{\text{P′/S′}}} \leq 1.$$

3.4.3 Relationship to Prior Work

Our guardrails provide the first mechanism to augment an arbitrary dispatching policy to ensure size balance at all job size scales. Moreover, we give the first bound on the mean response time of load balancing systems with SRPT scheduling at the servers. Using this bound, we prove that guarded dispatching policies have asymptotically optimal mean response time in the $\rho \to 1$ limit for any constant number of servers. Our guarded policies are the first dispatching policies known to have this property for general job size distributions.

We are not the first to consider load balancing systems with SRPT scheduling at the servers. Avrahami and Azar [12] consider a problem analogous to ours but in a *worst-case* setting, assuming adversarial arrival times and job sizes, in contrast to our stochastic setting. Their dispatching policy, which they call IMD, divides jobs into size ranks in a manner similar to our size ranks, except with the width of each rank set to $c = 2$. IMD dispatches each rank r job to the server that has received the least work of rank r jobs in the past. Put another way, IMD is the policy that keeps maximally tight guardrails with no other underlying policy. Avrahami and Azar prove that IMD is $O(\log P)$ competitive with an optimal migratory offline algorithm, where P is the ratio between the largest and smallest job sizes in this system. Note that P can be arbitrarily large for general job size distributions. Unfortunately, the $O(\log P)$ competitive ratio is optimal for any online dispatching policy in the worst-case setting [135]. Our result is much stronger thanks to our stochastic setting.

Down and Wu [51] also consider a stochastic setting with SRPT scheduling at the servers, and they also propose a dispatching policy that balances jobs of different sizes across the servers. Their analysis does not result in any formula for mean response time but instead uses a diffusion limit argument to show optimality in heavy traffic. However, this limits their results to discrete job size distributions, thus excluding many practically important continuous job size distributions. In fact, Down and Wu [51, Section 5] observe empirically that their policy performs poorly for Bounded Pareto job size distributions. In contrast, our analysis shows that any guarded dispatching policy is heavy-traffic optimal for general job size distributions, including Bounded Pareto (see Figure 3.6). Finally, the Down and Wu [51] result provides no insight into mean response time outside of the heavy traffic regime, whereas we derive a mean response time bound that is valid for all loads.

3.5 Analysis of Guarded Policies

In this section, we analytically bound the mean response time of a load balancing system using an arbitrary guarded dispatching policy G-P paired with SRPT scheduling. We then show that our bound implies that G-P/SRPT minimizes mean response time in heavy traffic.

3.5.1 Preliminaries and Notation

We use a tagged job analysis: we analyze the expected response time of a particular "tagged" job, which we call j, arriving to a steady-state system. The expected response time of j is equal to the system's mean response time by the PASTA property [235].

Instead of studying G-P/SRPT directly, we analyze G-P/Priority-c, which yields an upper bound on the mean response time under G-P/SRPT. Studying Priority-c simplifies the analysis because the priority classes of Priority-c match the ranks used by guardrails.

Suppose that j has rank r and is dispatched to server s. Under Priority-c scheduling, there are two types of work that might delay job j:

- The *current relevant work* at server s when j arrives. This is the total amount of remaining work at server s due to jobs of rank $\leq r$.
- The *future relevant work* due to arriving jobs dispatched to server s while j is in the system. These are the jobs dispatched to s of rank $< r$ (that is, rank $\leq r-1$).

We use the following notation, where "rank r work" denotes work due to jobs of rank r.

- $W_s^r(t)$ denotes the current amount of rank r work at server s at time t.
- $V_s^r(t)$ denotes the total amount of rank r work that has ever been dispatched to server s up to time t. In particular, the amount of rank r work dispatched to s during a time interval (t_1, t_2) is $V_s^r(t_2) - V_s^r(t_1)$.
- $G_s^r(t)$ denotes the rank r guardrail work counter for server s at time t (see Algorithm 1). Specifically, $G_s^r(t)$ is defined as follows:[2]

[2]The notations t^- and t^+ below refer to "just before" and "just after" time t. More formally, they refer to the left and right limits, respectively, of an expression that is piecewise-continuous in t.

- If a rank r job of size x is dispatched to server s at time t, we set $G_s^r(t^+) = G_s^r(t^-)+x$.
- If a server s becomes empty of all jobs at time t, we set $G_s^r(t^+) = \min_{s'} G_{s'}^r(t^-)$, where s' ranges over all servers. We call this a *reset* of server s.
- Otherwise, $G_s^r(t)$ does not change.

We write $W_s^{\leq r}(t)$, $V_s^{\leq r}(t)$, and $G_s^{\leq r}(t)$ to denote the corresponding quantities where we consider all ranks $\leq r$, rather than just rank r, and similarly for superscript $< r$.

Occasionally, we will be talking about the *total* work in the system, or the *total* work that has arrived, summed over all servers. In that case, we will drop the subscript s, writing $W^{\leq r}(t)$ or $V^{\leq r}(t)$. Finally, we write $W^{\leq r}$ to denote the stationary distribution of the amount of rank $\leq r$ work in the whole system.

3.5.2 Bounding Response Time: Key Steps

Our goal in this section is to bound the expected response time of a tagged job j under G-P/Priority-c. We assume that j has size x and rank $r = \lfloor \log_c x \rfloor$. We first bound current relevant work, then move on to bound future relevant work.

We begin by showing that guardrails ensure that any two servers have a similar amount of remaining rank $\leq r$ work.

Lemma 3.5.1. *For any dispatching policy P, consider the dispatching policy G-P with tightness g. In a G-P/Priority-c system, the difference in remaining rank $\leq r$ work between any two servers s and s' at any time t is bounded by*

$$W_s^{\leq r}(t) - W_{s'}^{\leq r}(t) \leq \frac{2gc^{r+2}}{c-1},$$

where c is the guardrail rank width.

We prove Lemma 3.5.1 in Section 3.5.3.

Roughly speaking, Lemma 3.5.1 shows that guarded policies do a good job of spreading out rank $\leq r$ work across the servers. This is important because if the rank $\leq r$ work is spread out well, then whenever there is a large amount of rank $\leq r$ work in the system, *all the servers* are doing rank $\leq r$ work. This allows us to bound the amount of rank $\leq r$ work in the k-server G-P/Priority-c system in terms of the remaining rank $\leq r$ work in an M/G/1/Priority-c system with a single server that runs k times as fast.

Lemma 3.5.2. *For any dispatching policy P, consider the dispatching policy G-P with tightness g. The total amount of remaining rank $\leq r$ work in a G-P/Priority-c system is stochastically bounded relative to the remaining rank $\leq r$ work in a M/G/1/Priority-c system whose server runs k times as fast:*

$$W^{\leq r} \leq_{\text{st}} W^{\leq r}_{\text{M/G/1/Priority-}c} + \frac{2gkc^{r+2}}{c-1},$$

where c is the guardrail rank width.

We prove Lemma 3.5.2 in Section 3.5.3.

Combining Lemmas 3.5.1 and 3.5.2 yields a bound on the amount of remaining rank $\leq r$ work at any server s, thus bounding the current relevant work.

We now turn to bounding future relevant work. Suppose that the tagged job j is dispatched to server s. The fact that guardrails spread out relevant work across the servers means that while j is in the system s will not receive much more rank $< r$ work than other servers, thus bounding future relevant work. Combining this with our bound on current relevant work yields the following bound on j's response time.

Lemma 3.5.3. *For any dispatching policy P, consider the dispatching policy G-P with tightness g. In a G-P/Priority-c system, the response time of a job of size x is stochastically bounded by*

$$T(x) \leq_{\text{st}} B_{<r}\left(W^{\leq r}_{\text{M/G/1/Priority-}c} + \frac{(4c+2)gkc^{r+1}}{c-1} + kx\right),$$

where c is the guardrail rank width, $r = \lfloor \log_c x \rfloor$ is the rank of the job, and $B_{<r}(w)$ is the length of a busy period comprising only jobs of rank $< r$ started by work w.

We prove Lemma 3.5.3 in Section 3.5.3.

Taking expectations in Lemma 3.5.3 and applying the well-known formula for $\mathbb{E}\left[W^{\leq r}_{\text{M/G/1/Priority-}c}\right]$, we obtain Theorem 3.4.1.

Theorem 3.4.1. For any dispatching policy P, consider the policy G-P with tightness g. The expected response time for a job of size x under G-P/Priority-c is bounded by

$$\mathbb{E}\left[T(x)\right]^{\text{G-P/Priority-}c} \leq \frac{\frac{\lambda}{2}\int_0^{c^{r+1}} t^2 f_X(t)dt}{(1-\rho_{c^r})(1-\rho_{c^{r+1}})} + \frac{(4c+2)gk\frac{c^{r+1}}{c-1} + kx}{(1-\rho_{c^r})},$$

where

- $f_X(\cdot)$ is the probability density function of X,
- c is the guardrail rank width
- $r = \lfloor \log_c x \rfloor$ is the rank of a job of size x, and
- $\rho_y = \lambda \int_0^y t f_X(t)\, dt$ is the load due to jobs of size $\leq y$.

We prove Theorem 3.4.1 in Section 3.5.3.

3.5.3 Bounding Response Time: Proofs

Proof of Lemma 3.5.1

Lemma 3.5.1. For any dispatching policy P, consider the dispatching policy G-P with tightness g. In a G-P/Priority-c system, the difference in remaining rank $\leq r$ work between any two servers s and s' at any time t is bounded by

$$W^{\leq r}_s(t) - W^{\leq r}_{s'}(t) \leq \frac{2gc^{r+2}}{c-1},$$

where c is the guardrail rank width.

Proof. Let t_0 be the most recent time up to time t when server s was empty of rank $\leq r$ work. Note that t_0 may equal t. We will bound the difference in rank $\leq r$ work at the two servers at time t by comparison with time t_0.

The remaining rank $\leq r$ work present at time t is

(i) the remaining rank $\leq r$ work present at time t_0
(ii) plus rank $\leq r$ work due to arrivals in the interval $[t_0, t]$
(iii) minus rank $\leq r$ work processed during the interval.

We consider these three quantities first for server s, then for server s'.

We begin with server s:

(i) By the definition of t_0, there is no remaining rank $\leq r$ work at server s.
(ii) The amount of work that arrives to server s over the interval $[t_0, t]$ is $V_s^{\leq r}(t) - V_s^{\leq r}(t_0)$.
(iii) The amount of rank $\leq r$ work processed during the interval $[t_0, t]$ is equal to $\frac{t-t_0}{k}$, because the server s processes work at speed $1/k$, s has rank $\leq r$ work available throughout the interval, and the Priority-c scheduling policy always prioritizes lower rank work.

These quantities give us the remaining rank $\leq r$ work at server s at time t:

$$W_s^{\leq r}(t) = (V_s^{\leq r}(t) - V_s^{\leq r}(t_0)) - \frac{t - t_0}{k}. \tag{3.3}$$

Because server s was never empty at any time during $(t_0, t]$, the guardrail work counters were never reset to the system-wide minimums during $[t_0, t]$. As a result, the changes in $G_s^{\leq r}(\cdot)$ and $V_s^{\leq r}(\cdot)$ over the interval $[t_0, t]$ must be equal. We can apply this fact to (3.3):

$$W_s^{\leq r}(t) = (G_s^{\leq r}(t) - G_s^{\leq r}(t_0)) - \frac{t - t_0}{k}. \tag{3.4}$$

We now turn to server s':

(i) The remaining rank $\leq r$ work present at server s' at time t_0 is non-negative.
(ii) The amount of work that arrives to server s over the interval $[t_0, t]$ is $V_{s'}^{\leq r}(t) - V_{s'}^{\leq r}(t_0)$.
(iii) The amount of rank $\leq r$ work processed over the interval $[t_0, t]$ is at most $\frac{t-t_0}{k}$.

Therefore, we may lower bound the remaining rank $\leq r$ work at server s' at time t:

$$W_{s'}^{\leq r}(t) \geq (V_{s'}^{\leq r}(t) - V_{s'}^{\leq r}(t_0)) - \frac{t - t_0}{k}. \tag{3.5}$$

The change in $G_{s'}^{\leq r}(\cdot)$ over the interval $[t_0, t]$ is no more than the change in $V_{s'}^{\leq r}(\cdot)$ over the same interval, since any reset to the system-wide minimum can only lead to a decrease in $G_{s'}^{\leq r}(t)$. We can apply this fact to (3.5):

$$W_{s'}^{\leq r}(t) \geq (G_{s'}^{\leq r}(t) - G_{s'}^{\leq r}(t_0)) - \frac{t - t_0}{k}. \tag{3.6}$$

Combining (3.4), (3.6), and the guardrail constraint in Definition 3.3.1 yields the desired

bound:

$$\begin{aligned}
W_s^{\leq r}(t) - W_{s'}^{\leq r}(t) &\leq (G_s^{\leq r}(t) - G_s^{\leq r}(t_0)) - (G_{s'}^{\leq r}(t) - G_{s'}^{\leq r}(t_0)) \\
&\leq |G_s^{\leq r}(t) - G_{s'}^{\leq r}(t)| + |G_s^{\leq r}(t_0) - G_{s'}^{\leq r}(t_0)| \\
&= \sum_{q=-\infty}^{r} (|G_s^q(t) - G_{s'}^q(t)| + |G_{s'}^q(t_0) - G_s^q(t_0)|) \\
&\leq \sum_{q=-\infty}^{r} 2gc^{q+1} \\
&= \frac{2gc^{r+2}}{c-1}.
\end{aligned}$$

□

Proof of Lemma 3.5.2

Lemma 3.5.2. For any dispatching policy P, consider the dispatching policy G-P with tightness g. The total amount of remaining rank $\leq r$ work in a G-P/Priority-c system is stochastically bounded relative to the remaining rank $\leq r$ work in a M/G/1/Priority-c system whose server runs k times as fast:

$$W^{\leq r} \leq_{\text{st}} W^{\leq r}_{\text{M/G/1/Priority-}c} + \frac{2gkc^{r+2}}{c-1},$$

where c is the guardrail rank width.

Proof. We consider two coupled systems receiving the same arrivals:

- a G-P/Priority-c system where each of the k servers runs at speed $1/k$, and
- a M/G/1/Priority-c system where the single server runs at speed 1.

We will refer to the total amount of remaining rank $\leq r$ work in the G-P/Priority-c system as $W^{\leq r}(t)$, and the total amount of remaining rank $\leq r$ work in the M/G/1/Priority-c system as $W^{\leq r}_{\text{M/G/1/Priority-}c}(t)$.

It suffices to show that at any time t, we have the following bound on the difference in the total amounts of remaining rank $\leq r$ work between the two systems:

$$W^{\leq r}(t) \leq W^{\leq r}_{\text{M/G/1/Priority-}c}(t) + \frac{2gkc^{r+2}}{c-1}. \tag{3.7}$$

To prove (3.7), we consider two cases:

(i) At least one server in the G-P/Priority-c system that has no remaining rank $\leq r$ work at time t.

(ii) All servers in the G-P/Priority-c system have remaining rank $\leq r$ work at time t.

In case (i), suppose server s' in the G-P/Priority-c system has no remaining rank $\leq r$ work at time t. By Lemma 3.5.1, we know that at all servers s,

$$W_s^{\leq r}(t) = W_s^{\leq r}(t) - W_{s'}^{\leq r}(t) \leq \frac{2gc^{r+2}}{c-1}.$$

Summing over all k servers implies (3.7).

We now turn to case (ii). Let t_0 be the most recent time before t when a G-P/Priority-c server had no remaining rank $\leq r$ work. Note that case (i) applies at time t_0. Therefore, it suffices to show that the difference in remaining rank $\leq r$ work between the two systems is no more at time t than at time t_0:

$$W^{\leq r}(t) - W^{\leq r}_{\text{M/G/1/Priority-}c}(t) \leq W^{\leq r}(t_0) - W^{\leq r}_{\text{M/G/1/Priority-}c}(t_0). \tag{3.8}$$

By definition of t_0, for the duration of entire time interval (t_0, t), each of the k servers in the G-P/Priority-c system processes rank $\leq r$ work at speed $1/k$, for a total of $t - t_0$ work. This is at least as much rank $\leq r$ work as the M/G/1/Priority-c system processes during (t_0, t), because the single server's speed is 1. Due to coupling, the two systems receive the same amount of rank $\leq r$ work during (t_0, t), implying (3.8). □

Proof of Lemma 3.5.3

Lemma 3.5.3. For any dispatching policy P, consider the dispatching policy G-P with tightness g. In a G-P/Priority-c system, the response time of a job of size x is stochastically bounded by

$$T(x) \leq_{\text{st}} B_{<r}\left(W^{\leq r}_{\text{M/G/1/Priority-}c} + \frac{(4c+2)gkc^{r+1}}{c-1} + kx\right),$$

where c is the guardrail rank width, $r = \lfloor \log_c x \rfloor$ is the rank of the job, and $B_{<r}(w)$ is the length of a busy period comprising only jobs of rank $< r$ started by work w.

Proof. Let

- j be the tagged job,
- x be j's size and $r = \lfloor \log_c x \rfloor$ be j's rank,
- a_j and d_j be j's arrival and departure times, respectively, and
- s be the server to which j is dispatched.

The time at which job j departs, d_j, can be calculated as the time required for server s to complete the following work:

- relevant work already present at time a_j, namely $W_s^{\leq r}(a_j)$;
- plus all relevant work that arrives at s while j is in the system, namely $V_s^{<r}(t) - V_s^{<r}(a_j)$;
- plus j's size, namely x.

Let $t \geq a_j$ be an arbitrary time while j is in the system. Because each server runs at speed $1/k$, the amount of work that server s has completed by time t is $(t - a_j)/k$. Writing

$$Z_s(t) = W_s^{\leq r}(a_j) + (V_s^{<r}(t) - V_s^{<r}(a_j))$$

gives the following expression for j's departure time d_j:

$$d_j = \inf\left\{ t \;\middle|\; \frac{t - a_j}{k} \geq Z_s(t) + x \right\}. \tag{3.9}$$

To bound d_j, we first bound $Z_s(t)$. Let

$$\overline{Z}(t) = \frac{1}{k}\sum_{s'} Z_{s'}(t) \tag{3.10}$$

be the average value of $Z_{s'}(t)$ over all servers s', and let

$$Z_s^{\text{maxdiff}}(t) = \max_{s'}(Z_s(t) - Z_{s'}(t))$$

be the maximum difference between $Z_s(t)$ and $Z_{s'}(t)$ over all servers s'. Observe that

$$Z_s(t) \le \overline{Z}(t) + Z_s^{\text{maxdiff}}(t).$$

Combining this with (3.9) gives a bound on d_j:

$$d_j \le \inf\left\{t \;\middle|\; \frac{t-a_j}{k} \ge \overline{Z}(t) + Z_s^{\text{maxdiff}}(t) + x\right\}. \tag{3.11}$$

To simplify (3.11), our next step is to bound $Z_s^{\text{maxdiff}}(t)$. We start by expanding $Z_s(t) - Z_{s'}(t)$:

$$Z_s(t) - Z_{s'}(t) = (W_s^{\le r}(a_j) - W_{s'}^{\le r}(a_j)) + (V_s^{<r}(t) - V_s^{<r}(a_j)) - (V_{s'}^{<r}(t) - V_{s'}^{<r}(a_j)).$$

We are left with an expression in terms of the rank $< r$ work dispatched to each server. We would like to turn this into an expression in terms of guardrail work counters, which will allow us to apply the constraints given by Definition 3.3.1. Consider the time interval (a_j, t). Job j is present at server s for the duration of the interval, so server s does not reset, implying

$$V_s^{<r}(t) - V_s^{<r}(a_j) = G_s^{<r}(t) - G_s^{<r}(a_j). \tag{3.12}$$

In contrast, server s' may reset during (a_j, t). When a reset occurs at some time t_{reset}, $G_{s'}^{<r}(t_{\text{reset}})$ decreases while $V_{s'}^{<r}(t_{\text{reset}})$ stays constant. Furthermore, $G_{s'}^{<r}(t')$ and $V_{s'}^{<r}(t')$ change in the same way at all other times t', so

$$V_{s'}^{<r}(t) - V_{s'}^{<r}(a_j) \ge G_{s'}^{<r}(t) - G_{s'}^{<r}(a_j). \tag{3.13}$$

Applying (3.12), (3.13), and Lemma 3.5.1 to (3.12) yields the bound

$$Z_s(t) - Z_{s'}(t) \le \frac{2gc^{r+2}}{c-1} + (G_s^{<r}(t) - G_s^{<r}(a_j)) - (G_{s'}^{<r}(t) - G_{s'}^{<r}(a_j)).$$

Because G-P is a guarded policy, we can apply the guardrail constraints from Definition 3.3.1:

$$\begin{aligned}
Z_s(t) - Z_{s'}(t) &\le \frac{2gc^{r+2}}{c-1} + (G_s^{<r}(t) - G_{s'}^{<r}(t)) - (G_s^{<r}(a_j) - G_{s'}^{<r}(a_j)) \\
&= \frac{2gc^{r+2}}{c-1} + \sum_{q=-\infty}^{r-1} (G_s^q(t) - G_{s'}^q(t)) + (G_{s'}^q(a_j) - G_s^q(a_j)) \\
&\le \frac{2gc^{r+2}}{c-1} + \sum_{q=-\infty}^{r-1} (gc^{q+1} + gc^{q+1}) \\
&= \frac{(2c+2)gc^{r+1}}{c-1}.
\end{aligned}$$

We have bounded $Z_s(t) - Z_{s'}(t)$ for arbitrary s' and hence bounded $Z_s^{\text{maxdiff}}(t)$. Substituting into (3.11) yields

$$d_j \leq \inf\left\{ t \;\middle|\; \frac{t - a_j}{k} \geq \overline{Z}(t) + \frac{(2c+2)gc^{r+1}}{c-1} + x \right\}.$$

Recalling the definition of $\overline{Z}(t)$ from (3.10) gives us

$$d_j \leq \inf\left\{ t \;\middle|\; t - a_j \geq W^{\leq r}(a_j) + (V^{<r}(t) - V^{<r}(a_j)) + \frac{(2c+2)gkc^{r+1}}{c-1} + kx \right\}.$$

Because the arrival process to the overall system is a Poisson process, we can rewrite this in terms of a "relevant" busy period, meaning one containing only jobs of rank $< r$:

$$d_j - a_j \leq B_{<r}\left(W^{\leq r}(a_j) + \frac{(2c+2)gkc^{r+1}}{c-1} + kx \right).$$

The Poisson arrival process also implies, by the PASTA property [235], that the amount of relevant work j sees on arrival, namely $W^{\leq r}(a_j)$, is drawn from the steady-state distribution, namely $W^{\leq r}$, so

$$T(x) \leq_{\text{st}} B_{<r}\left(W^{\leq r} + \frac{(2c+2)gkc^{r+1}}{c-1} + kx \right).$$

Applying Lemma 3.5.2 to $W^{\leq r}$ yields the desired bound. □

Remark 3.5.1. Note we can prove Lemmas 3.5.1 and 3.5.3 using only the following properties of resets:

- A server only resets when it is empty.
- When a server resets, its work counters do not increase.
- The guardrail constraints in Definition 3.3.1 continue to hold after each reset.

In particular, this means that resets are optional for proving our response time bounds, so the bounds hold even if the dispatcher chooses to omit some resets. This is helpful when implementing guarded dispatching policies in large systems (see Sections 3.8.1 and 3.8.2).

Proof of Theorem 3.4.1

Theorem 3.4.1. For any dispatching policy P, consider the policy G-P with tightness g. The expected response time for a job of size x under G-P/Priority-c is bounded by

$$\mathbb{E}\left[T(x)\right]^{\text{G-P/Priority-}c} \leq \frac{\frac{\lambda}{2}\int_0^{c^{r+1}} t^2 f_X(t)dt}{(1-\rho_{c^r})(1-\rho_{c^{r+1}})} + \frac{(4c+2)gk\frac{c^{r+1}}{c-1} + kx}{(1-\rho_{c^r})},$$

where

- $f_X(\cdot)$ is the probability density function of X,
- c is the guardrail rank width
- $r = \lfloor \log_c x \rfloor$ is the rank of a job of size x, and
- $\rho_y = \lambda \int_0^y t f_X(t)\, dt$ is the load due to jobs of size $\leq y$.

Proof. Recall the conclusion of Lemma 3.5.3,

$$T(x) \leq B_{<r}\left(W^{\leq r}_{\text{M/G/1/Priority-}c} + \frac{(4c+2)gkc^{r+1}}{c-1} + kx\right). \tag{3.14}$$

Standard results on busy periods [104] state that

$$\mathbb{E}\left[B_{<r}(Y)\right] = \frac{\mathbb{E}\left[Y\right]}{1-\rho_{c^r}},$$

and standard results on the single-server Priority-c system[104] give the expected steady-state remaining rank $\leq r$ work:

$$\mathbb{E}\left[W^{\leq r}_{\text{M/G/1/Priority-}c}\right] = \frac{\frac{\lambda}{2}\int_0^{c^{r+1}} t^2 f_X(t)dt}{(1-\rho_{c^{r+1}})}.$$

Taking expectations of (3.14) and applying these standard results yields the desired bound. □

We will also need a lemma relating the single-server Priority-c system to the single-server PSJF system:

Lemma 3.5.4. *For any job size distribution, the mean response time of a single-server Priority-c system is no more than $c+2\sqrt{c-1}$ times the mean response time of a single-server PSJF system:*

$$\mathbb{E}\left[T\right]^{\text{Priority-}c} \leq (c+2\sqrt{c-1})\mathbb{E}\left[T\right]^{\text{PSJF}}.$$

Proof. We will consider a new random variable, D, the delay due to a job. D is defined for scheduling policies that assign every job a fixed priority, like Priority-c and PSJF. For a given job j of size x, D_j is

- the amount j delays other jobs, namely x times the number of jobs with lower priority than j in the system when j arrives,
- plus the amount other jobs that arrived before j delay j, namely the total remaining size of jobs with higher priority than j in the system when j arrives,
- plus j's size.

Note that the response time of a job ℓ is equal to ℓ's size, plus the amount ℓ is delayed by jobs that arrived before ℓ, plus the amount ℓ is delayed by jobs that arrive after ℓ. Each of those amounts of time is accounted for in the delay of exactly one job. As a result, the sum of the delays of the jobs in a busy period equals the sum of the response times of those jobs. Therefore, in steady state, mean response time and mean delay are equal:

$$\mathbb{E}\left[T\right] = \mathbb{E}\left[D\right].$$

Therefore, it suffices to show that

$$\mathbb{E}\left[D\right]^{\text{Priority-}c} \leq (c+2\sqrt{c-1})\mathbb{E}\left[D\right]^{\text{PSJF}}.$$

Let us consider a pair of coupled systems receiving the same arrivals: A single-server system with PSJF scheduling, and a single-server system with Priority-c scheduling.

Note that PSJF and Priority-c both prioritize all jobs of lower ranks over all jobs of higher ranks. As a result, both coupled systems will server jobs of the same ranks at the same times, and will always have the same amount of remaining work of each rank.

Let us consider the expected delay due to a particular "tagged" job j, arriving to a steady-state system. The expected delay due to j is equal to each system's mean delay by the PASTA property [235]. Let j be a job of size x with rank $r = \log_c x$.

The delay due to j, D_j, is a summation over each job in the system at the moment j arrives. Let D_j^q be the delay caused by the interaction of j and jobs of rank q that are in the system when j arrives, including j's size in D_j^r. Then we can write D_j in terms of the D_j^qs:

$$D_j = \sum_{q=-\infty}^{\infty} D_j^q.$$

Therefore, it suffices to show for all ranks q that

$$\mathbb{E}\left[D_j^q\right]^{\text{Priority-}c} \leq (c + 2\sqrt{c-1})\mathbb{E}\left[D_j^q\right]^{\text{PSJF}}. \tag{3.15}$$

Let $X_{j'}$ denote the original size of a job j', and let $R_{j'}$ denote the remaining size of j'. Let J^q denote the set of jobs of rank q in the system at the time j arrives. Let N^q denote the number of jobs of rank q in the system at the time j arrives.

We now consider three cases: $q < r$, $q = r$, and $q > r$.

Case 1: $q < r$. Because $q < r$, all jobs in rank q have higher priority than j. As a result, under both PSJF and Priority-c, D_j^q is equal to the total remaining size of jobs of rank q:

$$D_j^q = \sum_{j' \in J^q} R_{j'}.$$

As noted above, this is equal in the two systems due to the coupling. This proves (3.15) in this case.

Case 2: $q = r$. Because Priority-c uses First-Come-First-Served scheduling within a rank, D_j^r in the Priority-c system is equal to the total remaining size of jobs of rank q:

$$D_j^{r\,(\text{Priority-}c)} = \sum_{j' \in J^{r\,(\text{Priority-}c)}} R_{j'}.$$

In contrast, D_j^r in the PSJF system is equal to the total remaining size of jobs of rank r with size at most x, plus x times the number of jobs of rank r with size more than x:

$$D_j^{r\,(\text{PSJF})} = \sum_{j' \in J^{r\,(\text{PSJF})} | X_{j'} \leq x} R_{j'} \quad + \sum_{j' \in J^{r\,(\text{PSJF})} | X_{j'} > x} x.$$

Noting that $x \geq c^r$ and the remaining size of any job of rank r is at most c^{r+1}, we can lower

bound $D_j^{r\,(\text{PSJF})}$:

$$\begin{aligned} D_j^{r\,(\text{PSJF})} &\geq \sum_{j' \in J^{r\,(\text{PSJF})} | X_{j'} \leq x} R_{j'} \quad + \sum_{j' \in J^{r\,(\text{PSJF})} | X_{j'} > x} c^r \\ &\geq \sum_{j' \in J^{r\,(\text{PSJF})} | X_{j'} \leq x} R_{j'} \quad + \sum_{j' \in J^{r\,(\text{PSJF})} | X_{j'} > x} \frac{R_{j'}}{c} \\ &\geq \frac{1}{c} \sum_{j' \in J^{r\,(\text{PSJF})}} R_{j'}. \end{aligned}$$

As noted above, $\sum_{j' \in J^q} R_{j'}$ is equal in both systems. As a result,

$$\mathbb{E}\left[D_j^q\right]^{\text{Priority-}c} \leq c\mathbb{E}\left[D_j^q\right]^{\text{PSJF}},$$

which proves (3.15) in this case.

Case 3: $q > r$. Because $q > r$, all jobs in rank q have lower priority that j. As a result, in both systems,

$$D_j^q = xN^q.$$

Also, note that x is independent of N^q. Therefore, we simply need to show that

$$\mathbb{E}\left[N^q\right]^{\text{Priority-}c} \leq (c + 2\sqrt{c-1})\mathbb{E}\left[N^q\right]^{\text{PSJF}}.$$

However, it is possible for there to be twice as many jobs of rank q in the Priority-c system as in the PSJF system, regardless of the value of c. In particular, there could be two rank q jobs in the Priority-c system and one rank q job in the PSJF system, if one of the rank q jobs in the Priority-c system has very little remaining size.

Let j^{old} be the oldest job of rank q in the Priority-c system at a given time. Because Priority-c serves jobs in FCFS order, only j^{old} has been processed, so only j^{old} can have remaining size under c^q. Therefore, we can bound the total remaining size of the rank q jobs in the Priority-c system:

$$\sum_{j' \in J^{q\,(\text{Priority-}c)}} R_{j'} \quad \geq R_{j^{\text{old}}} + c^q(N^{q\,(\text{Priority-}c)} - 1).$$

Likewise, we can bound the total remaining size of the rank q jobs in the PSJF system:

$$\sum_{j' \in J^{q\,(\text{PSJF})}} R_{j'} \leq c^{q+1} N^{q\,(\text{PSJF})}.$$

Using the fact that $\sum_{j' \in J^q} R_{j'}$ is equal in both systems gives us

$$R_{j^{\text{old}}} + c^q(N^{q\,(\text{Priority-}c)} - 1) \leq c^{q+1} N^{q\,(\text{PSJF})},$$

which rearranges to

$$N^{q\,(\text{Priority-}c)} \leq cN^{q\,(\text{PSJF})} + 1 - \frac{R_{j^{\text{old}}}}{c^q}.$$

This implies $N^{q(\text{Priority-}c)} \leq cN^{q(\text{PSJF})} + 1$, which allows us to relate the expected numbers of jobs in the systems:

$$\mathbb{E}\left[N^{q(\text{Priority-}c)}\right] \leq c\mathbb{E}\left[N^{q(\text{PSJF})}\right] + \Pr\{N^{q(\text{Priority-}c)} > cN^{q(\text{PSJF})}\}.$$

Recall that $N^{q(\text{Priority-}c)}$ and $N^{q(\text{PSJF})}$ are both integers. This means that if $N^{q(\text{Priority-}c)} > cN^{q(\text{PSJF})}$, then

$$N^{q(\text{Priority-}c)} \geq N^{q(\text{PSJF})} + 1.$$

As a result, if $N^{q(\text{Priority-}c)} > cN^{q(\text{PSJF})}$, then

$$N^{q(\text{PSJF})} + 1 \leq cN^{q(\text{PSJF})} + 1 - \frac{R_{j^{\text{old}}}}{c^q},$$

meaning that

$$R_{j^{\text{old}}} \leq (c-1)c^q N^{q(\text{PSJF})}. \tag{3.16}$$

Therefore, we will show that either $\Pr\{N^{q(\text{Priority-}c)} > cN^{q(\text{PSJF})}\}$ is small or $\mathbb{E}\left[N^{q(\text{PSJF})}\right]$ is large. In either case, (3.16) will imply the desired bound (3.15).

Let us condition on $R_{j^{\text{old}}}$. Because j is a Poisson arrival, j sees a time-average state of $R_{j^{\text{old}}}$. The (stochastically) smallest this distribution can be is the uniform distribution on $[0, c^q]$, because the original size of a rank q job is at least c^q. In particular, for any ℓ,

$$\Pr\{R_{j^{\text{old}}} < \ell\} \leq \frac{\ell}{c^q}.$$

Let ρ_q be the probability that j sees a rank q job in the system on arrival. Let m be the largest integer such that

$$\Pr\{N^{q(\text{Priority-}c)} > cN^{q(\text{PSJF})}\} \geq m(c-1)\rho_q.$$

Then (3.16) implies

$$\Pr\{R_{j^{\text{old}}} \leq c^q(c-1)N^{q(\text{PSJF})}\} \geq m(c-1)\rho_q. \tag{3.17}$$

Regardless of the correlation between $N^{q(\text{PSJF})}$ and $R_{j^{\text{old}}}$, we must have

$$\Pr\{N^{q(\text{PSJF})} \geq m\} \geq (c-1)\rho_q. \tag{3.18}$$

This is because if $N^{q(\text{PSJF})} \leq m-1$ on a particular arrival, and also $N^{q(\text{Priority-}c)} > cN^{q(\text{PSJF})}$, then by (3.16),

$$R_{j^{\text{old}}} \leq (c-1)c^q(m-1). \tag{3.19}$$

But an arrival only sees job j^{old} in the system at all with probability ρ_q. Moreover, conditional on seeing a job j^{old} at all, the arrival observes (3.19) with probability at most $(c-1)(m-1)$, because every rank q job has size at least c^q. This means

$$\Pr\{R_{j^{\text{old}}} \leq (c-1)c^q(m-1)\} \leq (m-1)(c-1)\rho_q,$$

which together with (3.17) implies (3.18).

By a similar argument as (3.18),

$$\Pr\{N^{q(\text{PSJF})} \geq m - z\} \geq (z+1)(c-1)\rho_q$$

for any integer $z < m$.

In addition, if there is a job in the Priority-c system then there is a job in the PSJF system:

$$\Pr\{N^{q(\text{PSJF})} \geq 1\} \geq \rho_q.$$

We can combine these bounds on the probability of $N^{q(\text{PSJF})}$ taking specific values to a derive a bound on its expectation:

$$\begin{aligned}
\mathbb{E}\left[N^{q(\text{PSJF})}\right] &\geq \left(\sum_{z=0}^{m-1}(m-z)(c-1)\rho_q\right) + (\rho_q - \rho_q m(c-1)) \\
&= \left(\left(\sum_{z=0}^{m-1}(m-z-1)(c-1)\right)+1\right)\rho_q \\
&= \left(1 + \frac{m}{2}(m-1)(c-1)\right)\rho_q.
\end{aligned}$$

We are now ready to show that either $\Pr\{N^{q(\text{Priority-}c)} > cN^{q(\text{PSJF})}\}$ is small or $\mathbb{E}\left[N^{q(\text{PSJF})}\right]$ is large by bounding their ratio for any value of m:

$$\begin{aligned}
\frac{\Pr\{N^{q(\text{Priority-}c)} > cN^{q(\text{PSJF})}\}}{\mathbb{E}\left[N^{q(\text{PSJF})}\right]} &\leq \frac{(m+1)(c-1)\rho_q}{(1+\frac{m}{2}(m-1)(c-1))\rho_q} \\
&= \frac{m+1}{\frac{1}{c-1} + \frac{m}{2}(m-1)}.
\end{aligned}$$

This expression is maximized when

$$m = \sqrt{\frac{2c}{c-1}} - 1,$$

so

$$\begin{aligned}
\frac{\Pr\{N^{q(\text{Priority-}c)} > cN^{q(\text{PSJF})}\}}{\mathbb{E}\left[N^{q}\right]^{(\text{PSJF})}} &\leq \frac{\sqrt{\frac{2c}{c-1}}}{\frac{1}{c-1} + (\sqrt{\frac{c}{c-1}} - \frac{1}{2})(\sqrt{\frac{2c}{c-1}} - 2)} \\
&= \frac{1}{\sqrt{\frac{2c}{c-1}} - \frac{3}{2}} \\
&\leq 2\sqrt{c-1}.
\end{aligned}$$

where the final bound holds due to the fact that $1 < c \leq 2$, by the definition of c. Combining this result with (3.16) yields

$$\mathbb{E}\left[N^{q(\text{Priority-}c)}\right] \leq (c + 2\sqrt{c-1})\mathbb{E}\left[N^{q(\text{PSJF})}\right],$$

which is (3.15), as desired. □

3.5.4 Asymptotic Behavior of Guarded Policies

Theorem 3.4.2. Consider a single-server SRPT system whose single server is k times as fast as each server in the load balancing system. For any dispatching policy P, consider the policy G-P with any constant tightness. Then for any size distribution X such that $\mathbb{E}\left[X^2 \log^2 X \mathbb{1}\{X > 1\}\right] < \infty$, the mean response times of G-P/SRPT, G-P/Priority-c, and (single-server) SRPT converge as load approaches capacity:

$$\lim_{\rho \to 1} \frac{\mathbb{E}\left[T\right]^{\text{G-P/SRPT}}}{\mathbb{E}\left[T\right]^{\text{SRPT}}} = \frac{\mathbb{E}\left[T\right]^{\text{G-P/Priority-}c}}{\mathbb{E}\left[T\right]^{\text{SRPT}}} = 1.$$

Proof. We start with the bound on the mean response time of G-P/Priority-c from Theorem 3.4.1:

$$\mathbb{E}\left[T(x)\right]^{\text{G-P/Priority-}c} \leq \frac{\frac{\lambda}{2}\int_0^{c^{r+1}} t^2 f_X(t)dt}{(1-\rho_{c^r})(1-\rho_{c^{r+1}})} + \frac{(4c+2)gk\frac{c^{r+1}}{c-1} + kx}{(1-\rho_{c^r})}.$$

Note that the first term in this expression also appears in the expression for mean response time under single-server Priority-c [104]:

$$\mathbb{E}\left[T(x)\right]^{\text{Priority-}c} = \frac{\frac{\lambda}{2}\int_0^{c^{r+1}} t^2 f_X(t)dt}{(1-\rho_{c^r})(1-\rho_{c^{r+1}})} + \frac{x}{(1-\rho_{c^r})}.$$

Therefore, let us simplify (3.20):

$$\mathbb{E}\left[T(x)\right]^{\text{G-P/Priority-}c} \leq \mathbb{E}\left[T(x)\right]^{\text{Priority-}c} + \frac{(4c+2)gk\frac{c^{r+1}}{c-1} + (k-1)x}{(1-\rho_{c^r})}.$$

We may simplify this bound by combining constant terms. Note that $c \leq 2$ and $x \geq c^r$. Let $m = 20gk + k - 1$. Then

$$mx \geq (4c+2)gkc^{r+1} + (k-1)(c-1)x.$$

Thus, we may simplify (3.20) further:

$$\mathbb{E}\left[T(x)\right]^{\text{G-P/Priority-}c} \leq \mathbb{E}\left[T(x)\right]^{\text{Priority-}c} + \frac{mx}{(c-1)(1-\rho_{c^r})}.$$

We want to relate this to something more similar to the mean response time under SRPT. A convenient policy for comparison with Priority-c that is similar enough to SRPT is Preemptive-Shortest-Job-First (PSJF), which prioritizes jobs according to their original size.

We now use Lemma 3.5.4 which says that

$$\mathbb{E}\left[T\right]^{\text{Priority-}c} \leq (c + 2\sqrt{c-1})\mathbb{E}\left[T\right]^{\text{PSJF}}.$$

For brevity, let

$$b(c) = (c + 2\sqrt{c-1}).$$

Using Lemma 3.5.4, we find that

$$\mathbb{E}\left[T(x)\right]^{\text{G-P/Priority-}c} \leq b(c)\mathbb{E}\left[T\right]^{\text{PSJF}} + \frac{mx}{(c-1)(1-\rho_{c^r})}.$$

Note that $x \geq c^r$, so $\rho_x \geq \rho_{c^r}$, so

$$\mathbb{E}\left[T(x)\right]^{\text{G-P/Priority-}c} \leq b(c)\mathbb{E}\left[T\right]^{\text{PSJF}} + \frac{mx}{(c-1)(1-\rho_x)}.$$

Based on standard results on mean response time under PSJF and SRPT [104], we know that

$$\mathbb{E}\left[T(x)\right]^{\text{PSJF}} \leq \mathbb{E}\left[T(x)\right]^{\text{SRPT}} + \frac{x}{1-\rho_x}.$$

Let $m' = m + 4$. Because $c \leq 2$, we know $b(c) \cdot (c-1) \leq 4$. Thus,

$$\mathbb{E}\left[T(x)\right]^{\text{G-P/Priority-}c} \leq b(c)\mathbb{E}\left[T(x)\right]^{\text{SRPT}} + \frac{m'x}{(c-1)(1-\rho_x)}. \tag{3.20}$$

Next, we take the expectation of (3.20) over all job sizes x. To do so, we need to integrate

$$\int_0^\infty \frac{x}{1-\rho_x} f_X(x)dx,$$

where $f_X(\cdot)$ is the probability density function of X. To compute the integral, we make a change of variables from x to ρ_x, using the following facts:

$$\begin{aligned}
\rho_x &= \int_0^x \lambda t f_X(t)dt \\
\frac{d\rho_x}{dx} &= \lambda x f_X(x) \\
\rho_0 &= 0 \\
\lim_{x\to\infty} \rho_x &= \rho.
\end{aligned}$$

Given this change of variables, we compute

$$\int_0^\infty \frac{x}{1-\rho_x} f_X(x)dx = \int_0^\rho \frac{1}{\lambda(1-\rho_x)} d\rho_x = \frac{1}{\lambda}\ln\left(\frac{1}{1-\rho}\right).$$

Applying this to (3.20), we find that

$$\mathbb{E}\left[T\right]^{\text{G-P/Priority-}c} \leq b(c)\mathbb{E}\left[T\right]^{\text{SRPT}} + \frac{m'}{\lambda(c-1)}\ln\left(\frac{1}{1-\rho}\right).$$

Dividing through by $\mathbb{E}\left[T\right]^{\text{SRPT}}$, we find that

$$\frac{\mathbb{E}\left[T\right]^{\text{G-P/Priority-}c}}{\mathbb{E}\left[T\right]^{\text{SRPT}}} \leq b(c) + \frac{m' \ln \frac{1}{1-\rho}}{\lambda(c-1)\mathbb{E}\left[T\right]^{\text{SRPT}}}.$$

Plugging in the value of c in terms of ρ from (3.2),

$$\frac{\mathbb{E}\left[T\right]^{\text{G-P/Priority-}c}}{\mathbb{E}\left[T\right]^{\text{SRPT}}} \leq b(c) + \frac{m' \cdot (\ln^2 \frac{1}{1-\rho} + \ln \frac{1}{1-\rho})}{\lambda \mathbb{E}\left[T\right]^{\text{SRPT}}}.$$

We now take the limit of the above ratio as $\rho \to 1$. In this limit,

- $\ln \frac{1}{1-\rho}$ diverges,
- λ approaches $\mathbb{E}\left[X\right]$, and
- c approaches 1, so $b(c)$ also approaches 1:

$$\lim_{c \to 1+} c + 2\sqrt{c-1} = 1.$$

Therefore, letting $m'' = 2m'/\mathbb{E}\left[X\right]$,

$$\lim_{\rho \to 1} \frac{\mathbb{E}\left[T\right]^{\text{G-P/Priority-}c}}{\mathbb{E}\left[T\right]^{\text{SRPT}}} \leq \lim_{\rho \to 1} \left(1 + \frac{m'' \ln^2 \frac{1}{1-\rho}}{\mathbb{E}\left[T\right]^{\text{SRPT}}}\right). \tag{3.21}$$

Recall now that we assume that $\mathbb{E}\left[X^2 \log^2 X \mathbb{1}\{X > 1\}\right] < \infty$. By Lemma 3.5.5, this implies that

$$\lim_{\rho \to 1} \frac{\ln^2 \frac{1}{1-\rho}}{\mathbb{E}\left[T\right]^{\text{SRPT}}} = 0.$$

Applying this to (3.21), we find that

$$\lim_{\rho \to 1} \frac{\mathbb{E}\left[T\right]^{\text{G-P/Priority-}c}}{\mathbb{E}\left[T\right]^{\text{SRPT}}} \leq 1. \tag{3.22}$$

SRPT yields optimal mean response time over all single-server policies [194], and a joint dispatching/scheduling policy can be emulated on a single server, so (3.22) is in fact an equality, as desired.

The optimality of SRPT's mean response time also implies that the mean response time under G-P/SRPT is no more than the mean response time under G-P/Priority-c. As a result,

$$\lim_{\rho \to 1} \frac{\mathbb{E}\left[T\right]^{\text{G-P/SRPT}}}{\mathbb{E}\left[T\right]^{\text{SRPT}}} = 1. \qquad \square$$

3.5.5 Bounding SRPT response time

We need to bound the mean response time of SRPT in the M/G/1. We need a result similar to Lemma 2.7.1, but slightly stronger. We prove that result here, paralleling the proof of Lemma 2.7.1.

Lemma 3.5.5. *In an M/G/1 with any service requirement distribution S such that $\mathbb{E}[S^2 \log^2 S \mathbb{1}\{S > 1\}]$ is finite,*

$$\lim_{\rho \to 1} \frac{\ln^2 \frac{1}{1-\rho}}{\mathbb{E}\left[T\right]^{\text{SRPT}}} = 0.$$

Proof. We start with a result from the literature on the performance of SRPT-1 in the heavy-traffic limit [138].

$$\mathbb{E}\left[T_1\right] = \Omega\left(\frac{1}{(1-\rho)\, G^{-1}(\rho)}\right),$$

where $G(r) = \mathbb{E}\left[S\mathbb{1}S \le r\right]/\mathbb{E}\left[S\right]$.

We therefore want to show

$$\frac{1}{G^{-1}(\rho)} = \omega\left((1-\rho)\log^2\frac{1}{1-\rho}\right),$$

which amounts to showing that in the $r \to \infty$ limit,

$$(1-G(r))\log^2\frac{1}{1-G(r)} = o\left(\frac{1}{r}\right).$$

It actually suffices to show $1 - G(r) = o(1/(r\log^2 r))$, as then

$$(1-G(r))\log^2\frac{1}{1-G(r)} = o\left(\frac{\log^2 r + \log^2\log^2 r}{r\log^2 r}\right) = o\left(\frac{1}{r}\right).$$

We now use the $\mathbb{E}\left[S^2\log^2 S\mathbb{1}\{S>1\}\right] < \infty$ assumption to prove $1-G(r) = o(1/(r\log^2 r))$. The first step is to express $1-G(r)$ in terms of S and S_e:

$$1 - G(r) = \frac{\mathbb{E}\left[S\mathbb{1}\left\{S>r\right\}\right]}{\mathbb{E}\left[S\right]} = \frac{\mathbb{E}\left[(S-r)^+\right]}{\mathbb{E}\left[S\right]} = \frac{r}{\mathbb{E}\left[S\right]}\mathbb{P}\left\{S>r\right\} + \mathbb{P}\left\{S_e>r\right\}.$$

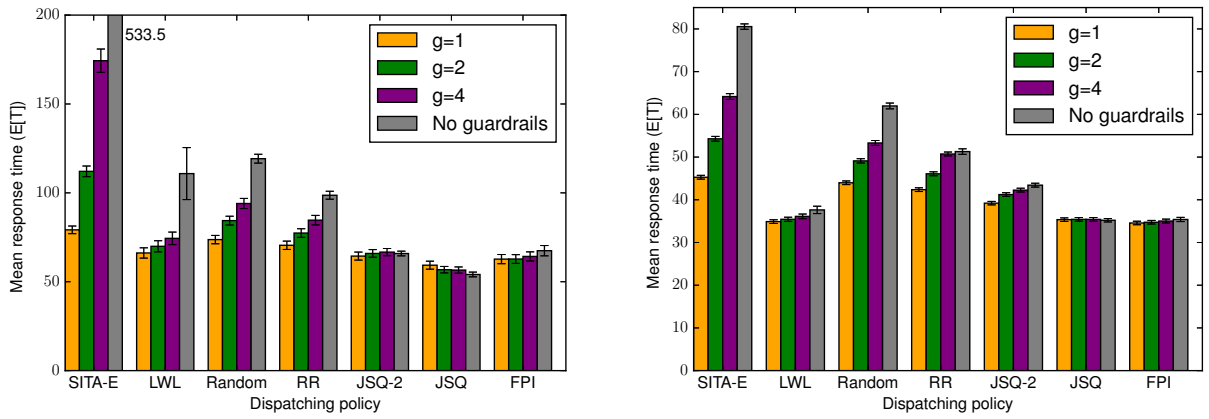
We show that both terms are $o(1/(r\log r))$. For the first term, finiteness of $\mathbb{E}\left[S^2\log^2 S\right]$ implies

$$\begin{aligned} r^2\log^2 r\mathbb{P}\left\{S>r\right\} &\le \mathbb{E}\left[S^2\log^2 S\mathbb{1}\left\{S>r\right\}\right] \\ &= \mathbb{E}\left[S^2\log^2 S\right] - \mathbb{E}\left[S^2\log^2 S\mathbb{1}\left\{S\le r\right\}\right] = o(1). \end{aligned}$$

We can bound the second term similarly because

$$\begin{aligned} \mathbb{E}\left[S\right]\mathbb{E}\left[S_e\log^2 S_e\mathbb{1}\{S_e>1\}\right] &= \mathbb{E}\left[\mathbb{1}\left\{S>1\right\}\int_1^S s\log^2 s\, ds\right] \\ &= \mathbb{E}\left[\mathbb{1}\left\{S>1\right\}\left(\frac{1}{2}S^2\log^2 S - \frac{1}{4}S^2\log S + \frac{1}{4}\right)\right] < \infty. \ \square \end{aligned}$$

Throughout this chapter, the $\mathbb{E}[S^2\log^2 S] < \infty$ assumption can be replaced by a $\mathbb{E}[S^2\log^{1+\epsilon} S] < \infty$ assumption, for any $\epsilon > 0$, by redefining c as $c = \frac{1}{1+\frac{1}{1+\ln^\epsilon(1-\rho)}}$. In the above proof, we would then replace 2 by $1+\epsilon$.



(a) $\rho = 0.98$ (b) $\rho = 0.80$

Figure 3.6: At heavy load $\rho = 0.98$, adding guardrails significantly reduces the mean response times of SITA-E, LWL, Random, and RR, while leaving the mean response times of JSQ, JSQ-2, and FPI nearly unchanged. At more moderate load $\rho = 0.8$, adding guardrails significantly reduces the mean response times of SITA-E, Random, RR, and JSQ-2 while leaving the mean response times of LWL, JSQ, and FPI nearly unchanged. The smallest tightness, $g = 1$, shows the best performance for all policies except JSQ and FPI, where it doesn't really matter. Simulation uses $k = 10$ servers. Size distribution shown is Bounded Pareto with $\alpha = 1.5$ and range $[1, 10^6]$. $C^2 \sim 333$. 40 trials simulated, 95% confidence intervals shown.

3.5.6 Optimality of Guarded Policies

As a simple corollary of Theorem 3.4.2, we find that for any dispatching policy P, G-P/SRPT has optimal mean response time in the heavy traffic limit over all joint dispatching/scheduling policies.

Corollary 3.4.1. For any dispatching policy P consider the policy G-P with any constant tightness. Consider any joint dispatching/scheduling policy P′/S′. Then for any size distribution X such that $\mathbb{E}\left[X^2 \log^2 X \mathbb{1}\{X > 1\}\right] < \infty$, the mean response times of G-P/SRPT and G-P/Priority-c are at least as small as the mean response time of P′/S′ as load approaches capacity:

$$\lim_{\rho \to 1} \frac{\mathbb{E}\left[T\right]^{\text{G-P/SRPT}}}{\mathbb{E}\left[T\right]^{\text{P}'/\text{S}'}} = \frac{\mathbb{E}\left[T\right]^{\text{G-P/Priority-}c}}{\mathbb{E}\left[T\right]^{\text{P}'/\text{S}'}} \leq 1.$$

Proof. SRPT has optimal mean response time among all single-server policies [194], and any joint dispatching/scheduling policy can be emulated on a single server, so $\mathbb{E}\left[T\right]^{\text{SRPT}} \leq \mathbb{E}\left[T\right]^{\text{P}'/\text{S}'}$. The result thus follows from Theorem 3.4.2. □

3.6 Simulation

We have shown that in heavy traffic, adding guardrails to any dispatching policy gives it optimal mean response time. We now turn to investigating loads outside the heavy-traffic regime. While

the mean response time upper bound in Theorem 3.4.1 holds for all loads, it is only tight in the heavy-traffic limit. We therefore focus on simulation.

We consider the following dispatching policies, each paired with SRPT scheduling at the servers:

Random The policy which dispatches each job to a uniformly random server.

Round-Robin (RR) The policy which dispatches each job to the server which least recently received a job.

Least-Work-Left (LWL) The policy which dispatches each job to the server with the least remaining work.

Size-Interval-Task-Assignment (SITA) The policy which classifies jobs into size intervals (small, medium, large, etc.) and dispatches all small jobs to one server, all medium jobs to another server, etc. Specifically, we simulate **SITA-E**, the SITA policy which chooses the size intervals to equalize the expected load at each server.

Join-Shortest-Queue (JSQ) The policy which dispatches each job to the server with the fewest jobs present.

Join-Shortest-Queue-*d* (JSQ-*d*) The policy which samples d uniformly random servers on each arrival and dispatches the job to the server with the fewest jobs present among those d. We focus on the $d = 2$ case.

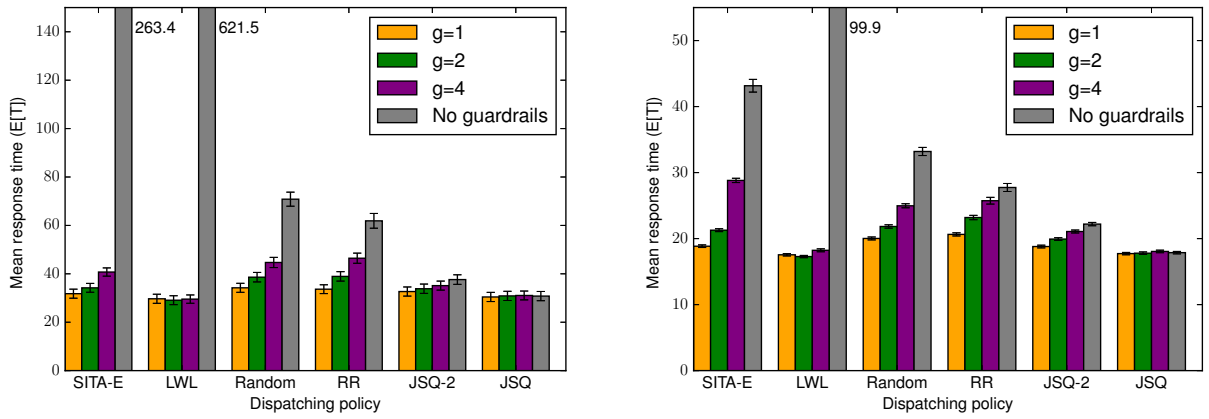
First Policy Iteration (FPI) The first policy iteration heuristic, as described by Hyytiä et al. [118] in the setting of dispatching to SRPT servers. FPI dispatches each job to the server that would be optimal if all jobs thereafter were dispatched randomly. Hyytiä et al.'s derivation of the FPI policy assumes that the job size distribution is continuous, and specifically that two different jobs almost surely have different sizes. As a result, we only implement the FPI policy for the Bounded Pareto distribution, shown in Figure 3.6.

3.6.1 Simulation Results

Figures 3.6 and 3.7 show the mean response time under all of the above dispatching policies with SRPT scheduling at the servers. We omit the FPI policy from Figure 3.7, because that simulation's job size distribution is not continuous, and Hyytiä et al. do not derive the FPI policy for such distributions [118]. We also show 95% confidence intervals for each mean response time. We consider two different job size distributions: a Bounded Pareto distribution (see Figure 3.6) and a Bimodal distribution (see Figure 3.7). In each case we show (a) very heavy traffic ($\rho = 0.98$) and (b) more moderate traffic ($\rho = 0.8$). We augment each dispatching policy with guardrails of varying tightness, $g = 1, 2,$ and 4.

The high-level message seen in Figures 3.6 and 3.7 is that adding guardrails to dispatching policies can greatly reduce their response times, even at more moderate loads. Simple dispatching policies like Random and RR improve by $15 - 40\%$ when $\rho = 0.8$ and $30 - 50\%$ when $\rho = 0.98$, in the figures shown. Other policies, like LWL and SITA-E, show even more dramatic improvement for certain job size distributions. We find using tightness $g = 1$ is generally best.

The FPI heuristic of Hyytiä et al. [118] performs about equally well with or without guardrails. Figure 3.6 shows that adding guardrails to FPI yields a slight reduction in mean response time at



(a) $\rho = 0.98$ (b) $\rho = 0.80$

Figure 3.7: At heavy load $\rho = 0.98$, adding guardrails significantly reduces the mean response time of SITA-E, LWL, Random, and RR, while leaving the mean response times of JSQ and JSQ-2 nearly unchanged. At more moderate load $\rho = 0.8$, adding guardrails significantly reduces the mean response times of SITA-E, LWL, Random, RR, and JSQ-2, while leaving the mean response time of JSQ nearly unchanged. The smallest tightness, $g = 1$, shows the best performance for all policies except JSQ, where it doesn't really matter. Simulation uses $k = 10$ servers. Size distribution shown is Bimodal with 99.95% size 1 jobs and 0.05% size 1000 jobs. $C^2 \sim 221$. 40 trials simulated, 95% confidence intervals shown.

$\rho = 0.98$, and has essentially no effect at $\rho = 0.8$. The FPI heuristic performs well in simulation, but its only theoretical guarantee is it outperforms Random. Applying guardrails guarantees optimal mean response time in heavy traffic, while maintaining or improving performance in simulation.

We observe that the JSQ dispatching policy performs well even without guardrails. In fact, guardrails can be seen as helping all the other dispatching policies to improve their performance to approach JSQ. We do not know of any guarantees on JSQ's performance with SRPT servers, even under heavy traffic, unless we add guardrails to JSQ. Figures 3.6 and 3.7 show that adding guardrails to JSQ does not affect its performance much.

In Section 3.7, we simulate guarded policies under a variety of alternative system conditions. We simulate systems with more servers, systems with lighter load and systems with different job size distributions.

3.6.2 Simulation Discussion and Intuition

Recall from Section 3.3.1 the intuition behind guardrails: Guardrails force the dispatching policy to spread out small jobs across all of the servers. Guardrails thereby ensure that the maximum possible number of servers are working on the smallest jobs available.

Let us consider this intuition in light of each of the dispatching policies. SITA-E does the opposite of spreading small jobs: It clumps the smallest jobs onto the same server. Therefore, G-SITA-E shows a massive improvement. In particular, we want g to be as low as possible ($g = 1$), corresponding to the greatest guardrail control, to prevent SITA-E from doing what it

was designed to do.

Random and RR are better at spreading jobs naturally, but they still make mistakes. In particular, Random and RR do not differentiate between jobs of different sizes and they do not observe the state of the servers, so they only spread out the small jobs by chance. As a result, G-Random and G-RR show sizable improvements. The tightest guardrails ($g = 1$) increase the spread of the small jobs the most, and hence show the most improvement.

LWL does not spread out the small jobs. In fact, one huge job at a server can keep away all of the small jobs for a long time. However, LWL is so efficient at using its servers that one does not experience its shortcomings unless both load and job size variability are high. Under those circumstances, LWL has very high mean response times. At load $\rho = 0.98$ with the Bimodal size distribution shown in Figure 3.7, LWL's mean response time is 7 times worse than that of Random. Guardrails are particularly effective in these situations where LWL fails because they force small jobs onto the servers with one large job. The tightest guardrails ($g = 1$) force small jobs onto those servers most aggressively and hence show the lowest mean response time in Figures 3.6 and 3.7.

3.6.3 Comparing Simulation to Analytical Bounds

In Theorem 3.4.1 we established an analytical upper bound on the mean response time of any guarded policy. This bound is tight in the limit as load $\rho \to 1$, and implies the heavy traffic optimality of any guarded policy.

However, this bound is not tight under the more moderate loads simulated in this section. Under the system conditions shown in Figure 3.6, at load $\rho = 0.8$ and guardrail tightness $g = 1$, Theorem 3.4.1 implies that any guarded policy has mean response time at most 350, and that at load $\rho = 0.98$ the mean response time is at most 700. The actual performance of guarded policies is much better than this, as shown in Figure 3.6. Tightening our bound is a potential direction for future research.

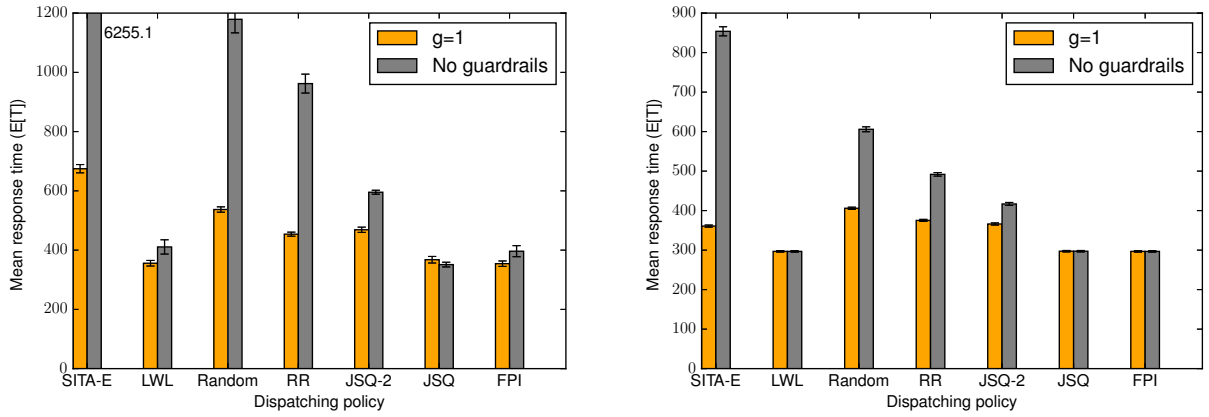
3.7 Additional Simulations

We include here some additional simulation results covering a wider range of cases than Section 3.6. We only show the tightest guardrails ($g = 1$), as those guardrails generally yield the lowest mean response times. We vary the parameters of the simulations in from Section 3.6 in three different ways:

- adding many more servers (Figures 3.8 and 3.9),
- decreasing load (Figure 3.10), and
- varying the job size distribution (Figures 3.11 and 3.12).

In nearly every case guardrails improve or at least do not degrade mean response time of the underlying policy. In particular, as a general rule, G-LWL is nearly always tied for minimum mean response time among all dispatching policies simulated.

We omit the FPI heuristic from simulations involving the Bimodal job size distribution because Hyttiä et al. [118] only derive the policy for continuous job size distributions. We omit the



(a) $\rho = 0.98$ (b) $\rho = 0.80$

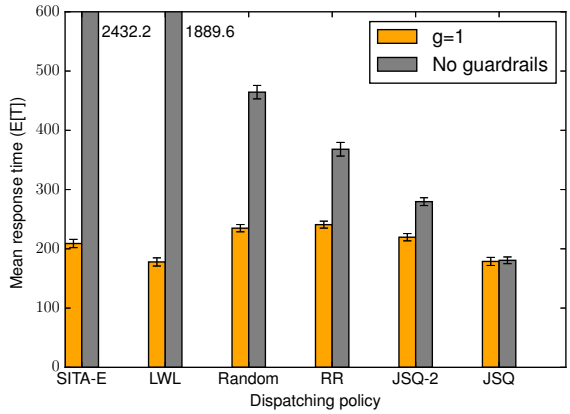
Figure 3.8: Simulation with many servers: $k = 100$. Size distribution shown is Bounded Pareto with $\alpha = 1.5$ and range $[1, 10^6]$. $C^2 \sim 333$. 10 trials simulated, 95% confidence intervals shown.

FPI heuristic from the simulations of other job size distributions in Figures 3.11 and 3.12 due to lack of time.

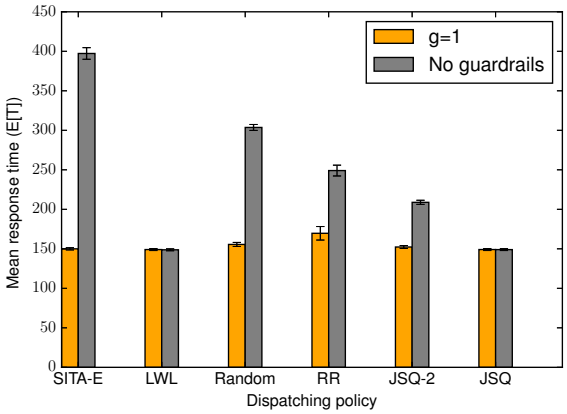
Figures 3.8 and 3.9 show simulations with *many more servers*. Specifically, they use $k = 100$ servers, as opposed to $k = 10$ in other simulations. The only setup where guardrails degrade mean response time of the underlying policy is JSQ with Bounded Pareto job size distribution in heavy traffic, shown in Figure 3.8 (a), but G-LWL and G-FPI have performance on par with JSQ in that case.

Figure 3.10 shows simulations with *light traffic*, specifically $\rho = 0.5$. The trends are largely the same as those in Section 3.6, but the differences in mean response time are smaller. This is to be expected because a large fraction of jobs experience no delay. In fact, the mean response time is nearly equal to the mean service time for many of the dispatching policies ($\mathbb{E}[T] \approx 30$ in (a), $\mathbb{E}[T] \approx 15$ in (b)). Guardrails are particularly effective in the Bimodal case shown in (b), dispatching nearly every small job to a server with no other small jobs. In addition to the loads shown, we have also simulated a range of loads from $\rho = 0.2$ to $\rho = 0.9975$. The trends are consistent across all loads, but the differences are less pronounced at lower loads.

Figures 3.11 and 3.12 show simulations with *different job size distributions*. We specifically simulate with Hyperexponential and Exponential job size distributions, representing another high-variance distribution and a low-variance distribution, respectively. In the Hyperexponential case shown in Figure 3.11, LWL performs particularly poorly without guardrails, similar to the Bimodal case. Roughly speaking, this is because there again are two types of jobs, though each has an exponential distribution instead of a deterministic one, and a job of the large type can cause many jobs of the small type to be dispatched to a single server. But again, guardrails effectively mitigate this problem, although JSQ slightly outperforms G-LWL in very heavy traffic. In the Exponential case, LWL without guardrails performs well already, but adding guardrails does not degrade its performance.

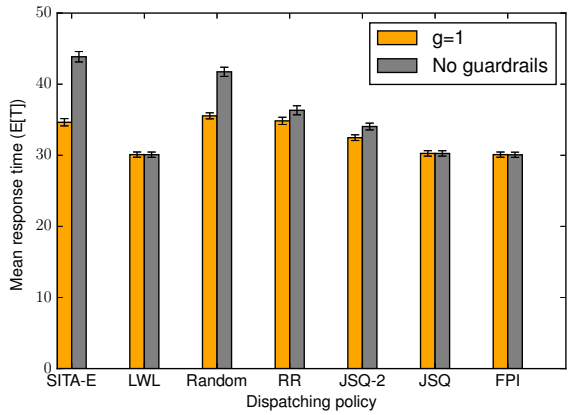


(a) $\rho = 0.98$

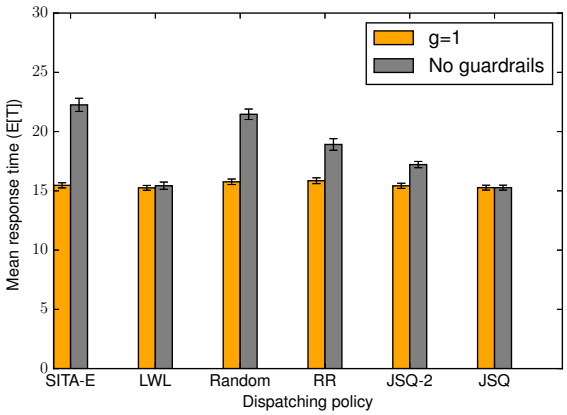


(b) $\rho = 0.80$

Figure 3.9: Simulation with many servers: $k = 100$. Size distribution shown is Bimodal with 99.95% size 1 jobs and 0.05% size 1000 jobs. $C^2 \sim 221$. 10 trials simulated, 95% confidence intervals shown.

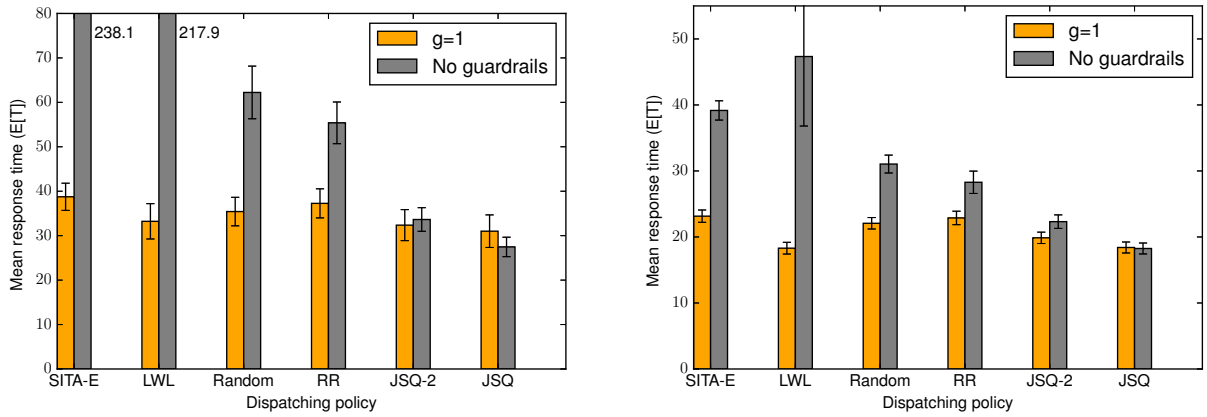


(a) Bounded Pareto job size distribution: $\alpha = 1.5$, range $[1, 10^6]$, $C^2 \sim 333$.



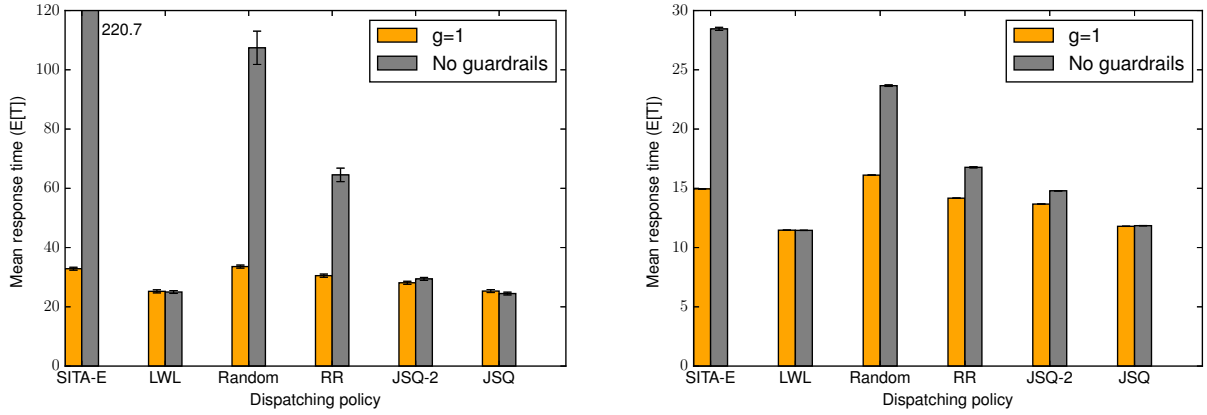
(b) Bimodal job size distribution: size 1 w.p. 99.95%, size 1000 w.p. 0.05%, $C^2 \sim 221$.

Figure 3.10: Simulation with light traffic: $\rho = 0.5$. Simulation uses $k = 10$ servers. Two job size distributions shown. 10 trials simulated, 95% confidence intervals shown.



(a) $\rho = 0.98$ (b) $\rho = 0.80$

Figure 3.11: Simulation with different size distribution: Hyperexponential with mean ~ 1.5 and $C^2 \sim 444$. Simulation uses $k = 10$ servers. 10 trials simulated, 95% confidence intervals shown.



(a) $\rho = 0.98$ (b) $\rho = 0.80$

Figure 3.12: Simulation with different size distribution: Exponential with mean 1. Simulation uses $k = 10$ servers. 10 trials simulated, 95% confidence intervals shown.

3.8 Practical Considerations

We now discuss several useful properties of guardrails that help when implementing them in practical systems. We also extend guardrails to cover a broader range of applications.

3.8.1 Robustness to Network Delays

Guardrails are relatively simple to implement: the dispatcher stores work counters for each rank and each server, increasing the appropriate work counter whenever it dispatches a job (see Algorithm 1). For the most part, the dispatcher does not need to monitor the precise state of each server. The only exception is that whenever a server becomes empty, the server *resets*, which decreases all of the dispatcher's work counters for that server. As we will explain shortly, this complicates the implementation, particularly in settings with network delays.

Fortunately, resets are optional for the purposes of heavy-traffic optimality (see Remark 3.5.1). However, resets are still desirable because they help decrease response time at lower loads. Specifically, resets ensure that a guarded policy is always allowed to dispatch jobs to empty servers. We thus do not want to ignore resets entirely.

To implement resets without the dispatcher needing to track the remaining work at each server at all times, servers can send "reset messages" to the dispatcher when they become empty. This works well so long as messages do not experience network delays, because our analysis (see Lemmas 3.5.1 and 3.5.3) assumes that servers only reset when they are empty, which might not be the case if a reset message is delayed.

In practice, reset messages may well experience network delays, To handle delays, the dispatcher should ignore reset messages from servers that might not be empty. One protocol for doing so is:

- The dispatcher stores, for each server s, a hash of all the job IDs sent to s.
- Each server s stores a hash of all the job IDs it has received.
- When a server becomes empty, it sends a reset message to the dispatcher which contains the currently stored hash.
- When the dispatcher receives a reset message from server s, it resets s if the reset message's hash matches the stored hash for s. Otherwise, the dispatcher ignores the reset message.

3.8.2 Multiple Dispatchers

Many large load balancing systems in practice have multiple dispatchers, each of which sends jobs to the same set of servers. Communication between the dispatchers may be limited, in which case they each have to make dispatching decisions independently. Fortunately, in systems with multiple dispatchers, it suffices to have each dispatcher independently implement guardrails. As explained below, we obtain the same theoretical guarantees for each of the following:

- A system with d dispatchers, each independently satisfying guardrails with tightness g.
- A system with a single dispatcher satisfying guardrails with tightness dg.

Guardrails thus guarantee heavy-traffic optimality for systems with any constant number of dispatchers.

To see why it suffices to implement guardrails separately for each dispatcher, consider a system with d dispatchers. Suppose each dispatcher separately keeps "local" guardrail work counters, which only track arrivals to that dispatcher, and implements guardrails with tightness g. We can also imagine what the "global" guardrail work counters, which track all arrivals at all dispatchers, would look like, even though there is no physical device storing them. We ask: given that the local counters have tightness g, what is the tightness of the global counters? Consider the local and global rank r guardrail work counters for two servers s and s'. Each dispatcher's local counter pair has difference at most gc^{r+1} (see Definition 3.3.1), and there are d dispatchers, so the global counter pair has difference at most dgc^{r+1}. This means the global counters stay within tightness dg.

Systems with multiple dispatchers tend to be large systems in which network delays are non-negligible. The reset protocol from Section 3.8.1 can be easily adapted to multiple dispatchers by having each server store a separate hash of job IDs for each dispatcher.

3.8.3 Scheduling Policies other than SRPT

We have shown that guarded dispatching policies provide theoretical guarantees and good empirical performance for load balancing systems using SRPT scheduling at the servers. However, in some settings it is impossible to use SRPT. For example, network hardware often allows scheduling using only finitely many priority classes, in which case SRPT can only be approximated [110, 160]. Systems may also choose a non-SRPT scheduling policy for other reasons, such as fairness concerns [231].

Guardrails are sometimes suitable even when the servers are using a scheduling policy other than SRPT. In particular, we can extend our theoretical guarantees to many preemptive size-based scheduling policies that favor small jobs. We have already proven such a guarantee for the Priority-c policy (see Theorem 3.4.2). Using bounds proved by Wierman et al. [233], one can extend our results to all policies in their SMART class, which includes Preemptive-Shortest-Job-First (PSJF) and Shortest-Processing-Time-Product (SPTP, also known as RS).

Guardrails can also provide guarantees for size-based policies with finitely many priority classes, which are used in some computer systems to approximate SRPT [110, 160]. In this setting, each priority class corresponds to an interval of job sizes. Here it is most natural to use a slightly modified version of guardrails: a guarded policy is one ensuring that for any class i, the maximum difference between two servers' class i work counters never exceeds the upper bound of class i's size interval. If the job size distribution is bounded, these modified guardrails guarantee a mean response time bound analogous to Theorem 3.4.1. This implies that in the heavy traffic limit, the system's performance approaches that of one large server using the same scheduling policy.

So far, we have only considered policies that use job size information to favor small jobs. This is the setting in which guardrails are most likely to be effective. We conjecture that guardrails might also be useful for servers using PS or Foreground-Background (FB) scheduling, as these policies also tend to favor small jobs, so they may benefit from spreading out small jobs across the servers.

3.8.4 Heterogeneous Server Speeds

We have thus far assumed that all servers in the system have the same speed, but this is not always the case. Fortunately, guardrails can be adapted to systems with heterogeneous server speeds. The key is to track each server's guardrail work counter $G_s(t)$ in units of time. That is, when we dispatch job of size x to a server with speed μ, we increase the server's guardrail work counter by x/μ. It is simple to generalize our response time bound in Theorem 3.4.1 to this setting by multiplying the bound's last term by $\mu_{\max}/\mu_{\min}$, where $\mu_{\min}$ and $\mu_{\max}$ are the minimal and maximal server speeds, respectively. This implies that any guarded policy paired with SRPT service is heavy-traffic optimal with heterogeneous servers.

3.9 Technical Conclusion

We introduce *load balancing guardrails*, a technique for augmenting dispatching policies that ensures low response times in load balancing systems using SRPT scheduling at the servers. We prove that guardrails guarantee optimal mean response time in heavy traffic, and we show empirically that guardrails reduce mean response time across a range of loads. Moreover, guardrails are simple to implement and are a practical choice for large load balancing systems, including those with multiple dispatchers and network delays.

One direction for future work could address a limitation of guardrails: they require the dispatcher to know each job's exact size. Many computer systems only have access to noisy job size estimates or have no size information at all. When exact size information is not available, minimizing mean response time becomes much more complex, as it is not even clear what scheduling policy should be used at the servers. It is possible that a variation of guardrails could be used to create good dispatching policies when using the celebrated Gittins index scheduling policy [1, 72] at the servers.

Our analysis of guardrails constitutes the first closed-form mean response time bound for load balancing systems with general job size distribution and complex dispatching and scheduling policies. However, the bound is only tight in the heavy-traffic limit. Developing better analysis tools for the light traffic case remains an important open problem.

3.10 General Conclusion

3.10.1 Summary

We start by summarizing the results proven in this chapter, as well as the key techniques behind these results.

New class of dispatching policies: Guardrails We introduce a new class of dispatching policies called *guardrails* policies (See Definition 3.3.1). The guardrails policies are designed for use with SRPT scheduling at each server. Whenever a job arrives, a guardrails policy specifies a restricted set of servers that the job must be dispatched to, then defers to a baseline dispatching policy to select among the restricted set of servers. This restriction ensures that each server will

have a similar amount of work of each possible size of job: Similar amounts of work of small jobs, of medium jobs, and so forth.

Results: Guardrails/SRPT is optimal We prove that any guardrails dispatching policy, when combined with SRPT scheduling at each server, achieves the best possible mean response time when the load on the system is high and the queue lengths become long (See Corollary 3.4.1).

We also prove an upper bound on the mean response time of any guardrails/SRPT combination, in the form of a clean mathematical formula (See Theorem 3.4.1). This upper bound proves that guardrails/SRPT achieves similar mean response time to that of resource-pooled SRPT, in which all k servers are combined into a single ultra-fast server using SRPT scheduling (See Theorem 3.4.2). This bound becomes tight as load becomes high.

Simulation results: Guardrails/SRPT is near-optimal under moderate traffic While our results focus on heavy traffic, we show via simulation in Figs. 3.6 and 3.7 that guardrails/SRPT policies are also very good for moderate load. In particular, we see that under a variety of high-variance job size distributions and a variety of loads, guardrails/SRPT policies achieve either the lowest mean response time or near the lowest of the policies simulated. Further simulations in Section 3.7 also confirm this result.

Key technique: Relevant work analysis To understand the response time of a specific job, we need to understand the work at each server that is *relevant* to the specific job: that is, the work that the job must wait behind before it can reach service: The work that is relevant to a specific job of size x consists of other jobs with remaining size smaller than x.

The guardrails class of dispatching policies ensure that the amount of relevant work at each server is close to equal to the amount at each other server. We then use a coupling analysis involving a resource-pooled SRPT system. We prove that this amount of relevant work at each server is always similar to $1/k$ of the amount of work at the resource-pooled system. Because these bounds hold at each server, a given job will achieve good mean response time, regardless of which server the job is dispatched to. The dispatching policy can therefore instead focus on maintaining the similarity of the servers. We use this bound on relevant work, combined with additional properties of the guardrails policies, to show that the guardrails/SRPT combination achieves nearly the same mean response time as resource-pooled SRPT (See Theorem 3.4.1).

3.10.2 Generalizing our setting

In our setting, we assumed SRPT service and that the dispatcher knows the sizes of the jobs. However, one could imagine three alternative scenarios:

A. The sizes of the jobs are unknown to both the scheduling and dispatching policy.

B. Estimates of the sizes are known, but not the exact sizes.

C. Size information is known for the purpose of dispatching, but due to legacy scheduling systems jobs must be served in FCFS order.

We consider these settings in the remainder of Section 3.10.2.

Generalizing to unknown job sizes

In the single-server setting, the optimal way to handle unknown job sizes is the Gittins index scheduling policy [72, 73, 199]. However, essentially nothing is known about dispatching in combination with Gittins scheduling.

A natural choice of dispatching policy would be Join-the-Shortest-Queue (JSQ), as little is known about the jobs at each server except for their number. However, the Gittins scheduling policy often performs a significant amount of preemption. As a result, there may be many incomplete jobs at each server, giving additional information beyond the number of jobs at the server. A partially complete job might be shorter or longer in expectation than a fresh job, depending on the job size distribution.

We therefore ask:

> What is the optimal dispatching policy to combine with Gittins scheduling when dispatching jobs with unknown sizes?

We discuss the problem of optimal dispatching in combination with Gittins scheduling in Section 8.3.3.

Generalizing to estimated job sizes

The Gittins index policy is also applicable in the setting where estimated job sizes are available. As estimates become more and more accurate, Gittins simplifies to the SRPT scheduling policy. How should we use size estimates to dispatch to servers using the Gittins scheduling policy? Intuitively, as estimates become more accurate, our dispatching policy should interpolate between a low-information policy like JSQ, and a high-information policy such as guardrails. We discuss this problem further in Section 8.3.3.

Size-based dispatching to servers using FCFS scheduling

In some settings, even though size information is available for dispatching, FCFS scheduling may be used at the servers, for instance due to the use of legacy systems. While size-based dispatching to servers using FCFS scheduling has been extensively studied [64, 108], *optimal* size-based dispatching remains open.

In preliminary work in the $k = 2$ server setting, Xie and Scully [237] claim that their novel Careful-Routing-to-Achieve-Bound (CRAB) dispatching policy achieves optimal mean response time for certain job size distributions, in the limit as arrival rate approaches capacity. CRAB is a load-imbalancing policy, in contrast to guardrails policies, which balance load. This optimality result for CRAB would be the first optimality result in a setting with size-based dispatching and non-size-based scheduling. We look forward to their full paper.

3.10.3 Potential Impact

We now explore potential directions in which our guardrails dispatching policies could be applied, and discuss ways of adapting guardrails to real-world environments. This section is intended to complement Section 3.8, where we discuss several real-world considerations, including multiple non-communicating dispatchers and network delays.

Adopting Guardrails into Dispatching-based Computing Systems

Our analysis of the combination of guardrails dispatching and SRPT scheduling shows that guardrails/SRPT can dramatically lower response times compared to other scheduling policies, and that the guardrails/SRPT combination is in fact optimal for mean response time under sufficiently heavy load. Our hope is that our results will lead computing system operators to adopt guardrails/SRPT dispatching and scheduling in their systems.

However, the road from theoretical results to adoption requires overcoming several hurdles. These include:

Dealing with size estimates. Often only approximate size information is known, rather than the exact size information utilized by guardrails. Guardrails policies use size information in two ways: To assign each job to a size class, and to track how much work has been sent to each server. For assigning jobs to a size class, one can simply estimate which size class a job should be in. In this approach, one should use a fairly large size class width c, to reflect the uncertainty in the estimate. For tracking how much work has been sent to each server, one can start with the estimated work sent, and update that estimate once the job runs and its true size is clear.

Handling jobs with differing importance. In the real world, some jobs may be far more latency-sensitive than others. This is naturally modeled by introducing a holding cost per second waited for each job. In this setting, research has previously considered the case where a job's holding cost is known to the dispatching policy, but its size is unknown [50].

Let us instead consider the setting where both the job's size and its holding cost are known. The natural generalization of SRPT to this setting is to prioritize jobs according to the ratio of holding cost to remaining size. To generalize guardrails dispatching policies to this setting, one should simply assign jobs to an importance class, based on this ratio, rather than a size class as in standard guardrails policies. The results of this chapter should generalize to prove optimality in that setting as well, if the ratio of largest to smallest holding cost is bounded.

Handling data locality. In many practical systems, data is housed at specific servers, and cannot be migrated easily. Jobs require certain data to be served, and can only be served at servers that have that data. These affinities can be modeled via a compatibility graph connecting job types to servers. This model has received extensive study in size-unknown dispatching settings [163, 189, 190, 243]. That line of research has demonstrated the importance of spreading data across servers to equalize the total loads at the servers. To incorporate size-based scheduling and dispatching, this chapter shows us the importance of additionally balancing the load of small jobs, medium jobs, and large jobs across the servers.

Guardrails for Dispatching People to Multiple Servers

When dealing with people, size is not the only important quality of a job. Jobs can be differentiated both by their sizes and by the time-sensitivity or *value* of the service. For instance, when ambulances are bringing patients to hospitals, a patient undergoing a heart attack might have a much higher value associated with being treated soon, as compared to a patient with a broken

arm. An appropriate performance metric for this scenario is the mean product of value and response time. Research has previously considered the dispatching setting in which a job's value is known to the dispatcher, but its size is not [50]. When dispatching people, it makes sense to consider the setting where both size and value are known, or at least can be estimated.

In the single-server setting, if both value and size are known, the policy which minimizes the product of value and response time is the "$c\mu$-rule", which serves the job with maximal ratio of value to size. We would therefore use $c\mu$ scheduling at the servers in our dispatching setting.

Generalizing guardrails In the $c\mu$-scheduling setting, we can generalize the guardrails policies to handle both size and value information. For each job, calculate its ratio of value to size. Split jobs into classes accordingly: High ratio jobs, medium ratio jobs, and so forth. We would then dispatch jobs to ensure that the amount of high ratio work at each server is similar, medium ratio work, etc.

However, it is not obvious that this dispatching policy would be optimal. In our bounds on mean response time for the basic guardrails policy, we make key use of the fact that size and importance are aligned. See Lemmas 3.5.1 and 3.5.2 in particular. In practice, with different values for different jobs, size and importance may not be aligned, rendering optimality more difficult to prove.

Additional consideration: Information delay Another wrinkle in dispatching people is that there may be a significant delay between when we tell people which server to go to, and when people actually arrive at that server. As a result, a queue which we previously recommended may end up inundated with people, all of whom are responding to out-of-date instructions. If the dispatching policy is not prepared for people to respond at a delay, this dynamic may result in terrible performance [176]. To alleviate this behavior, we can look at a long history of workload information when deciding which queues to recommend.

Part III

Multiserver Jobs

Chapter 4

First Multiserver-job Scheduling Response Time Analysis: ServerFilling

This chapter is based on my paper "WCFS: A new framework for analyzing multiserver systems", published in Queueing Systems in 2022, written with my coauthors Mor Harchol-Balter and Alan Scheller-Wolf [84, 86].

4.1 General Introduction

Modeling large-scale computing systems Modern computing systems scale up in two important ways. First, they contain huge numbers of servers, allowing them to process many jobs at once. Second, the *jobs* in modern computing systems require vastly different amounts of resources, ranging from a single CPU core to thousands of machines. The scheduling decision in these systems consists of selecting a combination of jobs to serve at once. Improving the scheduling policy can improve mean response time by orders of magnitude with no additional resources.

To capture the behavior of these computing systems, we use a *multiserver-job* (MSJ) queueing model, depicted in Fig. 4.1. Jobs arrive randomly over time, and wait in a central queue. Each job has a *server need*, which specifies the number of servers the job requires in order to enter service. The job requires a fixed number of servers throughout its time in service. All servers are identical, so the scheduling policy can decide to serve any set of jobs with total server need at most k, where k is the total number of servers in the system. An important scheduling policy, which we depict in Fig. 4.1, is the First-Come First-Served (FCFS) policy. FCFS places the oldest jobs in the system into service one-by-one, until a job is reached whose server need exceeds the number of currently available servers. At this point, the blocked job, and all jobs behind it in the queue, must wait until some of the jobs in service complete and more servers become available. Some other scheduling policies allow *preemption*, which refers to pausing a job in service, placing it back in the queue, and returning to that job later.

MSJ performance We are interested in the performance effects of different scheduling policies. Two natural measures of a policy's performance are its *stability region*, and its *mean response time*. A policy's stability region is the range of arrival rates of jobs for which the policy

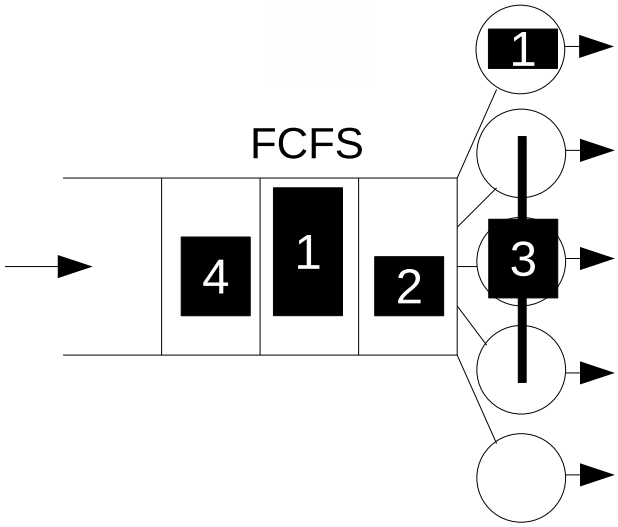


Figure 4.1: The FCFS scheduling policy in the multiserver-job setting. Each job has two characteristics: A *server need*, the number of servers it requires in order to run, and a *duration*, the amount of service time it requires. FCFS serves jobs in arrival order. If the next job in the queue has larger server need than the number of available servers, that job, and all jobs behind it, must wait in the queue.

can keep up with the arriving jobs, so that the queue length doesn't diverge to infinity. If a scheduling policy leaves servers idle while jobs wait in the queue, this diminishes its stability region. A policy's mean response time is the mean time from when a job arrives to when it completes.

Theoretical understanding of stability region Much of the theoretical study of scheduling in the MSJ model has focused on the stability region. Under the FCFS policy, the stability region has been theoretically characterized in a variety of settings [83, 88, 187]. It is known that FCFS never achieves the *optimal* stability region across all scheduling policies, except in trivial settings. By contrast, the MaxWeight [148] and Randomized Timers [71] policies are known to achieve *optimal* stability regions. By optimal stability region, we mean that these policies are stable for the largest possible set of arrival rates, among all scheduling policies.

MSJ mean response time not understood Unfortunately, almost nothing is known about mean response time in the multiserver-job setting. Mean response time has not been theoretically characterized in closed form for any scheduling policy in the MSJ setting, except for FCFS service in the extremely restrictive setting of $k = 2$ servers [31, 63].

Existing scheduling policies seem intractable to analysis, so we ask: Can we devise a novel MSJ scheduling policy, for which we can analyze mean response time? Either an exact characterization of mean response time or a proof of upper and lower bounds would be a major breakthrough.

Key idea: Work conserving & finite skip Our key technique is to invent a class of scheduling policies, which we call the *work-conserving finite-skip* (WCFS) policies, for which we can analyze mean response time in a clean, unified fashion. "Work conservation" refers to keeping all servers occupied, whenever a sufficient number of jobs are present. "Finite skip" refers to serv-

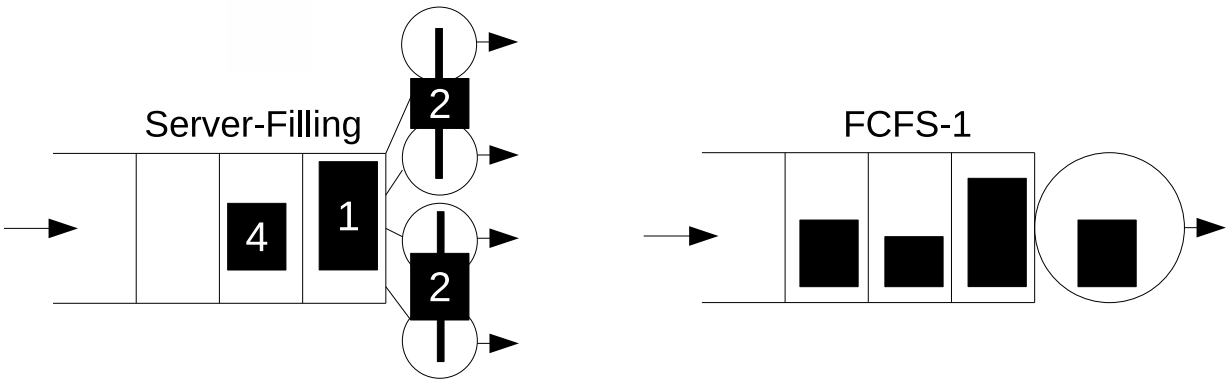


Figure 4.2: The ServerFilling scheduling policy in the multiserver-job setting, alongside the resource-pooled FCFS system. In the resource-pooled system, a job completes in a proportionately shorter duration. For instance, if a 2-server job completes in 10 seconds in a $k = 4$ server system, it would complete in 5 seconds in the resource-pooled system. In the resource-pooled system, it receives twice the service rate, and completes in half of the time. We show that these two systems have nearly identical mean response time.

ing jobs in near-FCFS order, only looking at finite number of the oldest jobs in the system when deciding which jobs to serve. We invent two MSJ policies which fall in the WCFS class, ServerFilling and DivisorFilling. Our unified analysis for the entire WCFS class of policies allows us to characterize their mean response times. We also find WCFS policies in a variety of other multiserver and single-server scheduling settings (see Section 4.2), allowing us to characterize those policies' mean response times as well.

Key idea: Resource pooling We show that all WCFS policies, including the MSJ scheduling policies ServerFilling and DivisorFilling, achieve mean response time nearly identical to that of *resource-pooled* FCFS. In the resource-pooled system, all k servers are combined into one ultra-fast server with k times the speed. Jobs speed up proportionately. For instance, a job that took one hour on $k/2$ servers in the original system would only take half an hour in the resource-pooled system. We show both systems in Fig. 4.2. Because WCFS policies such as ServerFilling serve jobs in near-FCFS order, resource-pooled FCFS is a natural target to compare against.

4.2 Technical Introduction

Consider the following four queueing models, which are each important, practical models, but which seem very different. We will refer to these models throughout this chapter as our *four motivating models*:

- **Heterogeneous M/G/k:** A k-server system where servers runs at different speeds. Jobs are held at a central queue and served in First-Come-First-Served (FCFS) order when servers become available. If multiple servers are vacant, a server assignment policy such as Fastest Server First is applied.
- **Limited processor sharing:** A single-server system where if at least k jobs are present, the k earliest arrivals each receive an equal fraction of service. If fewer than k jobs are

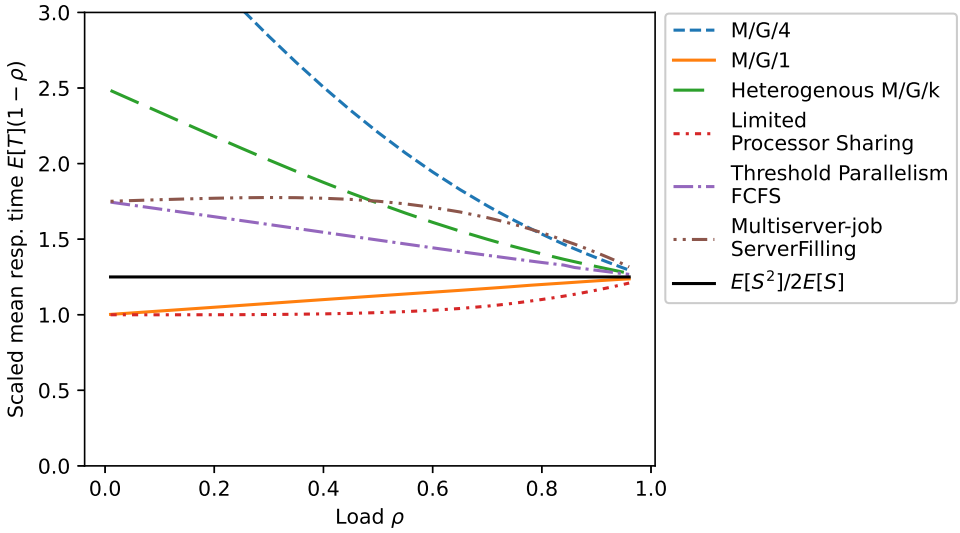


Figure 4.3: Scaled mean response time of our four motivating models, as well as the related M/G/k and M/G/1 models. Our four motivating models will be further defined in Section 4.4. In each case, the job size distribution S is distributed as $Hyperexp(\mu_1 = 2, \mu_2 = \frac{2}{3}, p_1 = \frac{1}{2})$. The black line is $\mathbb{E}[T](1-\rho) = \frac{\mathbb{E}[S^2]}{2\mathbb{E}[S]}$, the heavy traffic behavior of M/G/1/FCFS and each of our models of interest. 10^9 arrivals simulated. $\rho \in [0, 0.96]$ to ensure accurate results.

present, the server is split equally among jobs.

- **Threshold parallelism:** A multiserver system where jobs are moldable, meaning they can run on any number of servers, up to some threshold, with perfect speedup. We consider FCFS service, where jobs are allocated a number of servers equal to their threshold, as long as servers are available. The next job in FCFS order is then allocated the remaining servers, which may be less than the job's threshold.
- **Multiserver-jobs under the ServerFilling policy:** A multiserver system where the jobs are called "multiserver jobs," because each job requires a fixed number of servers, which it holds concurrently throughout its service. We examine a service policy called *ServerFilling*, which always fills all of the servers if enough jobs are available.

We define these models in more detail in Section 4.4.

We will show that, while our four motivating models appear quite different, their mean response times, $\mathbb{E}[T]$, are very similar, especially in the heavy-traffic limit. Specifically, we will show that their behavior in the heavy traffic limit is identical to that of the M/G/1/FCFS model, and in fact the mean response time of each of these disparate models only differs by an *additive* constant from that of M/G/1/FCFS for all loads, a much stronger result than convergence in heavy traffic.

The similarity of these models is illustrated by Fig. 4.3, which shows mean response time, $\mathbb{E}[T]$, scaled by a factor of $1-\rho$, to help illustrate the asymptotic behavior in the $\rho \to 1$ limit. Observe that in each of our models of interest, as well as in the M/G/1 and the M/G/4, $\mathbb{E}[T](1-\rho)$ converges to $\mathbb{E}[S^2]/2\mathbb{E}[S]$, the mean of the equilibrium (excess) distribution, where S denotes

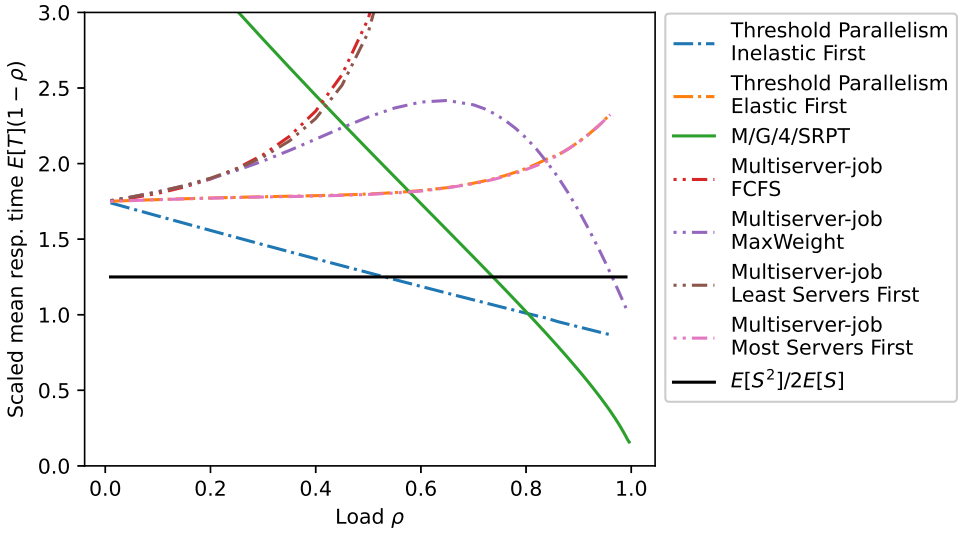


Figure 4.4: Scaled mean response time of alternative models and policies. All of these models and policies will be explained in Section 4.8. $S \sim Hyperexp(\mu_1 = 2, \mu_2 = \frac{2}{3}, p_1 = \frac{1}{2})$. Black line is $\mathbb{E}[T](1-\rho) = \frac{\mathbb{E}[S^2]}{2\mathbb{E}[S]}$. 10^9 arrivals simulated, $\rho \in [0, 0.96]$ to ensure accurate results, except MaxWeight: 10^{10} arrivals, $\rho \in [0, 0.99]$.

the job size distribution and $\rho = \lambda\mathbb{E}[S] < 1$ is the system load.

This similarity is striking – to see just how notable it is, consider a variety of alternative models and policies shown in Fig. 4.4. For these alternative models, scaled mean response time either does not converge at all, or converges to a different limit entirely.

This contrast poses an intriguing question:

Why do our four motivating models converge to M/G/1/FCFS in heavy traffic?

To put it another way, we ask what crucial property our four motivating models share, that is not shared by the alternative models in Fig. 4.4.

To answer this question, we define the "work-conserving finite-skip" framework (WCFS), which applies to a broad class of models. The WCFS class contains our four motivating queueing models, as well many others. We demonstrate that for any model in the WCFS class (which we call a "WCFS model"), if the job size distribution S has bounded expected remaining size, then its scaled mean response time converges to the same heavy traffic limit as the M/G/1/FCFS. Specifically, we prove that

Theorem 4.6.1. For any model $\pi \in$ WCFS with bounded expected remaining size[1],

$$\lim_{\rho \to 1} \mathbb{E}[T^\pi](1-\rho) = \frac{\mathbb{E}[S^2]}{2\mathbb{E}[S]}.$$

Theorem 4.6.1 follows from an even stronger result: We prove that the difference in mean response time between any WCFS model and M/G/1/FCFS is bounded by an explicit additive constant, that may depend on the specific WCFS model.

[1]This assumption is defined in Section 4.3.3.

Theorem 4.6.2. For any model $\pi \in$ WCFS with bounded expected remaining size,

$$\mathbb{E}[T^\pi] \leq \frac{\rho}{1-\rho}\frac{\mathbb{E}[S^2]}{2\mathbb{E}[S]} + c^\pi_{upper}$$
$$\mathbb{E}[T^\pi] \geq \frac{\rho}{1-\rho}\frac{\mathbb{E}[S^2]}{2\mathbb{E}[S]} + c^\pi_{lower}$$

for explicit constants c^π_{upper} and c^π_{lower} not dependent on load ρ.

Theorem 4.6.2 not only implies Theorem 4.6.1, it also guarantees rapid convergence of scaled mean response time to the heavy traffic limit specified in Theorem 4.6.1.

In summary, this chapter makes the following contributions:

- We define the WCFS framework and our bounded expected remaining size assumption. (Section 4.3)
- We prove that each of the four motivating models is a WCFS model. (Section 4.4)
- We discuss prior work on WCFS models. (Section 4.5)
- We prove that all WCFS models with bounded expected remaining size have the same scaled mean response time as M/G/1/FCFS, and mean response time within an additive constant of M/G/1/FCFS. (Section 4.6)
- We empirically validate our results, contrasting heavy traffic behavior of WCFS models and non-WCFS models. (Section 4.8)

4.3 The WCFS Framework and WCFS Models

In Sections 4.3.1 and 4.3.2, we define the WCFS framework and resulting class of models. In Section 4.3.3, we define our "bounded expected remaining size" assumption. In Section 4.3.4, we define a few more concepts that will be used in this chapter.

Job sizes are sampled $i.i.d.$ from a job size distribution. Once sampled, job sizes are fixed: we assume we assume preempt-resume service if a job is preempted while in service. Intuitively, the size of a job represents the amount of work associated with the job. Size will be defined in more detail in Section 4.3.1.

4.3.1 WCFS Framework and WCFS Models

The WCFS framework applies to the class of models with Poisson arrivals at rate λ, and with the following properties:

1. Finite skip (Section 4.3.1),
2. Work conserving (Section 4.3.1)
3. Non-idling (Section 4.3.1).

Finite skip

We first define the finite-skip property, which defines the class of finite-skip models. Consider the jobs in the system in arrival order. Associated with each finite-skip model, there is a finite

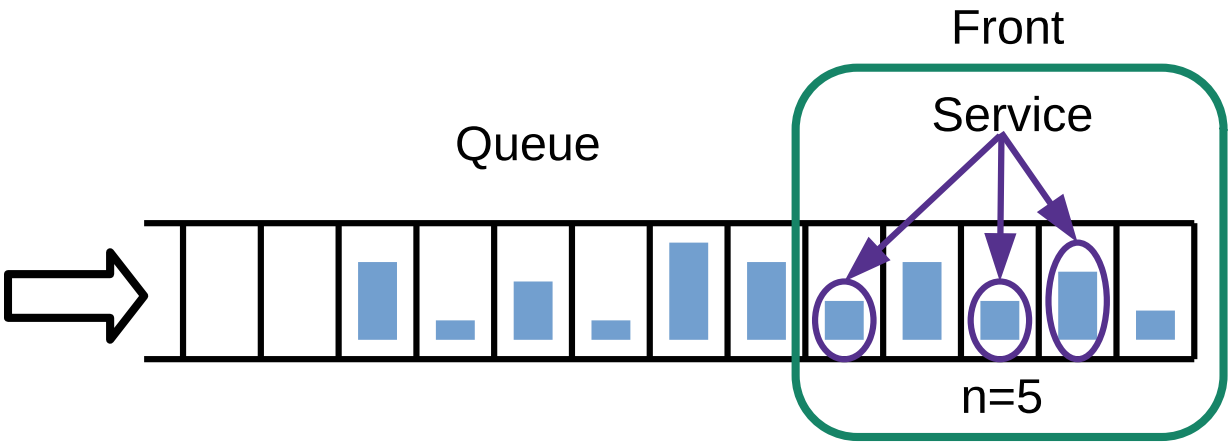


Figure 4.5: Diagram of a Finite-Skip Model

parameter n. We partition the jobs in the system into two sets: the (up to) n jobs which arrived longest ago, which we call the *front*, and all other jobs, which we call the *queue*. The *finite-skip property* specifies that, among all of the jobs in the system, the server(s) only serve jobs in the front. In particular, no jobs beyond the first n jobs in arrival order receive any service. Fig. 4.5 shows a generic finite-skip model.

Definition 4.3.1. *We call the front* full *if at least n jobs are present in the system, and therefore exactly n jobs are at the front.*

The intuition behind the term "finite skip" comes from imagining moving through the jobs in the system in arrival order, skipping over some jobs and serving others. In a finite-skip model only the first n jobs can be served, so only finitely many jobs can be skipped.

Work conserving

Now, we will specify what we mean by "work conserving," which is a different concept here than in previous work.

First, we normalize the total system capacity to 1, regardless of the number of servers in the system. For instance, in a homogeneous k-server system, we would think of each server as serving jobs at rate $1/k$.

Whenever a job is in service, it receives some fraction of the system's total service capacity, which we call the job's *service rate*. Let $B(t) \leq 1$ denote the total service rate of all jobs in service at time t, and let B be the stationary total service rate, assuming for now such a quantity exists.

We define a job's *age* at time t to be the total amount of service the job has received up to time t: a job's age increases at a rate equal to the job's service rate whenever the job is in service. Each job has a property called its *size*. When the job's age reaches its size, the job completes.

In particular, we assume that every job j has a size s_j and a class c_j drawn i.i.d. from some general joint distribution. Let (S, C) be the random variables denoting a job's size and class pair. A job's class is static information known to the scheduler, while a job's size is unknown to the scheduler. For instance, in the threshold parallelism model defined in Section 4.4.3, a job's

parallelism threshold is its class.

Definition 4.3.2. *We call the system* maximally busy *if the total capacity of the system is in use, namely if the total service rate of jobs in service is 1.*

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

In other words, a finite-skip model is work-conserving if, whenever there are at least n jobs in the system, the total service rate is 1.

Now that we have defined a job's size, we can also define the load of the system: $\rho = \lambda \mathbb{E}[S]$. Load ρ is the time-average service rate, or equivalently the time-average fraction of capacity in use. Specifically, $\rho = \mathbb{E}[B]$. We assume $\rho < 1$ to ensure stability.

Non-idling

We also assume that the total service rate $B(t)$ is bounded away from zero whenever a job is present. Specifically, whenever a job is present, we assume that $B(t) \geq b_{\text{inf}}$, for some constant $b_{\text{inf}} > 0$.

This assumption is key to bounding mean response time under low load. For an example, see the batch-processing system in Section 4.3.2.

4.3.2 Examples and non-examples

To clarify which models fit within the WCFS framework, we give several examples, both positive and negative.

- **M/G/k/FCFS:** This is a WCFS model with $n = k$.
- **M/G/∞:** This model is not finite skip. All jobs are in service, regardless of the number of jobs in the system: there is no finite bound on the number of jobs in service.
- **M/G/k/SRPT:** In this model, the k jobs with smallest remaining size are served at rate $1/k$. This model is not finite skip because the jobs with smallest remaining size can be arbitrarily far back in the arrival ordering.
- **Multiserver-job model:** First, consider a multiserver system with $k = 2$ servers, and where each job requires either 1 or 2 servers.

  If jobs are served in FCFS order, with head-of-the-line blocking, this policy is finite-skip, but not work-conserving. If the front consists of a job requiring 1 server followed by a job requiring 2 servers, the system will only utilize one server. In this case, the system is full, because $n = k = 2$ jobs are present in the system, and hence in the front, but the system is not maximally busy.

  In contrast, consider a service policy which serves a 2 server job if either of the jobs in the front are 2 server jobs, or else serves each of the 1 server jobs at the front. This policy is a special case of the ServerFilling policy, depicted in Fig. 4.3 and defined in general in Section 4.4.4. This policy is finite-skip and work-conserving.
- **Batch-processing M/G/k:** If there are at least k jobs present, the oldest k jobs in the system are each served at rate $\frac{1}{k}$. Otherwise, no service occurs. This model is finite-

skip and work-conserving, but is not non-idling. To see why the non-idling property is necessary for our main results, specifically Theorem 4.6.2, one can show that in the $\lambda \to 0$ limit, response times will grow arbitrarily large in the batch-processing M/G/k. To rule out systems where $\mathbb{E}[T]$ diverges in the $\lambda \to 0$ limit, we assume the non-idling property.

4.3.3 Bounded expected remaining size: Finite $\mathrm{rem}_{\sup}$

At a given point in time, let the *state* of a job j consist of its class c_j and its age a_j. Within our WCFS framework, we allow service to be based on the states of the jobs in the front, but not on the number or states of jobs in the queue.

A key assumption we make is that jobs have bounded expected remaining size from an arbitrary state. Let S_c be the job size distribution for jobs of class $c \in C$. We define $\mathrm{rem}_{\sup}(S, C)$ to be the supremum over the expected remaining sizes of jobs, taken over all states:

$$\mathrm{rem}_{\sup}(S, C) := \sup_{c \in C, a \in \mathbb{R}^+} \mathbb{E}[S_c - a \mid S_c > a].$$

When size S is independent of class C, or when a model has no class information, we will simply write $\mathrm{rem}_{\sup}(S)$.

In this chapter, we focus on job size distributions for which $\mathrm{rem}_{\sup}(S, C)$ is finite. To better understand the finite $\mathrm{rem}_{\sup}(S, C)$ assumption, let's walk through a couple of examples. In all of these examples, let's suppose that the class information is independent of the job size distribution S, so we can simply write $\mathrm{rem}_{\sup}(S)$.

Consider a job size distribution S that is hyperexponential:

$$S = \begin{cases} Exp(\mu_1) & \text{w.p. } p_1 \\ Exp(\mu_2) & \text{w.p. } p_2 \\ Exp(\mu_3) & \text{w.p. } p_3 \end{cases}$$

For all ages a, the expected remaining size is bounded:

$$\mathbb{E}[S - a \mid S > a] \leq \frac{1}{\min(\mu_1, \mu_2, \mu_3)} = \mathrm{rem}_{\sup}(S).$$

More generally, an arbitrary phase type job size distribution S' must have finite $\mathrm{rem}_{\sup}$.

On the other hand, Pareto job size distributions do not have finite $\mathrm{rem}_{\sup}$. Let $S'' \sim Pareto(\alpha = 3, x_{\min} = 1)$, which has finite first and second moments.

$$\mathbb{E}[S'' - a \mid S'' > a] = \frac{a}{2}, \forall a \geq 1$$
$$\lim_{a \to \infty} \mathbb{E}[S'' - a \mid S'' > a] = \infty$$
$$\mathrm{rem}_{\sup} = \sup_a \mathbb{E}[S'' - a \mid S'' > a] = \infty$$

In general, finite $\mathrm{rem}_{\sup}$ roughly corresponds to service time having an exponential or subexponential tail, though there are some subtleties. For instance, a Weibull distribution with $\mathbb{P}\{S \geq a\} = a^{-k}$ for some $k < 1$ has infinite $\mathrm{rem}_{\sup}$, while for $k \geq 1$, $\mathrm{rem}_{\sup}$ is finite.

As a final example, suppose the WCFS scheduling policy was a known-size policy, such as a policy which serves the job with least remaining size among the n jobs in the front, at rate 1. Because we require that service is based only on the age and class of a job, we model this situation by saying that a job's class is its original size. In this case, $S = C$, and the distribution S_x is simply the constant x. As a result, $\mathrm{rem}_{\sup}(S, C) = \sup(S)$. Therefore, in a known-size setting, $\mathrm{rem}_{\sup}$ is finite only if S is bounded.

4.3.4 Work, Number, Response Time

Let the *work* in the system be defined as the sum of the remaining sizes of all jobs in the system; a job's remaining size is its size minus its age.

Let $W(t)$ be the total work in the system at time t. Let $W_Q(t)$ and $W_F(t)$ be the work in the queue and the work at the front, respectively, at time t. (We will generally use the subscripts $_Q$ and $_F$ to denote the queue and the front.) Let W, W_Q, and W_F denote the corresponding time-stationary random variables.

Recall from Section 4.3.1 that $B(t)$ is the total service rate at time t. Note that $\frac{d}{dt}W(t) = -B(t)$, except at arrival instants.

Let $N(t)$ be the number of jobs in the system at time t. Note that $N_F(t) = n$ whenever $N(t) \geq n$, because the front is full, and $N_F(t) = N(t)$ otherwise.

Let T be a random variable denoting a job's time-stationary response time: the time from when a job arrives to when it completes.

4.4 Important WCFS Models

Here we define in more detail the four motivating models mentioned in the introduction and depicted in Fig. 4.3, and show that each is a WCFS model.

4.4.1 Heterogeneous M/G/k

The heterogeneous M/G/k/FCFS models multiserver systems where servers have different speeds. This scenario commonly arises in datacenters, which are often composed of servers with a wide variety of different types of hardware [150, 165]. In the mobile device setting, the big.LITTLE architecture employs heterogeneous processors to improve battery life [37].

Let each server i have speed $v_i > 0$, scaled so that $\sum_i v_i = 1$. While a job is being served by server i, the job's age increases at a rate of v_i, completing when its age reaches its size.

If there are multiple servers open when a job arrives, a server is chosen according to an arbitrary server assignment policy. We only assume that jobs are served in FCFS order, and that no job is left waiting while a server is idle. Under these assumptions, all assignment policies fit within the WCFS framework.

As an example, in Fig. 4.3 we show the scaled mean response time of a heterogeneous M/G/4 with server speeds $0.4, 0.3, 0.2, 0.1$, and the Preemptive Fastest Server First assignment policy.

Heterogeneous M/G/k is a WCFS model

To show that the heterogeneous M/G/k is a WCFS model, we must verify the three properties from Section 4.3.1.

Finite skip: Jobs enter service in FCFS order. As a result, the jobs in service are exactly the (up to) k oldest jobs in the system. The model is finite skip with parameter $n = k$.

Work conserving: The system has total capacity $\sum_i v_i = 1$. Whenever at least k jobs are present in the system, all servers are occupied, and the total service rate is 1. In other words, whenever the front is full, the system is maximally busy.

Positive service rate when nonempty: If a job is present, the job will be in service on some server. The system will therefore have minimum service rate $b_{\text{inf}} \geq v_{\min}$, where $v_{\min} = \min_i v_i$.

4.4.2 Limited Processor Sharing

The Processor Sharing policy for the M/G/1 is of great theoretical interest, and has been extensively studied [238]. However, in real systems, running too many jobs at once causes a significant overhead. A natural remedy is to utilize a policy is known as Limited Processor Sharing (LPS) [91, 170, 216, 240].

The LPS policy is parameterized by some Multi-Programming Level k. If at most k jobs are present 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 $1/k$. Limited Processor Sharing is a WCFS model with $n = k$.

As an example, in Fig. 4.3 we show the scaled mean response time of a LPS system with MPL 4.

Heterogeneous M/G/k is a WCFS model

To show that the heterogeneous M/G/k is a WCFS model, we must verify the three properties from Section 4.3.1.

Finite skip: Jobs enter service in FCFS order. As a result, the jobs in service are exactly the (up to) k oldest jobs in the system. The model is finite skip with parameter $n = k$.

Work conserving: The system has total capacity $\sum_i v_i = 1$. Whenever at least k jobs are present in the system, all servers are occupied, and the total service rate is 1. In other words, whenever the front is full, the system is maximally busy.

Positive service rate when nonempty: If a job is present, the job will be in service on some server. The system will therefore have minimum service rate $b_{\text{inf}} \geq v_{\min}$, where $v_{\min} = \min_i v_i$.

4.4.3 Threshold Parallelism

In modern datacenters, it is increasingly common for jobs to be parallelizable across a variety of different numbers of servers, where the level of parallelism is chosen by the scheduler [48, 178].

The Threshold Parallelism setting models this scenario by assuming that for each job the user gives the ideal, maximum number of servers that the job can utilize.

Under Threshold Parallelism, a job j has two characteristics: Its size s_j and its parallelism threshold ℓ_j, where ℓ_j is some number of servers. Job j may be parallelized across up to ℓ_j servers, with linear speedup. The pair (s_j, ℓ_j) is sampled i.i.d. from some joint distribution (S, L). Note that ℓ_j is the class of the job j.

Let k be the total number of servers. Note that $\ell_j \in [1, k]$. If a job j is served on $q \leq \ell_j$ servers, then it receives service rate $\frac{q}{k}$ and would complete after $\frac{ks_j}{q}$ time in service. The number of servers a job receives can change over time, correspondingly changing its service rate.

We focus on the FCFS service policy. Under this policy, jobs are placed into service in arrival order until their total parallelism thresholds sum to at least k, or all jobs are in service. Any job j which is not the newest job in service is served by ℓ_j servers. The newest job in service is served by the remaining servers. Under FCFS service, Threshold Parallelism fits the WCFS framework.

As an example, in Fig. 4.3 we show the scaled mean response time of a Threshold Parallelism model where the joint distribution (S, L) is $(Exp(2), 1)$ with probability $\frac{1}{2}$, and $(Exp(\frac{2}{3}), 4)$ with probability $\frac{1}{2}$, and with FCFS service.

As a comparison, in Fig. 4.4, we show Threshold Parallelism models with the same joint distribution (S, L), but with different service policies: "Elastic First," prioritizing jobs with $L = 1$, and "Inelastic First," prioritizing jobs with $L = 4$. These policies do not fit within the WCFS framework.

Threshold Parallelism with FCFS service is a WCFS model

Finite skip: The jobs in service are the initial set of jobs in arrival order whose parallelism thresholds sum to at least k. This initial set can contain at most k jobs, because every job has parallelism threshold at least 1. As a result, the model is finite skip with parameter $n = k$.

Work conserving: Whenever jobs are present in the system whose parallelism thresholds sum to at least k, all servers are occupied, and the system is maximally busy. Whenever k jobs are present, the system must be maximally busy.

Positive service rate when nonempty: If a job is present in the system, at least one server must be occupied, and so the service rate is at least $1/k$. Hence $b_{\text{inf}} \geq 1/k$.

4.4.4 Multiserver-jobs under the ServerFilling policy

First, we will describe the multiserver-job setting. Then we will specify the ServerFilling policy.

Multiserver-Job Setting

When we look at jobs in cloud computing systems [148] and in supercomputing systems [33, 61, 210], jobs commonly require an exact number of servers for the entire time the job is in service. To illustrate, in Fig. 4.6 we show the distribution of the number of CPUs requested by the jobs in

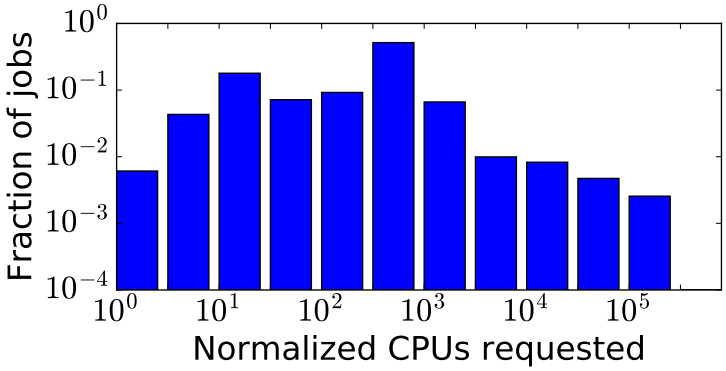


Figure 4.6: The distribution of number of CPUs requested in Google's recently published Borg trace [218]. Number of CPUs is normalized to the size of the smallest request observed, not an absolute value.

Google's recently published trace of its "Borg" computation cluster [83, 218]. The distribution is highly variable, with jobs requesting anywhere from 1 to 100,000 normalized CPUs[2].

The Multiserver-Job (MSJ) model is a natural model for these computing systems – it is a multiserver queueing model where each job requires a fixed number of servers.

Specifically, in an MSJ model, a job j has two requirements: A number of servers v_j and an amount of time x_j, which are sampled i.i.d. from some joint distribution (V, X). If job j requires v_j servers, then it can only be served when exactly v_j servers are allocated to it. The job will complete after x_j time in service.

Let a job j's size be defined as

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

There are a wide variety of possible service policies for placing jobs at open servers, including FCFS, MaxWeight, Most Servers First and many others. (We formally define these policies in Section 4.8.) As examples, in Fig. 4.4, we show the scaled mean response time of Multiserver-Job models under a variety of service policies, where the joint distribution (V, X) is $(1, Exp(\frac{1}{2}))$ with probability $\frac{1}{2}$, and $(4, Exp(\frac{2}{3}))$ with probability $\frac{1}{2}$.

Unfortunately, no existing policies fit within the WCFS framework – all existing policies, including those shown in Fig. 4.4, are either non-finite-skip, such as Most Servers First, or non-work-conserving, such as FCFS. Correspondingly, in Fig. 4.4, we see that no existing policy has its scaled mean response time converge to the same limit as $M/G/1/FCFS$.

We therefore define a novel service policy called ServerFilling which yields a WCFS model. The scaled mean response time of this service policy is depicted in Fig. 4.3, with the same joint distribution (V, X) as the policies shown in Fig. 4.4.

[2]The data was published in a scaled form [218]. We rescale the data so the smallest job in the trace uses one normalized CPU.

ServerFilling

For simplicity, we initially define the Server Filling policy for the common situation in computer systems where all jobs require a number of servers which is a power of 2 (V is always a power of 2), and where k is also a power of 2. We discuss generalizations in Section 4.4.4.

First, ServerFilling designates a candidate set M, consisting of the minimal prefix (i.e. initial subset) of the jobs in the system in arrival order which collectively require at least k servers. If all jobs in the system collectively require fewer than k servers, then all are served. Note that $|M| \leq k$ because all jobs require at least 1 server.

For instance, if $k = 8$ and the jobs in the system require $[1, 2, 1, 1, 4, 2, 2, 1]$ servers, in arrival order (reading from left to right), then M would consist of the first 5 jobs: $[1, 2, 1, 1, 4]$, which collectively require 9 servers.

Next, the jobs in M are ordered by their server requirements v_j, from largest to smallest, tiebroken by arrival order. Jobs are placed into service in that order until no more servers are available. For instance, if $k = 8$ and M contains jobs requiring $[1, 2, 1, 1, 4]$ servers, then jobs requiring $4, 2, 1$, and 1 server(s) would be placed into service.

To show that ServerFilling fits within WCFS with $n = k$, we must show that ServerFilling always utilizes all k servers if at least k jobs are in the system.

Lemma 4.4.1. *Let M be a set of jobs such that $\sum_{j \in M} v_j \geq k$, where each $v_j = 2^i$ for some i and $k = 2^{i'}$ for some i'. Label the jobs $m_1, m_2, \ldots$ in decreasing order of server requirement: $v_{m_1} \geq v_{m_2} \geq \ldots$. Then there exists some index $\ell \leq |M|$ such that*

$$\sum_{j=1}^{\ell} v_{m_j} = k.$$

Proof. Let $\text{REQ}(z)$ count the number of servers required by the first z jobs in this ordering:

$$\text{REQ}(z) = \sum_{j=1}^{z} v_{m_j}.$$

We want to show that $\text{REQ}(\ell) = k$ for some ℓ. To do so, it suffices to prove that:

$$\text{There exists no index } \ell' \text{ such that both } \text{REQ}(\ell') < k \text{ and } \text{REQ}(\ell' + 1) > k. \tag{4.1}$$

Equation (4.1) states that $\text{REQ}(z)$ cannot cross from below k to above k without exactly equalling k. Because $\text{REQ}(0) = 0$ and $\text{REQ}(|M|) \geq k$, $\text{REQ}(\ell)$ must exactly equal k for some ℓ.

To prove (4.1), let us examine the quantity $k - \text{REQ}(z)$, the number of remaining servers after z jobs have been placed in service. Because all v_js are powers of 2, $k - \text{REQ}(z)$ carries an important property:

$$\text{For all } z, k - \text{REQ}(z) \text{ is divisible by } v_{m_{z+1}}. \tag{4.2}$$

We write $a|b$ to indicate that a divides b.

We will prove (4.2) inductively. For $z = 0$, $k - \text{REQ}(0) = k$. Because k is a power of 2, and v_{m_1} is a power of 2 no greater than k, the base case holds. Next, assume that (4.2) holds for

some index z, meaning that $v_{m_{z+1}} | (k - \text{REQ}(z))$. Note that $\text{REQ}(z+1) = \text{REQ}(z) + v_{m_{z+1}}$. As a result, $v_{m_{z+1}} | (k - \text{REQ}(z+1))$. Now, note that $v_{m_{z+2}} | v_{m_{z+1}}$, because both are powers of 2, and $v_{m_{z+2}} \leq v_{m_{z+1}}$. As a result, $v_{m_{z+2}} | (k - \text{REQ}(z+1))$, completing the proof of (4.2).

Now, we are ready to prove (4.1). Assume for contradiction that there does exist such an ℓ'. Then $k - \text{REQ}(\ell') > 0$, and $k - \text{REQ}(\ell'+1) < 0$. Because $\text{REQ}(\ell'+1) = \text{REQ}(\ell') + v_{m_{\ell'+1}}$, we therefore know that $v_{m_{\ell'+1}} > k - \text{REQ}(\ell')$. But from (4.2), we know that $v_{m_{\ell'+1}}$ divides $k - \text{REQ}(\ell')$. A larger positive integer cannot divide a smaller positive integer, so this is a contradiction, as desired. □

ServerFilling for the Multiserver-Job system is a WCFS policy

Finite skip: The jobs in service are a subset of the candidate set M, the initial set of jobs in arrival order whose server requirements v_j sum to at least k. This initial set must contain at most k jobs, because every job requires at least 1 server. As a result, the model is finite skip with parameter $n = k$.

Work conserving: By Lemma 4.4.1, whenever jobs are present in the system whose server requirements v_j sum to at least k, all servers are occupied, and the system is maximally busy. Thus, whenever k jobs are present, the system must be maximally busy.

Positive service rate when nonempty: If a job is present in the system, at least one server must be occupied, and so the service rate is at least $1/k$. Hence $b_{\text{inf}} \geq 1/k$.

Generalizations of ServerFilling

The ServerFilling policy can be generalized, as long as all server requirements divide k. We describe the corresponding scheduling policy, which we call DivisorFilling, in Section 4.7.

DivisorFilling is the most general possible WCFS policy for the MSJ setting. If some server requirement does not divide k, then no policy fits within the WCFS framework, because the system is not be work conserving if all jobs present require that non-divisible number of servers, for any number of jobs present.

4.5 Prior Work

4.5.1 M/G/k

Fixed k

In this regime, the best known bounds on response time either require much stronger assumptions on the job size distribution S than we assume [144], or prove much weaker bounds on mean response time [132, 133].

A paper by Loulou [144] bounds mean work in system in the M/G/k to within an additive gap, under the strong assumption that the job size distribution S is bounded. This result is comparable to our Lemma 4.6.2, but in a simpler setting and under stronger assumptions. While the paper mostly focuses on the overload regime ($\rho > 1$), their equations (9) and (10) apply in our setting

($\rho < 1$) as well. They couple the multiserver system with a single-server system on the same arrival sequence. They show that

$$0 \leq W^{M/G/k}(t) - W^{M/G/1}(t) \leq k \max_{1 \leq i \leq A(t)} S_i,$$

where $A(t)$ is the number of jobs that have arrived by time t. In the case of a bounded job size distribution S, one can therefore show that

$$0 \leq W^{M/G/k}(t) - W^{M/G/1}(t) \leq k \sup(S). \tag{4.3}$$

One could then use this workload bound to prove a bound on mean response time in the M/G/k.

These bounds are comparable to those in our Lemma 4.6.2, but our bounds require a much weaker assumption on the job size distribution S.

Köllerström [132] proves convergence of queueing time to an exponential distribution in the GI/GI/k. Specialized to the M/G/k, the result states that in the $\rho \to 1$ limit, $T_Q^{M/G/k}$ converges to an exponential distribution with mean

$$\frac{\rho}{1-\rho} \frac{\mathbb{E}[S^2]}{2\mathbb{E}[S]} - \frac{1}{\lambda} = \mathbb{E}[T_Q^{M/G/1}] - \frac{1}{\lambda}.$$

Köllerström [133] improves upon [132] by characterizing the rate of convergence and thereby derives explicit moment bounds. However, unlike prior single-server results [128], these bounds are quite weak. Specialized to the M/G/k, Köllerström [133]'s bounds state that

$$\mathbb{E}[T_Q^{M/G/k}] - \mathbb{E}[T_Q^{M/G/1}] \geq \frac{c_{lower}}{(1-\rho)^{1/2}} \tag{4.4}$$

$$\mathbb{E}[T_Q^{M/G/k}] - \mathbb{E}[T_Q^{M/G/1}] \leq \frac{c_{higher}}{1-\rho} \tag{4.5}$$

for constants c_{lower}, c_{higher} not dependent on ρ.

The $\Theta(\frac{1}{1-\rho})$ scaling in (4.5) is especially poor: this bound is too weak to give any explicit bound on the convergence rate of $\mathbb{E}[T_Q^{M/G/k}](1-\rho)$ to the previously established limit of $\frac{\mathbb{E}[S^2]}{2\mathbb{E}[S]}$.

Our bounds are tighter in that they are constants not depending on ρ, but we assume S has finite $\mathrm{rem}_{\sup}$, while Köllerström [133] merely assumes that S has finite second moment.

Scaling k

Recent work has focused on regimes where both ρ and k scale asymptotically, such as the Halfin-Whitt regime. These results are not directly comparable to ours; they indicate that the limiting behavior in the Halfin-Whitt regime depends in a complex way on the job size distribution S [6, 43, 69].

Turning to the more general case of scaling k, in work currently under submission, Goldberg and Li [75] prove the first bounds on $\mathbb{E}[T_Q]$ that scale as $\frac{c}{1-\rho}$ for an explicit constant c and arbitrary k as a function of ρ. Unfortunately, the constant c is enormous, scaling as $10^{450}\mathbb{E}[S^3]$. In contrast, we focus on the regime of fixed k, and prove tight and explicit bounds on mean response time. Goldberg and Li [75] also provide a highly detailed literature review on bounds on $\mathbb{E}[T_Q]$ and related measures in the M/G/k and related models.

4.5.2 Heterogeneous M/G/k

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 [8, 54, 55, 139]. 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 [8]. 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.

Heterogeneous M/H$_m$/k

Van Harten and Sleptchenko [221] 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.

M/(M+G)/2 Model

Another intermediate model is the M/(M+G)/2 model of Boxma et al. [28]. 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. [28] derive 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, under both FCFS service and related service disciplines [123, 184, 192].

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 in solving even the simplest heterogeneous multiserver models. In contrast, our WCFS framework handles both differentiated jobs and an arbitrary number of servers with no additional effort.

4.5.3 Limited Processor Sharing

The Limited Processor Sharing policy has been studied by a wide variety of authors [91, 170, 216, 240–242], but none bound mean response time for all loads ρ.

Asymptotic Regimes

A series of papers by Zhang, Dai and Zwart [240–242] derive the strongest known results on Limited Processor Sharing in a variety of asymptotic regimes. These authors derive the measure-valued fluid limit [241], the diffusion limit [242] and a steady-state approximation [240], which they prove is accurate in the heavy traffic limit ($\rho \to 1$). The most comparable of their results to our work is their steady-state approximation. When specialized to mean response time in the M/G/1/LPS, their approximation states that

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

They prove that this approximation is accurate in the heavy-traffic limit; they do not provide specific error bounds, but empirically show the approximation performs well at all load ρ [240]. Our results therefore complement the results of Zhang et al., by proving concrete error bounds, in contrast to their approximation.

State-dependent Server Speed

To model the behavior of databases, Gupta and Harchol-Balter [91] introduce a variant of the Limited Processor Sharing model, where the total server speed is a function of the number of jobs in service. In their setting 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). While this two-moment approximation is not known to be tight, it indicates that 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.

Using our WCFS framework it is possible to derive bounds on mean response time for the state-dependent server speeds setting. For MPL parameters less than or equal to the service-rate-maximizing MPL, both our upper and lower bounds apply, while if the MPL parameter is greater than the service-rate-maximizing MPL, only our upper bounds apply, because the system only partially fulfills our definition of work conservation.

Subsequently, Telek and Van Houdt [216] derive the Laplace-Stieltjes transform of response time in the LPS model with state-dependent server speed, under phase-type job sizes. Unfortunately, the transform takes the form of a complicated matrix equation, making it difficult to derive general insights across general job size distributions. Instead, the authors numerically invert the Laplace transform for a handful of specific distributions to derive empirical insights. This is in contrast to our simple, explicit and tight bounds on mean response time in Theorem 4.6.2.

4.5.4 Threshold Parallelism

Berg et al. [19] [18] introduce the concept of "speedup functions" to capture the common situation in Machine Learning and other highly parallel computing settings where different jobs can be parallelized to different degrees. One important kind of speedup function is the Threshold

Parallelism model, where the service rate of a job is proportionate to the number of servers. However, results are only known for models with at most two speedup functions, and where the job size distribution is exponential, 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 Threshold Parallelism with general job size distributions and arbitrary parallelism thresholds, under FCFS service.

Elastic and Inelastic Jobs

A special case of Threshold Parallelism the Elastic/Inelastic model of Berg et al. [20]. 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 minimizes mean response time. 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 using a combination of the Busy-Period Transitions technique and Matrix-Analytic methods, to evaluate their multidimensional Markov chain. This gives a numerical approximation that is empirically within 1% of simulation.

Our bounds on Threshold Parallelism with FCFS service are 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.

4.5.5 Multiserver Jobs

The Multiserver-Job model has been extensively studied, in both practical [33, 61, 210] and theoretical settings [31, 71, 113, 115, 147, 148, 180, 181, 187]. It captures the common scenario in datacenters and supercomputing where each job requires a fixed number of servers in order to run. 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 scheduling policy, prior to our bound on ServerFilling.

FCFS Scheduling

The most natural policy is FCFS, where the oldest jobs are placed into service until a job requires more servers than remain, at which point the queue is blocked. Therefore, the FCFS policy can leave a large number of servers idle even when many jobs are present. As a result, FCFS does not in general achieve an optimal stability region. Even worse, deriving the stability region of FCFS is an open problem, and has only been found in a few special cases [31, 187].

One technique that may be useful for characterizing this stability region is the *saturated system* approach [13, 65]. The saturated system is a system in which additional jobs are always

available, so the front is always full, only the composition of jobs in the front varies. The completion rate of the saturated system exactly matches the stability region of the equivalent open system, under a wide variety of arrival processes. Unfortunately, solving the general Multiserver-job FCFS saturated system seems intractable.

Given the difficulty of proving results under FCFS scheduling, finding policies with better theoretical guarantees, such as ServerFilling, is desirable.

MaxWeight Scheduling

One natural throughput-optimal policy is the MaxWeight policy [148]. To express this policy, divide all jobs into classes based on server requirements, with $N_i(t)$ denoting the number of jobs requiring i servers in the system at time t. Next, consider the set Z of all packings of jobs onto servers. Let z be one particular packing, with z_i the number of jobs requiring i servers served by packing z.

The MaxWeight service policy picks the packing z which maximizes

$$\max_z \sum_i N_i(t) z_i.$$

While MaxWeight is throughput optimal, it is very computationally intensive to implement, requiring the scheduler to solve an NP-hard optimization problem whenever a job arrives or departs. For comparison, ServerFilling is also throughput-optimal given our assumptions on the server requirements V, but it is far computationally simpler, requiring approximately linear time as a function of k. Moreover, no bounds on mean response time are known for MaxWeight, due in part to its high complexity.

Nonpreemptive Scheduling

In certain practical settings such as supercomputing, a nonpreemptive service policy is preferred. In such settings, a backfilling policy such as EASY backfilling or conservative backfilling is often used [33, 61, 210]. These start by serving jobs in FCFS order, until a job is reached that requires more servers than remain. At this point, jobs further back in the queue that require fewer servers are scheduled, but only if they 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 a challenging problem. Several such policies have been designed [71, 147, 181], 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, which 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, Our bounds on the mean response time of ServerFilling can serve as such a baseline, albeit in the more permissive setting of preemptive scheduling.

4.6 Theorems and Proofs

We perform a heavy traffic analysis within our WCFS framework, assuming finite $\mathrm{rem}_{\sup}(S, C)$. Specifically, we prove that the scaled mean response time of any WCFS model converges to the same constant as an M/G/1/FCFS:

Theorem 4.6.1 (Heavy Traffic response time)**.** *For any model $\pi \in$ WCFS, if $\mathrm{rem}_{\sup}(S, C)$ is finite,*

$$\lim_{\rho \to 1} \mathbb{E}[T^\pi](1-\rho) = \frac{\mathbb{E}[S^2]}{2\mathbb{E}[S]}.$$

To prove Theorem 4.6.1, we prove a stronger theorem, tightly and explicitly bounding $\mathbb{E}[T^\pi]$ up to an additive constant, for any $\pi \in$ WCFS.

Theorem 4.6.2 (Explicit response time bounds)**.** *For any model $\pi \in$ WCFS, if $\mathrm{rem}_{\sup}(S, C)$ is finite,*

$$\mathbb{E}[T^\pi] \leq \frac{\rho}{1-\rho}\frac{\mathbb{E}[S^2]}{2\mathbb{E}[S]} + c^\pi_{upper}$$
$$\mathbb{E}[T^\pi] \geq \frac{\rho}{1-\rho}\frac{\mathbb{E}[S^2]}{2\mathbb{E}[S]} + c^\pi_{lower}$$

for explicit constants c^π_{upper} and c^π_{lower} not dependent on load ρ.

Proof deferred to Section 4.6.1. □

From Theorem 4.6.2, Theorem 4.6.1 follows via a simple rearrangement:

$$\frac{\rho}{1-\rho}\frac{\mathbb{E}[S^2]}{2\mathbb{E}[S]} = \frac{\frac{\mathbb{E}[S^2]}{2\mathbb{E}[S]}}{1-\rho} - \frac{\mathbb{E}[S^2]}{2\mathbb{E}[S]}.$$

Theorem 4.6.2 also implies rapid convergence of scaled mean response time to its limiting constant for any WCFS policy:

Corollary 4.6.1. *For any model $\pi \in$ WCFS, if $\mathrm{rem}_{\sup}(S, C)$ is finite,*

$$\mathbb{E}[T^\pi](1-\rho) = \frac{\mathbb{E}[S^2]}{2\mathbb{E}[S]} + O(1-\rho).$$

4.6.1 Outline of Proof of Theorem 4.6.2

We will prove Theorem 4.6.2 where

$$c^\pi_{upper} = (n-1)\mathrm{rem}_{\sup}(S, C) + \frac{n\mathbb{E}[S]}{b_{\inf}},$$
$$c^\pi_{lower} = -(n-1)\mathrm{rem}_{\sup}(S, C) + \mathbb{E}[S],$$

where n denotes the size of the front, and where $b_{\inf}$ is defined in Section 4.3.1.

Our goal is simply to prove the bounds in Theorem 4.6.2 for some constants $c^{\pi}_{upper}, c^{\pi}_{lower}$ independent of ρ; we have made no effort to optimize these constants, leaving that to future work. Specifically, for our motivating models, the $\frac{n}{b_{\inf}}$ term scales as $O(n^2)$ as the front size n grows. For these models this term is unnecessarily loose, and could easily be lowered to an $O(n)$ bound by using a more detailed view.

We start our proof of Theorem 4.6.2 in Section 4.6.2 by discussing two different views of a WCFS system: The *omniscient view* and the *limited view*. We will make use of these two views throughout our proof. Next, we split response time T into two pieces, queueing time T_Q and front time T_F, and bound the expectation of each separately.

We first bound $\mathbb{E}[T_Q]$, which forms the bulk of our proof. The two key ideas come from the intuition that a WCFS model behaves like a FCFS M/G/1 system. In Lemma 4.6.1, we prove that $\mathbb{E}[T_Q] = \mathbb{E}[W] + c$, for some constant c; in a WCFS model, jobs progress through the system in essentially FCFS order, and as $\rho \to 1$ work is completed essentially at rate 1.

In Lemma 4.6.2, we prove that $\mathbb{E}[W] = \mathbb{E}[W^{M/G/1}] + c$, for some constant c. The key idea here is that in a WCFS model, if W is large, work arrives and completes in exactly the same way as in an M/G/1. In particular, in a WCFS model, if the front is not full, then W cannot be large.

In Lemma 4.6.3, we combine Lemma 4.6.1 and Lemma 4.6.2 to prove that $\mathbb{E}[T_Q] = \mathbb{E}[T^{M/G/1}] + c$ for some constant c.

In Lemma 4.6.4, we prove that work W is indeed stationary with finite mean. This is a technical lemma that rules out pathological scenarios, which is necessary because our WCFS class of models is very general. Lemma 4.6.4 is used by both Lemma 4.6.1 and Lemma 4.6.2.

Finally, in Lemma 4.6.5, we bound $\mathbb{E}[T_F]$, utilizing Little's law.

Combining Lemmas 4.6.3 and 4.6.5 proves Theorem 4.6.2.

4.6.2 Two Views

At several steps in our proof of Theorem 4.6.2, we will make use of two different views of the queueing system, corresponding to two different state descriptors:

Omniscient view: In the omniscient view the state descriptor consists of the remaining size and class of all jobs in the system; we sample jobs' sizes and classes when the jobs enter the system. For a given system state, work is a deterministic quantity.

Limited view: In the limited view, the state descriptor consists of the age and class of the jobs in the front, and the number of jobs in the queue. We sample jobs' classes when they enter the front, and determine whether jobs complete according to the hazard rate of the job size distribution, as the job ages. For a given system state, work is a random variable.

We will make it clear which view of the system we are using in each step of the proof. Generally, the omniscient view is useful when analyzing total work in the system, and the limited view is useful when analyzing work at the front.

4.6.3 Lemma 4.6.1: $\mathbb{E}[T_Q]$ and $\mathbb{E}[W]$

First, we prove that mean queueing time and mean work are similar:

Lemma 4.6.1 (Queueing time and work)**.** *For any model $\pi \in$ WCFS, if $\mathrm{rem}_{\mathrm{sup}}(S, C)$ is finite,*

$$\mathbb{E}[W] - (n-1)\mathrm{rem}_{\mathrm{sup}}(S, C) \leq \mathbb{E}[T_Q] \leq \mathbb{E}[W].$$

Proof. start by writing time in queue T_Q in terms of work in system. Let us consider the omniscient view of the system, so work W is a deterministic quantity given the system state. Consider an arbitrary tagged job j. When j arrives, let $W^A(j)$ be the amount of work j sees in the system, and let $W_F^F(j)$ be the amount of work j sees in the front other than j itself, when j leaves the queue and enters the front. In W_F^F, the subscript F indicates that we are looking at the amount of work at the front, and the superscript F indicates that we are looking at the moment when j enters the front.

Because the model is finite-skip, jobs move from the queue to the front in arrival order, so 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. As a result, the amount of work which is completed while j is in the queue is exactly $W^A(j) - W_F^F(j)$. Note that if j enters the front upon arrival to the system, $W^A(j) = W_F^F(j)$, and no work is completed while j is in the queue.

While j is in the queue, the front must be full; the system must be maximally busy during this time, completing work at rate 1. Job j is in the queue for $T_Q(j)$ time, so the system must complete $T_Q(j)$ work during that time. We can therefore conclude that

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

Because j is an arbitrary job, we can write $W_F^F(j)$ as W_F^F, a random variable over all jobs that pass through the system. Likewise, $T_Q(j)$ is simply T_Q. Because Poisson arrivals see time averages, $W^A(j) \sim W$, the time-stationary amount of work in the system. Combining these equivalencies, we find that

$$W - W_F^F = T_Q, \tag{4.6}$$

where W_F^F represents the work seen at the front when a job enters the front. Note that W is time-stationary, while W_F^F and T_Q are event-stationary.

To rigorously demonstrate (4.6), we need to prove that the system converges to a stationary distribution, which we prove in Lemma 4.6.4.

To give bounds on W_F^F, we switch to the limited view of the system, where the state of the front consists of the classes and ages of the jobs at the front. We have two simple bounds on W_F^F: First, $W_F^F \geq 0$. Next, since $W_F^F(j)$ is the work of at most $n-1$ jobs, (the jobs at the front when a given job enters the front),

$$\mathbb{E}[W_F^F] \leq (n-1)\mathrm{rem}_{\mathrm{sup}}(S, C).$$

Combining these bounds with (4.6), we can bound $\mathbb{E}[T_Q]$ in terms of $\mathbb{E}[W]$:

$$\mathbb{E}[W] - (n-1)\mathrm{rem}_{\mathrm{sup}}(S, C) \leq \mathbb{E}[T_Q] \leq \mathbb{E}[W].$$

□

4.6.4 Lemma 4.6.2: Bounding $\mathbb{E}[W]$

Lemma 4.6.2. *(Work bounds) For any model $\pi \in$ WCFS, if $\mathrm{rem}_{\sup}(S, C)$ is finite,*

$$\frac{\rho}{1-\rho}\frac{\mathbb{E}[S^2]}{2\mathbb{E}[S]} \leq \mathbb{E}[W] \leq \frac{\rho}{1-\rho}\frac{\mathbb{E}[S^2]}{2\mathbb{E}[S]} + (n-1)\mathrm{rem}_{\sup}(S, C).$$

Proof. Consider the stationary random variable W^2 in the omniscient view, so work is a deterministic quantity at a given time on a given sample path. 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, due to the rate conservation law [157] with respect to W^2.

To calculate the expected rate of decrease, note that, ignoring moments where jobs arrive, $\frac{d}{dt}W(t) = -B(t)$, by definition, where $B(t)$ is the total service rate of the system at time t. As a result, $\frac{d}{dt}W(t)^2 = -2W(t)B(t)$, ignoring arrival epochs. This expected rate of decrease is a well-defined random variable, because the system converges to stationarity. Thus the expected rate of decrease of W^2 is $2\mathbb{E}[WB]$.

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^2 increases from $W(t^-)^2$ to $(W(t^-) + S)^2$, a change of $2W(t^-)S + S^2$. Note that $W(t^-)$ is distributed as W, by PASTA. Note also that W and S are independent, because S is sampled i.i.d.. As a result, the expected increase per arrival is $2\mathbb{E}[W]\mathbb{E}[S] + \mathbb{E}[S^2]$. Arrivals occur at rate λ. As a result, the expected rate of increase is $2\lambda\mathbb{E}[W]\mathbb{E}[S] + \lambda\mathbb{E}[S^2]$.

To show that these rates are equal, we must show that the rates are finite. This follows from the fact that $\mathbb{E}[W]$ is finite, which we prove in Lemma 4.6.4.

As a result, the rates of increase and decrease of W^2 are equal:

$$\begin{aligned}
2\mathbb{E}[WB] &= 2\lambda\mathbb{E}[W]\mathbb{E}[S] + \lambda\mathbb{E}[S^2] \\
\mathbb{E}[WB] &= \lambda\mathbb{E}[W]\mathbb{E}[S] + \frac{\lambda}{2}\mathbb{E}[S^2] \\
\mathbb{E}[WB] &= \rho\mathbb{E}[W] + \frac{\lambda}{2}\mathbb{E}[S^2] \\
\mathbb{E}[W] - \mathbb{E}[W(1-B)] &= \rho\mathbb{E}[W] + \frac{\lambda}{2}\mathbb{E}[S^2] \\
\mathbb{E}[W](1-\rho) &= \mathbb{E}[W(1-B)] + \frac{\lambda}{2}\mathbb{E}[S^2] \\
\mathbb{E}[W] &= \frac{\mathbb{E}[W(1-B)]}{1-\rho} + \frac{\lambda\mathbb{E}[S^2]}{2(1-\rho)} \qquad (4.7)
\end{aligned}$$

Now, we merely need to bound $\mathbb{E}[W(1-B)]$. We do so by switching to the limited view. Note that

$$\begin{aligned}
\mathbb{E}[W(1-B)] &= \mathbb{E}[W(1-B)\mathbb{1}\{B=1\}] + \mathbb{E}[W(1-B)\mathbb{1}\{B<1\}] \\
&= \mathbb{E}[W(1-B)\mathbb{1}\{B<1\}]
\end{aligned}$$

Because the model is work-conserving, if $B < 1$, the front is not full, and there are at most $n-1$ jobs in the system. Taking expectations over the future randomness of these jobs, at any time t for which $B(t) < 1$,

$$\mathbb{E}[W(t)] \leq (n-1)\text{rem}_{\text{sup}}(S, C)$$

Therefore,

$$\begin{aligned}
\mathbb{E}[W(1-B)\mathbb{1}\{B<1\}] &\leq (n-1)\text{rem}_{\text{sup}}(S,C)\mathbb{E}[(1-B)\mathbb{1}\{B<1\}] \\
&= (n-1)\text{rem}_{\text{sup}}(S,C)\mathbb{E}[1-B] \\
&= (n-1)\text{rem}_{\text{sup}}(S,C)(1-\rho) \\
\mathbb{E}[W(1-B)] &\leq (n-1)\text{rem}_{\text{sup}}(S,C)(1-\rho).
\end{aligned}$$

Substituting this into (4.7), our equation for $\mathbb{E}[W]$, we find that

$$\mathbb{E}[W] \leq \frac{\lambda \mathbb{E}[S^2]}{2(1-\rho)} + (n-1)\text{rem}_{\text{sup}}(S, C).$$

Dropping the first term of (4.7), we also get a lower bound:

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

□

One might alternatively try to prove Lemma 4.6.2 via a coupling argument, by coupling the WCFS system to an M/G/1 with the same arrival process. Unfortunately, this proof strategy does not succeed, for a subtle reason.

One can show that the difference in work between the two systems during an interval when the WCFS system has a full front is bounded by the amount of work in the WCFS system at the beginning of the interval. This is analogous to the many-jobs interval argument used by Grosof et al. [81] to analyze relevant work in the M/G/k/SRPT. The key difference is that in the WCFS setting, we consider total work, not relevant work, meaning that job sizes are not bounded. As a result, while the expected work at the beginning of a full-front interval is bounded, the realization of that work may be arbitrarily large.

A coupling argument would therefore need to bound the relative length of full-front intervals started by different amounts of work, to prove a time-average bound on the gap between $\mathbb{E}[W]$ and $\mathbb{E}[W^{M/G/1}]$. This seems intractable, given the generality of WCFS policies.

Instead, by using a Palm Calculus approach, we directly connect the small expected amount of work in a WCFS system with non-full front to a small expected difference in work between the two systems. We therefore prove Lemma 4.6.2, while avoiding all of the complications of a coupling-based argument.

4.6.5 Lemma 4.6.3: Bounding $\mathbb{E}[T_Q]$

Now, we can bound $\mathbb{E}[T_Q]$ by combining Lemmas 4.6.1 and 4.6.2:

Lemma 4.6.3 (Queueing time bounds)**.** *For any model $\pi \in$ WCFS, if $\mathrm{rem}_{\sup}(S, C)$ is finite,*

$$\mathbb{E}[T_Q^\pi] \leq \frac{\rho}{1-\rho}\frac{\mathbb{E}[S^2]}{2\mathbb{E}[S]} + (n-1)\mathrm{rem}_{\sup}(S, C)$$
$$\mathbb{E}[T_Q^\pi] \geq \frac{\rho}{1-\rho}\frac{\mathbb{E}[S^2]}{2\mathbb{E}[S]} - (n-1)\mathrm{rem}_{\sup}(S, C)$$

4.6.6 Lemma 4.6.4: Finite $\mathbb{E}[W]$

Lemma 4.6.4 (Finite mean work)**.** *For any model $\pi \in$ WCFS, if $\mathrm{rem}_{\sup}(S, C)$ is finite, for any load $\rho < 1$, W is a well-defined stationary random variable and $\mathbb{E}[W]$ is finite.*

Proof. Recall that $W = W_F + W_Q$; we first focus on W_F. There are at most n jobs in the front at any time. In the limited view, each job has expected remaining size at most $\mathrm{rem}_{\sup}(S, C)$, so $\mathbb{E}[W_F] \leq n\mathrm{rem}_{\sup}(S, C)$. We now turn to W_Q.

To prove that W_Q is stationary and well-defined with finite mean, we will apply the "inventory process" results of Sigman and Yao [206], and Scheller-Wolf [193]'s refinement of those results.

We upper bound W_Q by $\mathcal{W}$, which we will write as an inventory process.

$$\mathcal{W} := W\mathbb{1}\{W_Q > 0\}.$$

Here we will use the omniscient view, so $\mathcal{W}(t)$ is a specific value. By proving $\mathcal{W}$ is stationary and well-defined with finite mean, we also show the same is true of W_Q.

To see why, recall from Section 4.3.3 our assumption that the service policy is dependent only on the class and age of jobs at the front. The front therefore evolves in a self-contained, Markovian fashion, except for the process by which jobs move from the queue to the front. To show that W_F is stationary, it suffices to show that the indicator $\mathbb{1}\{W_Q > 0\}$ is stationary. Only the indicator of whether the queue is occupied influences the state of the front, not the specific number of jobs in the queue. As a result, the stationarity of $\mathcal{W}$ also implies the stationarity of W_F. Because $W_Q = (\mathcal{W} - W_F)^+$, the stationarity of $\mathcal{W}$ also implies the stationarity of W_Q.

To write $\mathcal{W}$ as an inventory process as in [206], we must define a process $X(t)$ with stationary and ergodic increments, such that

$$\mathcal{W}(t) = X(t) + L(t),$$

where

$$L(t) := \sup_{0 \leq s \leq t} (-\min\{0, X(s)\})$$

Here $X(t)$ represents the potential workload process, and $L(t)$ corrects for the fact that the queue can empty.

We will apply [193, Theorem 2.2.1], for the special case of the first moment. Note by Remarks 1 and 3, for the first moment of an inventory process, it suffices to show:

- Negative drift: There exists an amount of work $w < \infty$ and a drift rate $\delta > 0$ such that conditioned on $\mathcal{W}(t) \geq w$,

$$\lim_{\epsilon \to 0} \frac{E_{\mathcal{F}_t}[X(t+\epsilon) - X(t)]}{\epsilon} \geq -\delta$$

[Isaac: Define $\mathcal{F}_t$]

- Finite second moment of positive jumps: There exists a constant $k_1 < \infty$ such that

$$\lim_{\epsilon \to 0} E_{\mathcal{F}_t}[((X(t+\epsilon) - X(t))^+)^2] \leq k_1$$

Now, we define the potential workload process $X(t)$ based on $W(t)$ and $W_Q(t)$.

During intervals when $W_Q(t) = 0$, $X(t)$ is constant. If t_0 is the beginning of an interval where $W_Q(t) > 0$, $X(t)$ jumps up by $W(t_0^+)$ at time t_0. During an interval where $W_Q(t) > 0$, $X(t)$ mimics $W(t)$: $X(t)$ rises by S when a job arrives, and decreases at rate 1. If t_1 is the end of an interval where $W_Q(t) > 0$, $X(t)$ jumps down by $W(t_1^-)$ at time t_1.

By construction, $X(t)$ generates $\mathcal{W}(t)$ as an inventory process. For example, let t_1 be the end of an interval where $W_Q(t) > 0$. Assume that the desired relationship between $X(t)$ and $\mathcal{W}(t)$ holds up to time t_1^-. In particular, $\mathcal{W}(t_1^-) = W(t_1^-)$. Then $\mathcal{W}(t_1^+) = 0$, as desired.

Next, we show that $X(t)$ has stationary and ergodic increments. $X(t)$ has two types of increments: First, Poisson arrivals cause increments sampled i.i.d. from S, which are clearly stationary and ergodic. Second, the beginning and end of intervals where $W_Q(t) = 0$ cause increments equal to $W_F(t)$. To prove these increments are stationary and ergodic it suffices to show that the state of the front is stationary and ergodic. This follows from two assumptions we made in Section 4.3.3. First, we assumed that the service policy is dependent only on the state of the front. Second, the front must empty and thereby undergo renewals, because the service rate $B(t)$ is at least b_{inf} whenever the system is nonempty. Thus, $X(t)$ has stationary and ergodic increments.

To demonstrate negative drift, let w be an arbitrary nonzero amount of work. Whenever $\mathcal{W}(t) \geq w$, $X(t)$ has two types of increments: jumps of size S occurring at rate λ, and continuous decrease at rate 1. As a result, the drift of $X(t)$ is $\rho - 1 < 0$.

To demonstrate finite second moment of positive jumps, note that $X(t)$ has two kinds of positive jumps: Jumps of size S, when $W_Q(t) > 0$, and jumps of size $W(t)$, at the beginning of such an interval.

Switching back to the limited view, note that the latter kind of jump consists of the remaining size of at most n jobs. These remaining sizes are distributed as

$$R(a, c) \sim [S_c - a \mid S_c > a]$$

for some age a and class c.

It therefore suffices to show that there exists a constant r such that for all a, c,

$$\mathbb{E}[R(a, c)^2] \leq r < \infty$$

To do so, note that we can write $R(a, c)_e$, the excess of the remaining size distribution, as a mixture of remaining size distributions for different ages. Note that for any distribution Y, the

excess Y_e is equivalent to

$$Y_e \sim [Y - Y_e \mid Y > Y_e].$$

This holds because the forward and backwards renewal times are distributed identically [104, Chapter 23]. By using this construction for $R(a, c)$, we find that

$$\begin{aligned} R(a,c)_e &\sim [R(a,c) - R(a,c)_e \mid R(a,c) > R(a,c)_e] \\ &= [S_c - (a + R(a,c)_e) \mid S_c > a + R(a,c)_e]. \end{aligned}$$

As a result, $a + R(a, c)_e$ is the desired age distribution.

For any age a', $\mathbb{E}[R(a', c)] \leq \text{rem}_{\text{sup}}(S, C)$. Because $R(a, c)_e$ can be written as a mixture of such distributions, $\mathbb{E}[R(a, c)_e] \leq \text{rem}_{\text{sup}}(S, C)$, which is finite by assumption.

We can now bound $\mathbb{E}[R(a, c)^2]$:

$$\begin{aligned} \mathbb{E}[R(a,c)_e] &= \frac{\mathbb{E}[R(a,c)^2)]}{2\mathbb{E}[R(a,c)]} \\ \mathbb{E}[R(a,c)^2] &= 2\mathbb{E}[R(a,c)]\mathbb{E}[R(a,c)_e] \leq 2\text{rem}_{\text{sup}}(S,C)^2 \end{aligned}$$

Thus, the requirements of [193, Theorem 2.2.1] are satisfied, so both $\mathcal{W}$ and W_Q are stationary and well-defined, and have finite mean.

□

4.6.7 Lemma 4.6.5: Bounding $\mathbb{E}[T_F]$

Lemma 4.6.5 (Front time bounds)**.** *For any model $\pi \in$ WCFS,*

$$\mathbb{E}[S] \leq \mathbb{E}[T^F] \leq \frac{n\mathbb{E}[S]}{b_{\text{inf}}}$$

Proof. First, to prove that $\mathbb{E}[T^F] \geq \mathbb{E}[S]$, note that if a job receives service at the maximum possible rate of 1 for the entire time it is in the front, then the job will complete in time S. As a result, $\mathbb{E}[T^F] \geq \mathbb{E}[S]$.

To prove the upper bound, recall that by the non-idling assumption from Section 4.3.1, in all states of the front s where $N_F(s) \geq 1$, the service rate $B(s) \geq b_{\text{inf}}$. Because $N_F(s) \leq n$, we can bound the ratio $B(s)/N_F(s)$ in all $N_F(s) \geq 1$ states:

$$\frac{B(s)}{N_F(s)} \geq \frac{b_{\text{inf}}}{n}.$$

Therefore, in all states,

$$B(s) \geq \frac{b_{\text{inf}}}{n} N_F(s).$$

In expectation, the same must hold:

$$\mathbb{E}[B] \geq \frac{b_{\text{inf}}}{n} \mathbb{E}[N_F]$$

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

$$\rho \geq \frac{b_{\inf}}{n}\lambda\mathbb{E}[T_F]$$
$$\frac{n\mathbb{E}[S]}{b_{\inf}} \geq \mathbb{E}[T_F]$$

□

Note that Lemma 4.6.5 proves a relatively weak bound on $\mathbb{E}[T^F]$, because we have only made the weak assumption that $b_{\inf}$ is positive. In many models, one can prove a stronger bound on $\mathbb{E}[T^F]$ by using more information about the model's dynamics when the front is not full.

From Lemma 4.6.3 and Lemma 4.6.5, Theorem 4.6.2 follows immediately, with explicit formulas for c^{π}_{upper} and c^{π}_{lower}.

4.7 DivisorFilling

The DivisorFilling policy is a Multiserver-job service policy which assumes that all server requirements v_j divide the total number of servers k. The DivisorFilling policy is a WCFS policy with front size $n = k$, as we will show. Finite-skip will be straightforward, the main difficulty is showing work-conservation.

We first define the DivisorFilling policy. DivisorFilling is a preemptive policy, in that when a job completes, the set of jobs in service may change, removing partially-complete jobs from service. The DivisorFilling policy is defined recursively. The policy's behavior with respect to larger k is defined based on its behavior for smaller k. In particular, we will prove work conservation inductively.

Let M be the set of jobs at the front.

To define DivisorFilling, we split into three cases:

- M contains at least $k/6$ jobs with server requirement $v_j = 1$.
- $k = 2^a 3^b$ for some integers a, b, and M contains $< k/6$ jobs with $v_j = 1$.
- k has a prime factor $p \geq 5$ and M contains $< k/6$ jobs with $v_j = 1$.

4.7.1 At least $k/6$ jobs requiring 1 server

First, assume that M contain at least $k/6$ jobs requiring 1 server.

Just as in the ServerFilling policy, label the jobs $f_1, f_2, \ldots$ in decreasing order of server requirement. Let i^* be defined as

$$i^* = \arg\max_i \sum_{\ell=1}^{i} v_{f_\ell} \leq k.$$

In this case, the DivisorFilling policy serves jobs $f_1, \dots f_{i^*}$, as well as any jobs requiring 1 server that fit in the remaining servers. Specifically, DivisorFilling serves

$$k - \sum_{\ell=1}^{i^*} v_{f_\ell}.$$

additional jobs that require 1 servers, or all jobs requiring 1 server if fewer are available.

Work conservation

We want to show that if M contains k jobs, DivisorFilling serves jobs requiring k servers in this case.

Let us write $\text{SUM}_{i^*} := \sum_{\ell=1}^{i^*} v_{f_\ell}$. Because we have at least $k/6$ jobs requiring 1 server, it suffices to show that $\text{SUM}_{i^*} \geq 5k/6$. The remaining servers are filled by the jobs requiring 1 server.

First, note that $\text{SUM}_k \geq k$, because there are k jobs, each requiring at least 1 server. Next, note that $k - \text{SUM}_{i^*} < f_{i^*+1}$, because the $i^* + 1$ job does not fit in service. Because the labels $f_1, f_2, \dots$ are in decreasing order of server requirement, $k - \text{SUM}_{i^*} < f_{i^*}$.

Therefore, to prove that $k - \text{SUM}_{i^*} \leq k/6$, we need only consider sequences of the i^* largest server requirements in M in which all such requirements are greater than $k/6$. We need only consider requirements equal to $k, k/2, k/3, k/4, k/5$.

We enumerate all such sequences. Note that if k is not divisible by all of $\{2, 3, 4, 5\}$, some entries will not apply. This only tightens the resulting bound on $k - \text{SUM}_{i^*}$ for such k.

We list i^* requirements if $\text{SUM}_{i^*} = k$, and $i^* + 1$ otherwise. We write g_{i^*} as a shorthand for $k - \text{SUM}_{i^*}$.

Sequence	g_{i^*}	Sequence	g_{i^*}
k	0	$k/2, k/2$	0
$k/2, k/3, k/3$	$k/6$	$k/2, k/4, k/4$	0
$k/2, k/4, k/5, k/5$	$k/20$	$k/2, k/5, k/5, k/5$	$k/10$
$k/3, k/3, k/3$	0	$k/3, k/3, k/4, k/4$	$k/12$
$k/3, k/3, k/5, k/5$	$2k/15$	$k/3, k/4, k/4, k/4$	$k/6$
$k/3, k/4, k/5, k/5, k/5$	$k/60$	$k/3, k/5, k/5, k/5, k/5$	$k/15$
$k/4, k/4, k/4, k/4$	0	$k/4, k/4, k/4, k/5, k/5$	$k/20$
$k/4, k/4, k/5, k/5, k/5$	$k/10$	$k/4, k/5, k/5, k/5, k/5$	$3k/20$
$k/5, k/5, k/5, k/5, k/5$	0		

In all cases, $k - \text{SUM}_{i^*} \leq k/6$. As a result, DivisorFilling is work conserving in this case.

4.7.2 $k = 2^a 3^b$

Suppose that k is of the form $2^a 3^b$, for some integers a and b, and that the number of jobs in M that require 1 server is less than $k/6$.

Let M_2 be the set of jobs requiring an even number of servers in M, and let M_r be the remaining jobs:

$$M_2 := \{j \mid j \in M, v_j \text{ is even}\}$$
$$M_r := \{j \mid j \in M, v_j \text{ is odd}, v_j > 1\}$$

Note that because 2 and 3 are the only prime factors of k, all jobs in M_r have server requirements divisible by 3.

How we now schedule is based on which is larger: $2|M_2|$, or $3|M_r|$. In this case of a tie, either would be fine, so we arbitrarily select M_2.

If $2|M_2|$ is larger, we will only serve jobs from among M_2. To do so, imagine that we combine pairs of servers, reducing k by a factor of 2, and reducing the server requirement of every job in M_2 by a factor of 2. We now compute which jobs from M_2 DivisorFilling would serve, in this simplified subproblem. DivisorFilling serves the corresponding jobs.

If $3|M_r|$ is larger, we do the same, except that we combine triples of jobs.

Work conservation

If at least k jobs are present, we will show that this process fills all of the servers.

Because there are $< n/6$ jobs requiring 1 server, $|M_2| + |M_3| \geq 5k/6$. As a result, either $2|M_2| \geq k$ or $3|M_r| \geq k$. Consider the case where $2|M_2| \geq k$. The constructed subproblem has $k/2$ servers and $|M_2| \geq k/2$ jobs, so by induction DivisorFilling fills all of the servers in the subproblem. That property is carried over in the main problem. The case where $3|M_r| \geq k$ is equivalent.

4.7.3 k has a prime factor $k \geq 5$

Finally, suppose that k has a prime factor $p \geq 5$, and that M contains $< k/6$ jobs requiring 1 server. Specifically, let p be k's largest prime factor.

Let us form the set M_p consisting of the jobs in M whose server requirements are multiples of p, and M_r consisting of jobs which require more than 1 server, but not a multiple of p. As in Section 4.7.2, if $|M_p| \geq k/p$, we can recurse by combining groups of p servers to fill all of M.

Otherwise, we turn to M_r. Note that all jobs in M_r have server requirements which are divisors of k/p, because their requirements are divisors of k which are not multiples of p.

If $|M_r| \geq k/p$, let us apply the DivisorFilling policy on an arbitrary subset of M_r of size k/p. By induction, DivisorFilling finds a subset of these jobs requiring exactly k/p servers. Let us extract this subset from M_r, creating M_r^1. We repeat this process until we have extracted p subsets, or $|M_r^i| < k/p$ for some i. DivisorFilling serves the extracted subsets.

Work conservation

We must show that the extraction procedure always successfully extracts p subsets, if $|M| = k$.

In the extraction case, note that $|M_p| < k/p \leq k/5$, and that there are $\leq k/6$ jobs requiring 1 server. M_r consists of the remaining jobs. As a result,

$$|M_r| \geq k - k/6 - k/5 = 19k/30.$$

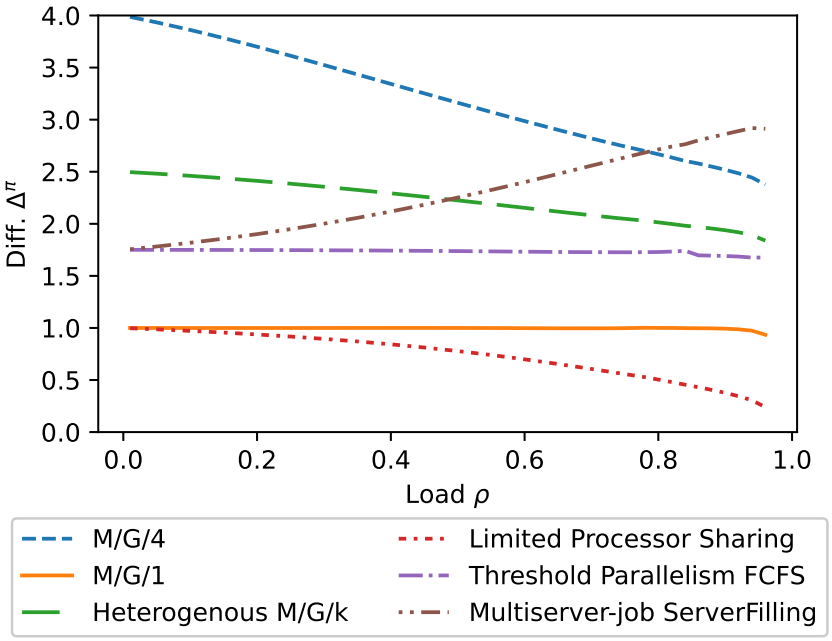


Figure 4.7: Δ^π for WCFS models. Job size distribution S is hyperexponential: $Exp(2)$ w.p. $1/2$, $Exp(2/3)$ otherwise. 10^9 arrivals simulated. $\rho > 0.96$ omitted due to the large amount of random noise under high load. Specific settings: Heterogeneous M/G/k with speeds $[0.4, 0.3, 0.2, 0.1]$. Limited Processor Sharing with Multi-programming Level 4. Threshold Parallelism FCFS with joint random variable (S, L) of $(Exp(2), 1)$ w.p. 1/2, $(Exp(2/3), 4)$ otherwise. Multiserver-job ServerFilling with joint random variable (V, X) of $(1, Exp(1/2))$ w.p. 1/2, $(4, Exp(2/3))$ otherwise.

Note also that every job in M_r requires at least 2 servers, so at most $k/2p$ jobs are extracted at each step. To prove that p subsets can be extracted, we must show that at least k/p jobs remain after $p-1$ subsets have been extracted.

$$|M_r^{p-1}| \geq \frac{19k}{30} - \frac{(p-1)k}{2p} = \frac{19k}{30} - \frac{k}{2} + \frac{k}{2p} = \frac{2k}{15} + \frac{k}{2p}$$

To prove that $|M_r^{p-1}| \geq k/p$, we just need to show that $2k/15 \geq k/2p$. But $p \geq 5$, so $2k/15 > k/10 \geq k/2p$.

Thus, we can always extract p disjoint subsets of jobs, each requiring a total of k/p servers, from M_r. Combining these subsets fills all k servers, as desired.

4.8 Empirical Comparison: WCFS and non-WCFS

We have proven tight bounds on mean response time for all WCFS policies. To quantify the tightness of our bounds, we define the *mean response time difference* Δ^π for a given policy π:

$$\Delta^\pi = \mathbb{E}[T^\pi] - \frac{\rho}{1-\rho}\frac{\mathbb{E}[S^2]}{2\mathbb{E}[S]} = \mathbb{E}[T^\pi] - \mathbb{E}[T_Q^{M/G/1}].$$

For instance, $\Delta^{M/G/1} = \mathbb{E}[S]$.

This definition is useful, as we have shown in Theorem 4.6.2 that for any load ρ, $\Delta^\pi \in [c^\pi_{lower}, c^\pi_{upper}]$, for constants $c^\pi_{lower}, c^\pi_{upper}$ not dependent on ρ, but potentially depending on the model π.

To investigate the behavior of Δ^π, we turn to simulation. We simulate both WCFS models, to confirm our results, as well as non-WCFS models, to show that non-WCFS models typically do not have constant Δ^π in the $\rho \to 1$ limit.

In Fig. 4.7, we simulate WCFS models: our four motivating models from Section 4.4, as well as the simpler M/G/k and M/G/1 models. In each case, we find that Δ^π remains bounded quite close to 0. In particular, we find that Theorem 4.6.2 holds with constants very close to 0.

In Fig. 4.7, we see that for some models, Δ^π increases with ρ, while for others, Δ^π decreases with ρ. Intuitively, this depends on which jobs tend to be prioritized as $\rho \to 1$. Policies which serve many jobs at once, such as the $M/G/4$ and Limited Processor Sharing systems, typically have Δ^π decrease as $\rho \to 1$, because they allow small and large jobs to share service. As a result, small jobs can complete faster than in an M/G/1, lowering Δ^π if ρ is large enough that many jobs are typically in the system.

In contrast, policies which reorder large jobs ahead of small jobs typically have Δ^π increase as $\rho \to 1$, by the same principle. For example, Multiserver-Job ServerFilling prioritizes jobs in the front which require 4 servers. In the setting depicted in Fig. 4.7, such jobs have mean size $3/2$ in this system, compared to the overall mean size $\mathbb{E}[S] = 1$.

In all of the settings simulated in Fig. 4.7, $\Delta^\pi > 0$. This is merely a coincidence, not a general rule, as can be seen in Fig. 4.9b.

Regardless of the different reordering behavior of these different WCFS policies, Δ^π does not diverge as $\rho \to 1$, as predicted by Theorem 4.6.2.

In contrast, in Fig. 4.8, we simulate several non-WCFS models, which we depicted earlier in Fig. 4.4. These models are:

- **Threshold Parallelism Inelastic First:** This is the Threshold Parallelism model from Section 4.4.3, but rather than serving jobs in FCFS order, we prioritize jobs j with smaller parallelism threshold p_j [19].
- **Threshold Parallelism Elastic First:** This is the Threshold Parallelism model from Section 4.4.3, but we prioritize jobs j with larger parallelism threshold p_j.
- **M/G/k/SRPT:** This is an M/G/k, where each of the k servers runs at speed $1/k$, and we prioritize jobs of least remaining size.
- **Multiserver-job FCFS:** This is the Multiserver-job model from Section 4.4.4, but we serve jobs in FCFS order. If the next job to be served doesn't "fit" in the remaining servers,

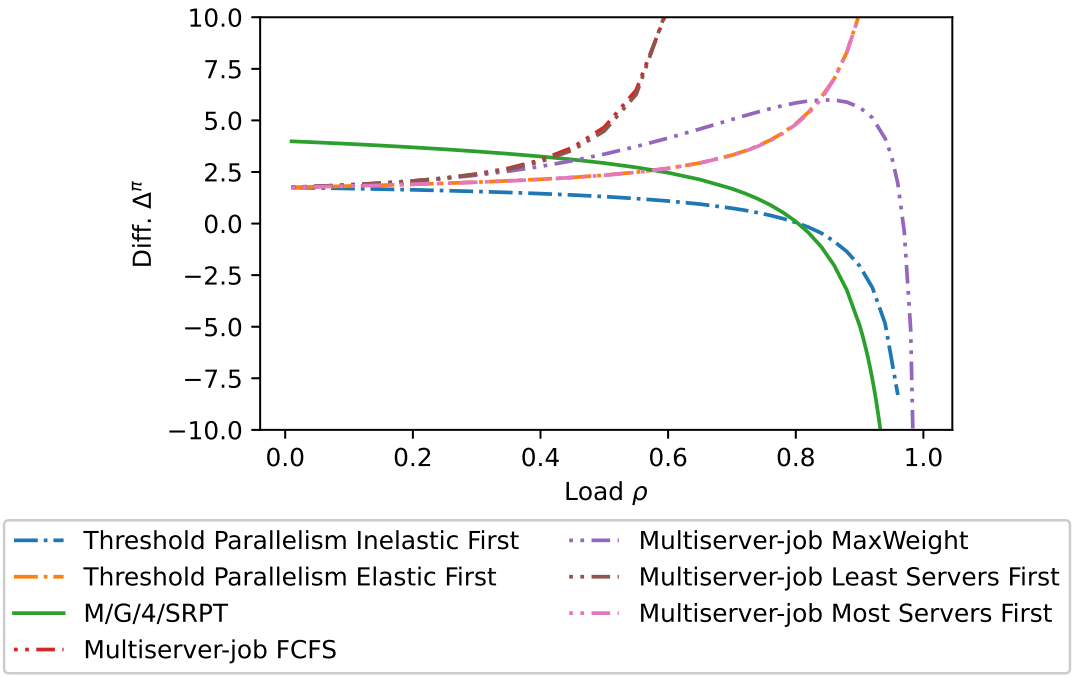


Figure 4.8: Δ^π for non-WCFS models. Same job sizes and specific settings as in Fig. 4.7. Same number of arrivals and range of ρ except MaxWeight: 10^{10} arrivals, $\rho \in [0, 0.99]$.

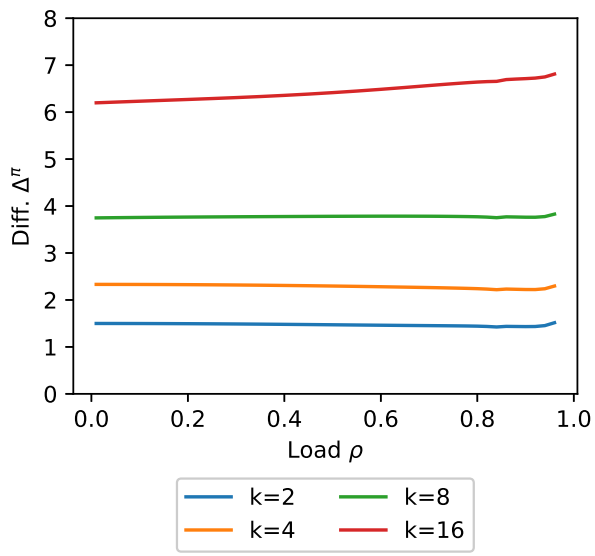
those servers remain idle until other jobs complete, idling sufficient servers to allow the job to fit.

- **Multiserver-job Least Servers First:** This is the Multiserver-job model from Section 4.4.4, but we prioritize jobs j with smaller server requirements v_j. Again, if the next job doesn't fit, the remaining servers remain idle until the job can fit.
- **Multiserver-job Most Servers First:** This is the Multiserver-job model from Section 4.4.4, but we prioritize jobs j with larger server requirements v_j.
- **Multiserver-job MaxWeight:** This is the Multiserver-job model from Section 4.4.4, but we serve jobs according to the "MaxWeight" policy, from the prior literature which we describe in Section 4.5.5.

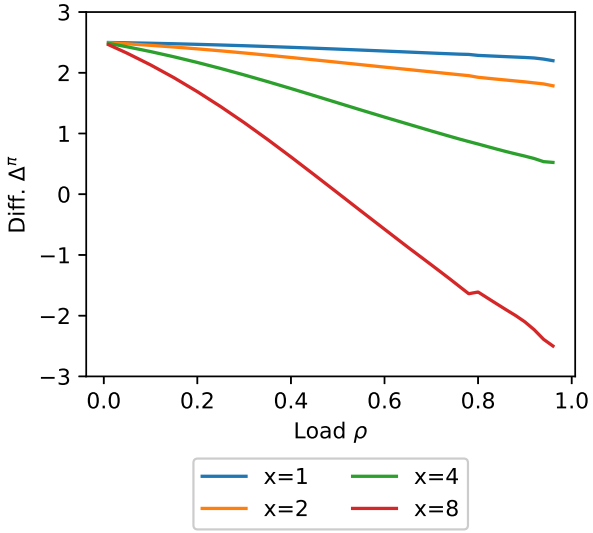
In all cases, prioritization is preemptive.

Our empirical results in Fig. 4.8 indicate that for these non-WCFS policies, Δ^π diverges as $\rho \to 1$. Specifically, for Threshold Parallelism Elastic First, Multiserver-job FCFS, Multiserver-job Least Servers First, and Multiserver-job Most Servers First, Δ^π appears to diverge in the positive direction. For Threshold Parallelism Inelastic First, M/G/k/SRPT, and Multiserver-job ServerFilling, Δ^π appears to diverge in the negative direction. Note the expanded scale of Fig. 4.8 as compared to Fig. 4.7. For Multiserver-job MaxWeight, we performed additional simulation, which indicated that Δ^π diverged in the negative direction as $\rho \to 1$.

This demonstrates that the bounded Δ^π property observed for WCFS models in Fig. 4.7 and proven in Theorem 4.6.2 is highly non-trivial.



(a) Varying front size n. Multiserver-job ServerFilling with $k = [2, 4, 8, 16]$. S distributed $Exp(1)$. Server requirement V distributed uniformly over all integer powers of 2 $\leq k$.



(b) Varying job size distributions. Heterogeneous M/G/4 with speeds $[0.4, 0.3, 0.2, 0.1]$. S distributed hyperexponential: $Exp(1/x)$ with probability $1/2x$, else $Exp((2x-1)/x)$, for $x \in [1, 2, 4, 8]$. $\mathbb{E}[S] = 1, C^2 \approxeq [1, 1.67, 3.57, 7.53]$.

Figure 4.9: Δ^π under WCFS models with varying conditions. Up to 10^9 arrivals simulated.

Next, we explore the behavior of Δ^π for WCFS models, as we vary the front size n and the job size distribution S.

First, in Fig. 4.9a, we investigate the effects of varying front size n on Δ^π for the Multiserver-job model with our ServerFilling policy; under this model, the front size n is equal to the number of servers k. In this setting, the difference Δ^π empirically grows approximately linearly with the number of servers k, and is nearly constant as $\rho \to 1$. This matches the behavior of our bounds proven in Theorem 4.6.2, which expand linearly with n. Our simulations indicate that other WCFS policies similarly experience linear relationships between n and Δ^π.

In Fig. 4.9b we investigate the effects of varying job size distribution S on Δ^π in the Heterogeneous M/G/k where the job size distribution S is parameterized by a real value x. Each S is a hyperexponential distribution with $\mathbb{E}[S] = 1$. At large ages a, the remaining size distributions $[S - a \mid S > a]$ of these job size distributions converge to $Exp(1/x)$, the larger exponential branch. From this, it is straightforward to show that $\text{rem}_{\text{sup}}(S) = x$.

In Fig. 4.9b, we see that as x increases, Δ^π at loads near 1 falls linearly, with more negative slope for larger x. However, for each specific x, it does not appear that Δ^π is diverging to positive or negative infinity. For instance, consider the red curve, $x = 8$: as $\rho \to 1$, Δ^π converges to a value near -3, rather than diverging.

Broadly, Fig. 4.9b matches the behavior of our bounds proven in Theorem 4.6.2, which expand linearly with $\text{rem}_{\text{sup}}(S)$, which here is x. We have empirically found that other WCFS policies similarly experience linear relations between $\text{rem}_{\text{sup}}(S)$ and Δ^π, for hyperexponential job size distributions S, and we believe that similar behavior will occur for other common job

size distributions.

4.9 Technical Conclusion

We introduce the *work-conserving finite-skip* (WCFS) framework, and use it to analyze many important queueing models which have eluded analysis thus far. We prove that the scaled mean response time $\mathbb{E}[T^\pi](1-\rho)$ of any WCFS model π converges in heavy traffic to the same limit as M/G/1/FCFS. Moreover, we prove that the additive gap $\Delta^\pi = \mathbb{E}[T^\pi] - \mathbb{E}[T_Q^{M/G/1}]$ remains bounded by explicit constants at all loads ρ, proving rapid convergence to the heavy traffic limit.

A possible direction for future work would be to to tighten the explicit constants on Δ^π. Doing so will likely require use of more detailed properties of the WCFS models being analyzed, but seems quite doable.

This chapter considers models which are finite skip and work conserving relative to the FCFS service ordering. Another interesting direction would be to investigate policies which are "finite-skip" relative to other base service orderings. Hopefully, one could prove bounds on mean response time of models in this new class relative to an M/G/1 operating under the base service ordering.

Finally, one could try to characterize other metrics of response time for WCFS policies, such as tail metrics. One possible approach to doing so would be to generalize the rate-conservation technique used in Lemma 4.6.2.

4.10 General Conclusion

4.10.1 Summary

We start by summarizing the results proven in this chapter, as well as the key techniques behind these results.

Results: ServerFilling mean response time analysis We prove tight upper and lower bounds on the mean response time of the ServerFilling and DivisorFilling scheduling policies, in the form of a clean mathematical formula (See Theorem 4.6.2). These bounds prove that ServerFilling and DivisorFilling achieve similar mean response time to that of resource-pooled FCFS, in which all k servers are combined into a single ultra-fast server, which runs single-server FCFS. This bound becomes tight as load becomes high (See Theorem 4.6.1). These are the first closed-form bounds on mean response time for any MSJ scheduling policy.

Simulation result: Tight approximation at all loads While our results are tightest in heavy traffic, we show via simulation in Fig. 4.7 that ServerFilling's mean response time consistently lies close to the mean response time of resource-pooled FCFS, across all loads. The same is not true for prior MSJ scheduling policies, as we show via simulation in Fig. 4.8. We further demonstrate the improvement in mean response time of ServerFilling over prior MSJ policies in follow-up work [80].

Results: Other settings We prove similar tight bounds for the entire work-conserving finite-skip (WCFS) class of scheduling policies. The WCFS class includes policies of interest

in a wide variety of settings: FCFS scheduling in the Heterogeneous M/G/k (Section 4.4.1), Limited Processor Sharing (Section 4.4.2), and FCFS scheduling in the threshold parallelism system (Section 4.4.3). Our bounds proven in Theorems 4.6.1 and 4.6.2 apply to all WCFS systems.

Key technique: Work-squared Our key technique is the "work-squared" (W^2) method, where W is the random variable denoting the total amount of work in the system. We use the W^2 method to prove Lemma 4.6.2, a key lemma towards our main result. The W^2 method involves calculating the rate of increase and rate of decrease of W^2. In stationarity, the expected rate of increase and expected rate of decrease must be equal. Using this fact, we can relate relate the overall mean amount of work, $\mathbb{E}[W]$, to the *waste* $\mathbb{E}[W(1-B)]$, the expected product of work and the fraction of idle servers. Here B is a random variable representing the fraction of active servers, so $1-B$ measures the fraction of idle servers.

Because WCFS scheduling policies such as ServerFilling fill all of the servers whenever enough jobs are present, the mean waste of the system is very small. Using the W^2 method, we can therefore show that the overall mean amount of work in the ServerFilling MSJ system is close to the mean work in a resource-pooled M/G/1, where all k servers are combined into a single ultra-fast server.

4.10.2 Subsequent results

The techniques discussed in Section 4.10.1 form the basis for the next two chapters of this thesis, which study more advanced problems in the MSJ setting.

Optimal MSJ scheduling This chapter's main result is devising MSJ scheduling policies whose mean response time can be analyzed. As we show in Chapter 5, one can apply the same ideas to devise scheduling policies with *optimal* mean response time, in the limit as the load on the system approaches its capacity. The policies in this chapter serve jobs in near-FCFS order, and their performance resembles that of resource-pooled FCFS. By contrast, the ServerFilling-SRPT policy, which we consider in Chapter 5, serves jobs in near-SRPT (Shortest Remaining Processing Time) order. We prove that its performance resembles that of resource-pooled SRPT, which by the single-server optimality of SRPT [194], implies our MSJ optimality result.

Analyzing MSJ FCFS While the ServerFilling policy studied in this chapter has many advantageous policies, there are also good reasons to study the FCFS policy. FCFS performs no preemption, and it is used in practice, either standalone or as a component of other MSJ scheduling policies. However, MSJ FCFS is challenging, as it is a non-work-conserving policy: FCFS wastes servers, regardless of how many jobs are present. In Chapter 6, we derive the first mean response time analysis of FCFS in the MSJ setting. In doing so, we derive a much more powerful version of the W^2 method, which can accommodate non-work-conserving settings.

4.10.3 Future Direction: General MSJ Scheduling

This chapter invents the ServerFilling scheduling policy and proves tight bounds on its mean response time. However, the ServerFilling policy is limited to the setting where all jobs' server needs are powers of 2, and where k, the total number of servers, is also a power of 2. Even

the more general DivisorFilling policy is limited to the setting where all jobs' server needs are divisors of k.

As we discuss in Section 8.3.4, in the *general* MSJ scheduling setting, much less is known about mean response time. This is true even in settings where 100% server utilization (i.e. *full* server utilization) is achievable. For instance, consider a system with $k = 3$ servers, server needs $1, 2$, and 3, and where the load of 1-server jobs exceeds that of 2-server jobs. Full server utilization can be achieved by pairing 1-server jobs with 2-server jobs for service, to ensure that servers are never wasted. With full server utilization, the MSJ system starts to look like a resource pooled system. If we pair 1-server and 2-server jobs in arrival order, can we design a MSJ scheduling policy whose mean response time matches that of a resource-pooled single-server scheduling policy, such as resource-pooled FCFS?

We pose the following open problem:

> Under what MSJ workloads can we design a scheduling policy whose mean response time is nearly identical to that of resource-pooled FCFS?

4.10.4 Potential Impact

We now explore potential directions in which the ServerFilling and DivisorFilling MSJ scheduling policies could be applied.

Adopting ServerFilling into Modern Computing Systems

Our analysis of the ServerFilling scheduling policy in the MSJ model shows that ServerFilling achieves consistent, low mean response times, when compared to other common MSJ scheduling policies [80]. Our hope is that our results will lead computing system operators to adopt ServerFilling scheduling in their systems.

However, the road from theoretical results to adoption requires overcoming several hurdles. These include:

Restrictions on server needs. The key idea behind ServerFilling and DivisorFilling is that when the jobs' server needs are easy to pack onto the k servers, we can guarantee mean response time near that of resource-pooled FCFS. This guarantee typically translates to low mean response times, when compared to other common MSJ scheduling policies [80]. ServerFilling and DivisorFilling focus on the scenario where jobs' server needs are *perfect* divisors of the total number of servers k. However, the primary obstacles to packing jobs onto servers are jobs whose server needs are just *over* a divisor of the total number of servers k. For instance, in a $k = 1000$ server system, if many 501-server jobs are present, no two such jobs can be served at once. As a result, it would be difficult to consistently utilize all 1000 servers. By contrast, jobs with server need just under a divisor, such as a 499-server job, do not pose a severe obstacle to full utilization.

Judicious use of preemption. In real systems, preemption can be difficult or expensive, either from an implementation perspective or because of performance overheads. To design a scheduling policy with a lower preemption rate, one could expand the pool of jobs that are under consideration for service, in contrast to ServerFilling. For instance, a policy could

consider for service the $10k$ oldest jobs in the system. A scheduling policy could prioritize keeping servers occupied while avoiding preemption, only performing preemption if jobs with the right server needs are not available in the pool of $10k$ jobs. With a finite limit on the pool size, the resulting policy is still be a WCFS policy, and the methods of this chapter still bound its mean response time.

Avoiding "starvation". If 1-server jobs are very rare, and only one such job is among the oldest jobs in the system, the ServerFilling policy will not serve that job until another 1-server job comes along, or the system empties out. This will lead to to very long response times for those rare jobs. This undesirable dynamic is referred to as "starvation" in the computer systems community. One approach to alleviate this starvation would be to use a policy which is a hybrid of the ServerFilling policy and a policy which strictly prioritizes the oldest job in the system, such as FCFS or a backfilling policy. Using the methods of this chapter and Chapter 6, one could bound prove strong bounds on the mean response time of each type of job in that system, including rare 1-server jobs.

Multidimensional resources. In real systems, jobs often require a variety of resources, such as CPU cores, GPUs, memory, network bandwidth, disk IO, and more. These can be modeled as multidimensional resource requirements, in contrast to the single-dimensional server need model considered in this chapter. In a general multidimensional setting, full utilization is not attainable. There is no equivalent of the power-of-2 server need assumption which can ensure that any assortment of jobs will always be able to use all of the system's resources. Jobs will use more of one resource or another, inevitably wasting some resources. The more flexible "finite-skip" class of policies which we study in Chapter 6 can handle such non-work-conserving settings. If a single bottleneck resource exists, then we can focus on fully utilizing that one resource, and this chapter's policies may be applicable. We discuss scheduling under multidimensional resource constraints further in Section 8.3.4.

ServerFilling for Scheduling People to Handle Group Tasks

Given how good ServerFilling is at delivering consistently low mean response time for computing systems, it is natural to apply it to handling people. For example, in a contracting-based business, different contracts might require different sizes of teams, and a ServerFilling approach could be employed to schedule those contracts. A landscaping company might have a variety of lawns to tend to, requiring different numbers of people based on the size of the job. Here contracts are modeled as jobs, while people are modeled as servers.

However, challenges arise when scheduling people. Preemption must be handled carefully, as people cannot effectively hop on and off of different projects without significant training and delay. Moreover, people are typically not interchangeable: Each person has different skills and needs. To employ the ServerFilling approach in an employment context, it makes sense to form groups of people who can complete a variety of jobs, including many of the oldest jobs in the system. This ad-hoc team formation allows for the consideration of individual skills. The group should stick together through those jobs, ensuring that no preemption is needed and no one is left with nothing to do. Teams ideally will only be reformed after a significant number of jobs

are completed.

As long as everyone's abilities are consistently being employed, and jobs are completed in near-FCFS order, the WCFS results of this section indicate that the resulting policy will achieve mean response times similar to that of resource-pooled FCFS, which is often better than prior MSJ scheduling policies [80].

Chapter 5

Optimal Multiserver-job Scheduling: ServerFilling-SRPT

This chapter is based on my paper "Optimal Scheduling in the Multiserver-job Model under Heavy Traffic", published in SIGMETRICS in 2023, written with my coauthors Ziv Scully, Mor Harchol-Balter, and Alan Scheller-Wolf [87].

5.1 General Introduction

Modeling large-scale computing Modern computing systems scale up in two important ways. First, they contain huge numbers of servers, allowing them to process many jobs at once. Second, the *jobs* in modern computing systems require vastly different amounts of resources, ranging from a single CPU core to thousands of machines. As a result, the amount of work that a single job represents can vary enormously, both in terms of the amount of resources required at one time, and in terms of the duration for which those resources are required. In order to handle this variability and achieve good performance, good *scheduling* decisions are critical. The scheduling decision consists of selecting a combination of jobs to serve at once. Improving the scheduling policy can improve mean response time by orders of magnitude with no additional resources.

To capture the behavior of these computing systems, we use a *multiserver-job* (MSJ) queueing model, depicted in Fig. 5.1, which was introduced in Chapter 4. Jobs arrive randomly over time, and wait in a central queue. Each job has a *server need*, which specifies the number of servers the job requires in order to enter service. The job requires a fixed number of servers throughout its time in service. All servers are identical, so the scheduling policy can decide to serve any set of jobs with total server need at most k, where k is the total number of servers in the system. An example of a scheduling policy is the First-Come-First-Served (FCFS) policy, which we depict in Fig. 5.1. FCFS places the oldest jobs in the system into service one-by-one, until a job is reached whose server need exceeds the number of currently available servers. At this point, the blocked job, and all jobs behind it in the queue, must wait until some of the jobs in service complete and more jobs become available.

In this chapter, we allow the scheduling policy to know the service duration of each job in advance. We also allow the scheduling policy to perform *preemption*, which refers to pausing

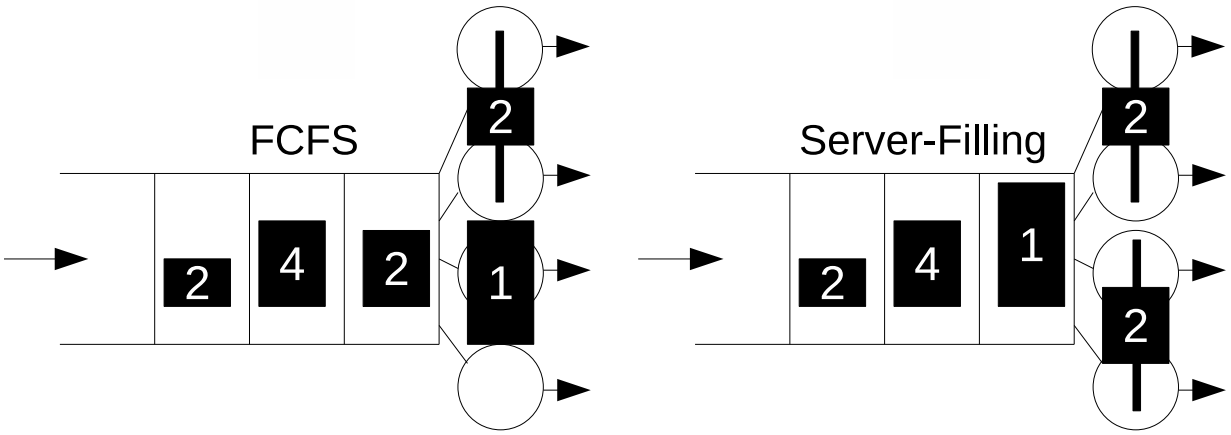


Figure 5.1: The FCFS and ServerFilling scheduling policies in the multiserver-job setting. Each job has two characteristics: A *server need*, the number of servers it requires in order to run, and a *duration*, the amount of time in service it requires. The height of each rectangle represents the job's *size*, the total amount of work necessary to complete the job, which is the product of the job's server need and its duration. FCFS serves jobs in arrival order. If the next job in the queue has larger server need than the number of available servers, that job, and all jobs behind it, must wait in the queue. The ServerFilling policy is defined in Section 4.4.4.

a job in service, placing it back in the queue, and returning to that job later. To measure a scheduling policy's performance, we are interested in studying the policy's *mean response time*, the mean time from when a job arrives to when it completes.

Optimizing MSJ mean response time In Chapter 4, we introduce and analyze the ServerFilling scheduling policy, depicted in Fig. 5.1. This analysis of ServerFilling is the first mean response time analysis of *any* MSJ scheduling policy. We now want to find a MSJ scheduling policy that we can prove achieves *optimal* mean response time.

Prioritizing small jobs To achieve optimal mean response time, we need to prioritize small jobs. In particular, we'd like to serve jobs in near-SRPT order. SRPT refers to the Shortest-Remaining-Processing-Time policy, which is known to achieve minimal mean response time in the single-server setting [194], and for which we proved an optimality result in the one-server-per-job multiserver setting in Chapter 2. Even properly defining small jobs in the MSJ setting is not immediately obvious: Short duration? Small server need? A combination of the two? As we show in this chapter, the right notion of "small jobs" are jobs with a small amount of overall inherent work, i.e. a small *size*, defined as the product of server need and service duration.

Filling all servers However, it is not sufficient to merely prioritize small jobs. We must also ensure that we fill all of the servers with jobs, rather than wasting servers. Policies which prioritize small jobs but waste servers achieve poor mean response times, as we show in Fig. 5.4. A useful tool for filling the servers is the ServerFilling policy from Chapter 4. ServerFilling always fills all k servers, whenever a sufficient number of jobs are present in the system. Thus, it is natural to combine the ServerFilling policy with the prioritization of small jobs, giving rise to the ServerFilling-SRPT scheduling policy, which is the primary focus of this chapter. We depict ServerFilling-SRPT in Fig. 5.2.

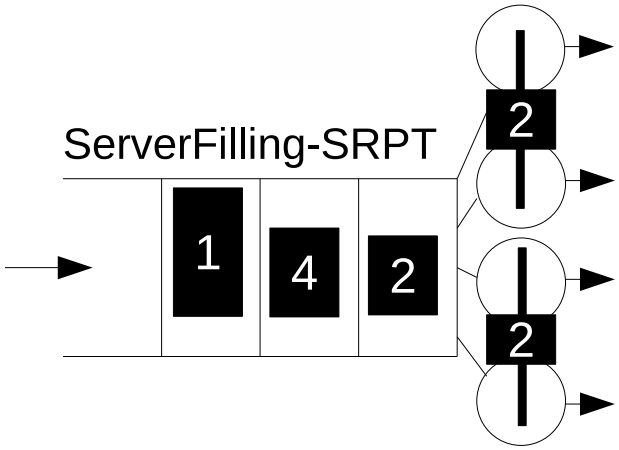


Figure 5.2: The ServerFilling-SRPT scheduling policy in the multiserver-job setting. It starts with the SRPT service order, then fills jobs into service in the same manner as ServerFilling. We define ServerFilling-SRPT in Section 5.4.2.

5.2 Technical Introduction

5.2.1 The multiserver-job model

Traditional multiserver queueing theory focuses on models, such as the M/G/k, where every job occupies exactly one server. For decades, these models remained popular because they captured the behavior of computing systems, while being amenable to theoretical analysis. However, such one-server-per-job models are no longer representative of many modern computing systems.

Consider today's large-scale computing centers, such as the those of Google, Amazon and Microsoft. While the *servers* in these data centers still resemble the *servers* in traditional models such as the M/G/k, the *jobs* have changed: Each job now requires many servers, which it holds simultaneously. While some jobs require few servers, other jobs require many more servers. For instance, in Fig. 5.3, we show the distribution of the number of CPUs requested by jobs in Google's recently published trace of its "Borg" computation cluster [86, 218]. The distribution is highly variable, with jobs requesting anywhere from 1 to 100,000 normalized CPUs. Throughout this chapter, we will focus on this "multiserver-job model" (MSJ), by which we refer to the common situation in modern systems where each job concurrently occupies a fixed number of servers (typically more than one), throughout its time in service.

The multiserver-job model is fundamentally different from the one-server-per-job model. In the one-server-per-job model, any work-conserving scheduling policy such as First-Come First-Served (FCFS) can achieve full server utilization. In the multiserver-job model, a naïve scheduling policy such as FCFS will waste more servers than necessary. As a result, server utilization and system stability are dependent on the scheduling policy in the multiserver-job model. While finding throughput-optimal scheduling policies is a challenge, several such policies are known, including MaxWeight [148], Randomized Timers [71, 181], and ServerFilling [86]. However, none of these policies give consideration to optimizing mean response time; each policy solely focuses on optimizing throughput. In fact, the empirical mean response time of

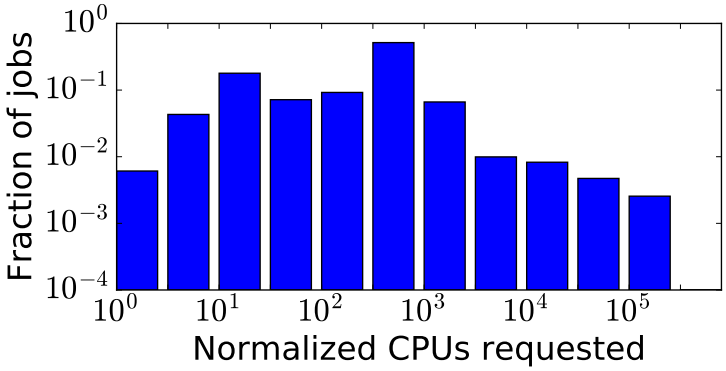


Figure 5.3: The distribution of number of CPUs requested by jobs in Google's recently published Borg trace [218]. Number of CPUs is normalized so that the smallest job in the trace uses one normalized CPU.

such policies can be very poor [71], motivating our goal of finding throughput-optimal policies which moreover minimize mean response time.

5.2.2 The challenges of minimizing MSJ mean response time

In the M/G/k setting, where each job requires a single server, it was recently proven that the SRPT-k (Shortest Remaining Processing Time-k) scheduling policy minimizes mean response time in the heavy-traffic limit [81]. SRPT-k is a very simple policy: serve the k jobs of least remaining duration (service time).

Unfortunately, in the multiserver-job system, trying to simply adapt the SRPT-k policy does not result in an optimal policy for two reasons:

- Prioritizing by remaining job duration is not the right way to minimize mean response time. We will show that an optimal policy must prioritize by remaining *size*, which we define to be proportional to the product of a job's duration and its *server need*, the number of servers the job requires. We define these terms in more detail in Section 5.4.
- Even with this concept of size, a prerequisite for minimizing mean response time in the heavy-traffic limit is throughput-optimality, which requires a policy to efficiently utilize all of the servers whenever possible. Unfortunately, greedily prioritizing the job of least remaining size, as in SRPT-k, is not throughput optimal. Our policy must be throughput-optimal, while *also* prioritizing small jobs.

We therefore ask:

> *What scheduling policy for the multiserver-job model should we use to* minimize mean response time *in the heavy-traffic limit?*

By "heavy-traffic" we mean as load $\rho \to 1$, while the number of servers, k, stays fixed. The precise definition of load ρ and the heavy-traffic limit will be explained in detail in Section 5.4.

5.2.3 ServerFilling-SRPT and ServerFilling-Gittins

To answer this question, we introduce the ServerFilling-SRPT scheduling policy, the first scheduling policy to minimize mean response time in the multiserver-job model in the heavy traffic limit.

ServerFilling-SRPT is defined in the setting where k is a power of 2, and all server needs are powers of 2. This setting is commonly seen in practice in supercomputing and other highly-parallel computing settings [38, 53].

To define ServerFilling-SRPT, imagine all jobs are ordered by their remaining size. Select the smallest initial subset M of this sequence such that the jobs in M collectively require at least k servers. Finally, place jobs from M into service in order of largest server need. This procedure is performed preemptively, whenever a job arrives or completes. As we show in Section 5.4.2, whenever jobs with total server need at least k are present in the system, this procedure will fill all k servers. We use this property to prove in Section 5.5 that ServerFilling-SRPT minimizes mean response time in the heavy-traffic limit.

ServerFilling-SRPT requires the scheduler to know job durations, and hence sizes, in advance. Sometimes the scheduler does not have duration information. In the M/G/1 setting, when job sizes are unknown, the Gittins policy [72] is known to achieve optimal mean response time. We therefore introduce the ServerFilling-Gittins policy in Section 5.6. We prove similar heavy-traffic optimality results for ServerFilling-Gittins.

5.2.4 A generalization: DivisorFilling-SRPT and DivisorFilling-Gittins

While ServerFilling-SRPT requires that the server needs are powers of 2, we have developed a more general scheduling policy which requires only that the server needs all divide k. We call this generalization DivisorFilling-SRPT. The DivisorFilling-SRPT policy is more complex than ServerFilling-SRPT, and hence we defer its discussion to Section 5.7. In Section 5.7, we define both DivisorFilling-SRPT and DivisorFilling-Gittins. We then show that all of our results about ServerFilling-SRPT and ServerFilling-Gittins hold for DivisorFilling-SRPT and DivisorFilling-Gittins.

5.2.5 A Novel Proof Technique: MIAOW

In recent years, there have been a plethora of proof techniques developed to handle the analysis of multiserver systems. These include:

- Multiserver tagged job analysis [81, 82, 204],
- Worst-case work gap [81, 82, 204],
- WINE (Work Integral Number Equality) [198, 202],
- Work Decomposition law [202].

Unfortunately, none of these techniques suffice to handle the analysis of ServerFilling-SRPT and DivisorFilling-SRPT. As we discuss in Section 5.5.2, the analysis of ServerFilling-SRPT requires bounding the *waste* relative to a resource-pooled single-server SRPT system, where waste is the expected product of work and unused system capacity. In order to analyze waste, we introduce a new technique called MIAOW, Multiplicative Interval Analysis of Waste. MIAOW subdivides

Table 5.1: Comparison of multiserver-job scheduling policies

	Maximize throughput		Minimize mean response time	
Policies	Attempted	Proven	Attempted	Proven
MaxWeight [148]	✓	✓		
Randomized Timers [71, 181]	✓	✓		
ServerFilling [86]	✓	✓		
FCFS [60, 120, 187]				
Simple backfilling heuristics: First-Fit, BestFit, etc. [120, 225]	✓			
Size-aided backfilling: EASY, conservative, dynamic, etc. [33, 120]	✓			
Size-based heuristics: GreedyS-RPT, FirstFitSRPT, etc. [33]			✓[1]	
Size & learning heuristics [90]	✓		✓	
ServerFilling-SRPT (Section 5.5) & **DivisorFilling-SRPT** (Sec. 5.7)	✓	✓	✓	✓

jobs into multiplicative intervals based on their remaining sizes, and bounds the waste in each interval.

5.2.6 Comparison with other policies

In Table 5.1, we compare our ServerFilling-SRPT and DivisorFilling-SRPT policies and our asymptotic optimality results with prior work in the multiserver-job setting. Prior work broadly falls into two categories: theoretical results focusing on throughput-optimality, and good heuristic policies. Our result is the first to theoretically study the problem of minimizing mean response time.

Fig. 5.4 compares the mean response time of ServerFilling-SRPT to that of prior throughput-optimal policies, as well as naïve size-based heuristic policies. These selected policies are representative of the empirical behavior of a wide variety of prior policies: Some of the policies shown have SRPT-like behavior, some policies are throughput-optimal, but only our ServerFilling-SRPT policy achieves both. Correspondingly, in this simulation and others we have performed, ServerFilling-SRPT has the best mean response time at all loads ρ, often by huge margins.

5.2.7 Summary of our contributions and outline

- In Section 5.4, we introduce the ServerFilling-SRPT scheduling policy.
- In Section 5.5, we bound the mean response time of ServerFilling-SRPT. We introduce MIAOW, a new technique for bounding the total "relevant" work in the system. Using that

[1] Because these heuristics are not throughput optimal, they are only competitive for mean response time at low to moderate load ρ.

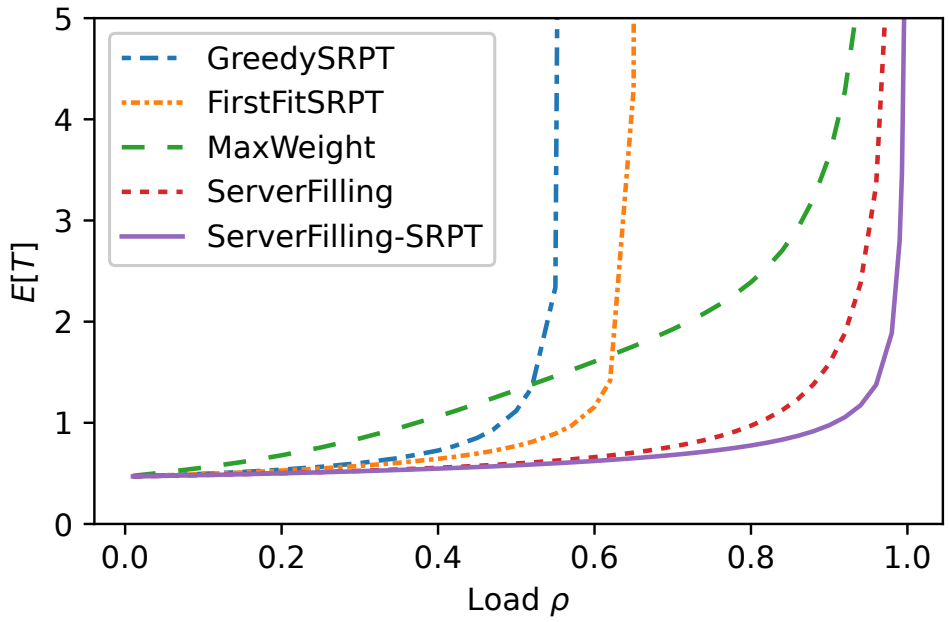


Figure 5.4: Simulated mean response time $\mathbb{E}[T]$ as a function of load ρ in a multiserver-job setting with $k = 8$ total servers. Server need is sampled uniformly from $\{1, 2, 4, 8\}$. Size is exponentially distributed, independent of the number of servers required. Policies defined in Section 5.8. Simulations use 10^7 arrivals. Loads $\rho \in [0, 0.999]$ simulated.

bound, we prove that ServerFilling-SRPT has asymptotically optimal mean response time as load $\rho \to 1$.

- In Section 5.6, we introduce the ServerFilling-Gittins scheduling policy, in the setting of unknown or partially-known job sizes and durations. We prove a similar bound and asymptotic optimality result for ServerFilling-Gittins.
- In Section 5.8, we empirically evaluate ServerFilling-SRPT using simulation, showing that it outperforms prior policies on realistic distributions over a variety of loads, not just the $\rho \to 1$ limit.

All of our results for ServerFilling-SRPT and ServerFilling-Gittins also extend to DivisorFilling-SRPT and DivisorFilling-Gittins.

5.3 Prior work

There are no prior optimality or asymptotic optimality results for mean response time in the multiserver-job system. The most similar system where such results have been proven is the M/G/k, a multiserver system with single-server jobs, and those results build off of classical results in the M/G/1.

5.3.1 Single-server-job models (one server per job)

In the single-server setting, the Shortest Remaining Processing Time policy (SRPT), which prioritizes the job of least remaining size, has been proven to minimize mean response time in the known-size M/G/1, as well as the worst-case single-server system [194, 196]. Note that in the single-server setting, a job's size is simply its duration. In the unknown- and partially-known-size settings, the Gittins policy is known to minimize mean response time in the M/G/1 [72, 199].

In the M/G/k, where jobs require a single server, [81] proves that the SRPT-k scheduling policy, the natural analogue of SRPT in the M/G/k, asymptotically minimizes mean response time in the known-size M/G/k in the heavy-traffic limit. There, as in this chapter, load ρ is defined as the long-term average fraction of busy servers; $\rho \to 1$ is the heavy-traffic limit. Specifically, the paper shows that

$$\lim_{\rho \to 1} \frac{\mathbb{E}[T^{SRPT\text{-}k}]}{\mathbb{E}[T^{OPT\text{-}k}]} = 1.$$

This is proven despite the fact that the optimal policy OPT-k is unknown.

In the unknown job size setting, similar asymptotic optimality results for mean response time have been proven for the Gittins-k policy [202] and a monotonic variant thereof [204]. Moreover, for the Gittins-k policy, these results generalize to the partially-known job size setting, such as a setting with imperfect job size estimates.

5.3.2 Multiserver-job model (many servers per job)

Theoretical results in the multiserver-job model are limited. The *blocking* model, where arriving jobs either immediately receive service or are dropped, has received significant attention, with many strong results [11, 217, 220, 229] such as the exact steady state distribution. However, without any queue these models don't fit most real computing systems well. In the *queueing* MSJ model, which we focus on, results are much more limited [106]. The best-studied scheduling policy is the first-come first-served (FCFS) policy. Stability region results for FCFS are known in several limited settings [187, 188], and steady state results are only known in the case of two servers [31, 63, 125].

Recently, the Work Conserving Finite Skip (WCFS) framework has been used to analytically characterize response time under the ServerFilling and DivisorFilling scheduling policies [86], both of which serve jobs in near-FCFS order. We modify the ServerFilling and DivisorFilling policies to prioritize jobs of shortest remaining size (SRPT). We then use a novel proof technique called MIAOW to demonstrate that ServerFilling-SRPT and DivisorFilling-SRPT achieve asymptotically similar mean response time to SRPT in an analogous M/G/1 setting.

There has also been work in the *scaling* multiserver-job model, where one analyzes a sequence of multiserver-job systems with jointly increasing arrival rate, number of servers, and server needs [113, 226]. The regimes investigated include multiserver-job analogues of the Halfin-Whitt regime. Our results complement these, as we study a system with a fixed number of servers k in the heavy-traffic limit.

5.3.3 Supercomputing

Supercomputing centers are one of the originators of the multiserver-job model: Supercomputing jobs closely resemble the jobs in the multiserver-job model. Jobs commonly demand anywhere from one core to thousands of concurrent cores [119, 224]. Unfortunately, all of the papers in this area focus on simulation or empirical results, rather than analytical results [10, 58, 60, 61, 120, 214, 215]. These papers study a variety of scheduling policies, such as FCFS, various backfilling policies, and other more novel policies. Backfilling policies considered include simpler, no-duration-information policies such as FirstFit and BestFit [120, 225], as well as more complex, duration-information-based policies such as EASY backfilling [210], conservative backfilling [210], Smallest Area[2] First-backfilling [33], Dynamic Backfilling [120], and many more.

Often the primary goal of these papers is achieving high utilization, with secondary goals including minimizing mean response time and ensuring fairness between different types of jobs. However, their settings are sometimes more restrictive than our setting: preemption may be either limited or impossible. When preemption is impossible, maximum utilization is lower, often around $\rho = 70\%$, and mean response times are often high near the utilization threshold.

Our scheduling policies, such as ServerFilling-SRPT, can only be defined for the subset of settings where preemption is possible, and our policies leverage preemption to achieve much stronger results in those settings.

5.3.4 Virtual Machine Scheduling

In the field of cloud computing, the Virtual Machine (VM) scheduling problem is essentially a multi-resource generalization of the multiserver-job model. In this model, rather than a single requirement like server need, each job requires concurrent utilization of several different limited resources, such as RAM, CPU, GPU, network bandwidth, etc. Of course, any results in this more general setting also apply to the multiserver-job setting. In the VM scheduling literature, papers typically focus on finding a throughput-optimal policy. Two major categories of such policies are the preemptive MaxWeight [148] and non-preemptive Randomized Timers [71, 181] scheduling frameworks.

These papers focus entirely on achieving throughput optimality, and the mean response time of the resulting policies can be poor, as several of the above papers note. Work on optimal mean response time in the VM scheduling literature has been limited to heuristic policies and empirical evaluation [90].

5.4 Setting

5.4.1 Multiserver-job Model

The multiserver-job (MSJ) model is a multiserver queueing model where each job requires a fixed number of servers concurrently over its entire time in service. The jobs are therefore called "multiserver jobs."

[2]The term "area" used in [33] is equivalent to our "size".

A job j has two requirements: A server need k_j and a service duration d_j. These requirements are sampled i.i.d. from some joint distribution with random variables (K, D). Note that K and D can be correlated. A job's server need k_j is at most the total number of servers, k. The total server need of the jobs in service at any time must sum to at most k. The job j will complete after d_j time in service.

We assume Poisson arrivals with rate λ, and we assume preemption is allowed with no loss of progress.

Let a job j's *size* s_j be defined as $k_j d_j/k$, and likewise define the job size distribution $S = KD/k$. Job j's size can be viewed as the area of a rectangle with height equal to the job's duration d_j and width equal to k_j/k, the fraction of the total service capacity occupied by job j. Likewise, a job's *remaining size* r_j is its remaining duration multiplied by k_j/k. We define a job j's *service rate* to be k_j/k, the rate at which the job's remaining size decreases during service. We define a job's age a_j to be $s_j - r_j$, which increases at rate k_j/k whenever the job is in service.

A *resource-pooled* M/G/1 is defined to be a system with a single server with the same capacity as all k original servers pooled together, and the same arrival rate λ and job size distribution S as the original MSJ system. We allow the resource-pooled M/G/1 to divide its capacity arbitrarily among the jobs in the system. In particular, while jobs in the MSJ system have fixed service rates depending on their server needs, in the resource-pooled system any combination of service rates is allowed, decreasing remaining sizes accordingly. Note that the resource-pooled system is strictly more flexible than the MSJ system, so the optimal policy in the resource-pooled system is superior to the optimal policy in the MSJ system.

Let $W(t)$ be the total work in the system at time t: The sum of the remaining sizes r_j of all job's in the system at time t. Let $B(t)$ be the *"busyness"* of the system at time t: The fraction of servers that are occupied at time t. Note that $B(t)$ is also the total service rate of all jobs in service at time t, and so $B(t) = -\frac{d}{dt}W(t)$, outside of arrival moments. We also define W and B to be the corresponding stationary random variables.

Let *load* $\rho = \lambda\mathbb{E}[S]$ be the long-run average rate at which work arrives to the system. We assume $\rho < 1$ as a necessary condition for stability. We will focus on settings where $\rho < 1$ is also sufficient for stability for some feasible scheduling policy. Note that ρ is a constant and that $\rho = \mathbb{E}[B]$, under any scheduling policy for which the system is stable.

Next, let us define an *r-relevant* job, where r is a remaining size threshold. A job j is r-relevant if $r_j \leq r$. This terminology is in reference to the tagged job analysis used in studying SRPT in the M/G/1 and M/G/k settings [81, 196]; in those settings, the service of a job with remaining size r is only affected by the presence of r-relevant jobs in the system. The multiserver-job system is not as simple, so we do not employ a tagged-job approach, but we reuse the terminology.

Correspondingly, let the r-relevant work $W_r(t)$ be the total remaining size of all r-relevant jobs in the system at time t, and let $B_r(t)$ be the fraction of servers which are serving r-relevant jobs at time t. Define B_r and W_r correspondingly. The core of our proof lies in bounding expectations of random variables involving B_r and W_r, and combining these with a characterization of mean response time $\mathbb{E}[T]$ in terms of B_r and W_r.

Next, let us define the r-relevant load ρ_r to be the long-run average r-relevant busyness of the system. A job with size s_j receives $\min(s_j, r)$ service while having remaining size $\leq r$. As a result, $\rho_r = \lambda\mathbb{E}[\min(S, r)] = \mathbb{E}[B_r]$. We further divide the r-relevant load based on whether

the job in question has initial size $\leq r$. Let the *arrival load* $\rho_r^A = \lambda \mathbb{E}[S \mathbb{1}\{S < r\}]$, and let the *recycled load* $\rho_r^R = \lambda r \mathbb{P}\{S > r\}$. Note that $\rho_r = \rho_r^A + \rho_r^R$. Note also that ρ_r, ρ_r^A, and ρ_r^R are all not dependent on the policy π.

Finally, let us define an *r-recycling moment* to be a moment when a job j with initial size $s_j > r$ reaches remaining size $r_j = r$. Let $\mathbb{E}_r[\cdot]$ be an expectation taken over r-recycling moments, just prior to the job recycling.

5.4.2 ServerFilling-SRPT

This chapter considers two settings of server needs:

- The "power of two" setting: k is power of two, and all server needs k_j are powers of two.
- The "divisible" setting: k is general, and all server needs k_j are divisors of k.

Corresponding to these two settings, we have two policies of interest: ServerFilling-SRPT for the power of two setting, and DivisorFilling-SRPT for the divisible setting. We define ServerFilling-SRPT here, and DivisorFilling-SRPT in Section 5.7. When writing equations throughout this chapter, we abbreviate ServerFilling-SRPT as SFS-k.

To implement SFS-k, start by ordering jobs in increasing order of remaining size r_j, breaking ties arbitrarily. Define $j_1, j_2, \ldots$ such that

$$r_{j_1} \leq r_{j_2} \leq \ldots.$$

Next, consider initial subsets of this ordering:

$$\{j_1\}, \{j_1, j_2\}, \{j_1, j_2, j_3\} \ldots.$$

We are interested in the smallest initial subset M in which the total server need is at least k. In other words, let i^* be the smallest index such that

$$\sum_{i=1}^{i^*} k_{j_i} \geq k.$$

If there is no such index, then ServerFilling-SRPT serves all jobs in the system simultaneously.

Otherwise, ServerFilling-SRPT will serve a subset of $M = \{j_1, j_2, \ldots, j_{i^*}\}$. Among this subset, ServerFilling-SRPT prioritizes jobs of largest server need, placing jobs into service in descending order of server need, until no servers remain or the next job cannot fit, breaking ties by smallest remaining size, and further ties arbitrarily.

In the power-of-two setting, ServerFilling-SRPT guarantees the following strong property: At all times, either all servers are busy, or all jobs are in service. This was proven for the ServerFilling policy [86, Lemma 1], which is identical to ServerFilling-SRPT, except that jobs are ordered in arrival order, rather than SRPT order. For completeness, we reprove this result here:

Lemma 5.4.1. *Under the ServerFilling-SRPT policy, in the power-of-two setting, if the total server need of jobs in the system is at least k servers, all k servers are busy.*

Proof. Recall that M is a set of jobs, each with server need a power of two, which have a total server need of at least k. Label the jobs $m_1, m_2, \ldots$ in decreasing order of server need, tiebroken by least remaining size.

$$k_{m_1} \geq k_{m_2} \geq \ldots$$

Let $\text{NEED}(z)$ represent the total server need of the first z jobs in this ordering:

$$\text{NEED}(z) = \sum_{i=1}^{z} k_{m_i}.$$

The set of jobs served by ServerFilling-SRPT is an initial sequence of this server need ordering: $\{m_i \mid i \leq \ell\}$ for some ℓ. Specifically, the index ℓ up to which ServerFilling-SRPT serves jobs is the largest index z such that $\text{NEED}(z) \leq k$. To prove Lemma 5.4.1, it suffices to show that $\text{NEED}(\ell) = k$.

Note that $\text{NEED}(0) = 0$ and $\text{NEED}(|M|) \geq k$. As a result, $\text{NEED}(z)$ must cross k at some point. To prove that $\text{NEED}(\ell) = k$, it suffices to prove that:

$$\text{There exists no index } \ell' \text{ such that } \text{NEED}(\ell') < k \text{ and } \text{NEED}(\ell'+1) > k. \tag{5.1}$$

To prove (5.1), let us define $\text{REMAIN}(z)$, the number of servers remaining after z jobs have been placed into service:

$$\text{REMAIN}(z) = k - \text{NEED}(z).$$

Because all server needs k_j are powers of two, we will show that $\text{REMAIN}(z)$ carries an important property:

$$\text{REMAIN}(z) \text{ is divisible by } k_{m_{z+1}} \text{ for all } z. \tag{5.2}$$

We will use (5.2) to prove (5.1). We write $a|b$ to indicate that a divides b.

We will prove (5.2) by induction on z. For $z = 0$, $\text{REMAIN}(0) = k$. Because k is a power of two, and k_{m_1} is a power of two no greater than k, the base case holds. Next, assume that (5.2) holds for some index z, meaning that $k_{m_{z+1}} | \text{REMAIN}(z)$. Note that $\text{REMAIN}(z+1) = \text{REMAIN}(z) - k_{m_{z+1}}$. As a result, $k_{m_{z+1}} | \text{REMAIN}(z+1)$. Now, note that $k_{m_{z+2}} | k_{m_{z+1}}$, because both are powers of two, and $k_{m_{z+2}} \leq k_{m_{z+1}}$. As a result, $k_{m_{z+2}} | \text{REMAIN}(z+1)$, completing the proof of (5.2).

Now, we are ready to prove (5.1). Assume for contradiction that such an ℓ' exists. Then $\text{REMAIN}(\ell') > 0$, and $\text{REMAIN}(\ell'+1) < 0$. Because $\text{REMAIN}(\ell'+1) = \text{REMAIN}(\ell') - k_{m_{\ell'+1}}$, we therefore know that $k_{m_{\ell'+1}} > \text{REMAIN}(\ell')$. But from (5.2), we know that $k_{m_{\ell'+1}}$ divides $\text{REMAIN}(\ell')$, which is a contradiction. □

Note that Lemma 5.4.1 remains true if the power-of-two setting is replaced by the power-of-x setting, for any integer x. In fact, the only condition on the server needs necessary to prove Lemma 5.4.1 is that all server needs divide k, and all server needs divide all larger server needs.

An important corollary of Lemma 5.4.1 is a property which we call "relevant work efficiency":

Corollary 5.4.1 (Relevant work efficiency)**.** *Under the ServerFilling-SRPT policy, in the power-of-two setting, if there are k or more r-relevant jobs in the system, all servers are occupied by r-relevant jobs, meaning that $B_r = 1$.*

Proof. Note that $|M| \leq k$, because M is the smallest initial subset of the SRPT ordering with total server need at least k, and all jobs have server need at least 1. Therefore, if there are k or more r-relevant jobs in the system, then all jobs in M are r-relevant, so ServerFilling-SRPT fills all k servers with r-relevant jobs, meaning that $B_r = 1$. □

Corollary 5.4.1 is the sole property of ServerFilling-SRPT that we will use to prove our main theorems, Theorems 5.5.1 and 5.5.2.

DivisorFilling-SRPT in the divisible setting also satisfies the relevant work efficiency property: If there are k or more r-relevant jobs in the system, then $B_r = 1$, as we discuss in Section 5.7. As a result, our main theorems, Theorems 5.5.1 and 5.5.2, also hold for DivisorFilling-SRPT.

5.5 ServerFilling-SRPT: Asymptotically Optimal Mean Response Time

5.5.1 Summary of Results and Proofs

To prove the optimality of ServerFilling-SRPT, we will compare ServerFilling-SRPT's mean response time against a resource-pooled M/G/1/SRPT system with the same size distribution S. Let "SRPT-1" denote the M/G/1/SRPT system. Recall that SRPT-1 combines the power of all k servers into a single server, which can work on any job or any mixture of jobs. This resource-pooled system is strictly more flexible than the multiserver-job system, so the optimal policy in the resource-pooled system forms a lower bound on the optimal policy in the MSJ system. Because SRPT minimizes mean response time in the M/G/1, SRPT-1 yields a lower bound on the optimal mean response time in the MSJ system.

We will upper bound the gap in mean response time between ServerFilling-SRPT and SRPT-1 for all loads ρ, and prove that the gap asymptotically grows slower than $\mathbb{E}[T^{SRPT\text{-}1}]$. By doing so, we will show that ServerFilling-SRPT is asymptotically optimal in the multiserver-job system.

First, we prove a bound on the gap in mean response time between ServerFilling-SRPT and SRPT-1:

Theorem 5.5.1. *For all loads ρ, in the power-of-two setting, the mean response time gap between ServerFilling-SRPT and SRPT-1 is at most*

$$\mathbb{E}[T^{SFS\text{-}k}] - \mathbb{E}[T^{SRPT\text{-}1}] \leq \frac{(e+1)(k-1)}{\lambda} \ln \frac{1}{1-\rho} + \frac{e}{\lambda}.$$

The same is true of DivisorFilling-SRPT in the divisible setting.

Proof deferred to Section 5.5.3. □

We use this bound to prove that ServerFilling-SRPT yields optimal mean response time in the heavy-traffic limit:

Theorem 5.5.2. *If $\mathbb{E}[S^2(\log S)^+] < \infty$, then ServerFilling-SRPT is asymptotically optimal in the multiserver-job system:*

$$\lim_{\rho\to 1} \frac{\mathbb{E}[T^{SFS\text{-}k}]}{\mathbb{E}[T^{SRPT\text{-}1}]} = \lim_{\rho\to 1} \frac{\mathbb{E}[T^{SFS\text{-}k}]}{\mathbb{E}[T^{OPT\text{-}k}]} = 1.$$

The same is true of DivisorFilling-SRPT in the divisible setting.

Proof deferred to Section 5.5.3. □

The condition $\mathbb{E}[S^2(\log S)^+] < \infty$ is very slightly stronger than finite variance.

In Section 5.6, we generalize both results to the settings of unknown- and partially-known job duration.

5.5.2 A Novel Proof Technique: MIAOW

Challenges of multiserver-job analysis

As mentioned in Section 5.2, mean response time analysis in the multiserver-job system is a difficult problem, with no size- or age-based scheduling policies having previously been analyzed. The difficulty arises from two sources: First, analyzing the mean response time of any system with multiple servers under a size- or age-based scheduling policy is already very difficult, even in a single-server-job setting such as the M/G/k. New techniques based on relevant work have recently been developed to handle this challenge. The first such analysis is as recent as 2018, when the SRPT-k policy was analyzed in the M/G/k [81], followed by the analysis of the monotonic-Gittins-k and Gittins-k policies in the M/G/k in 2020 and 2021 [202, 204].

Unfortunately, the multiserver-job system presents a major additional challenge. We will show in Section 5.5.2 that these recent techniques for multiserver systems break when dealing with our multiserver-job system. As a result, we need a new technique to analyze the multiserver-job systems, which we introduce in Section 5.5.2.

Key idea of previous approaches: Relevant work similarity

The first step in applying relevant-work-based techniques [81, 202, 204] is to prove a property which we call "relevant work similarity":

Definition 5.5.1. *A policy π achieves* relevant work similarity (RWS) *if, for all remaining sizes r (or ranks[3] r), the policy π system and the optimal resource-pooled system OPT-1 (e.g. SRPT-1 or Gittins-1) have similar expected r-relevant work:*

$$\mathbb{E}[W_r^{\pi}] - \mathbb{E}[W_r^{OPT\text{-}1}] \leq O(r).$$

The RWS property holds for all three policies and systems analyzed previously [81, 202, 204], as well as for ServerFilling-SRPT. Unfortunately, the RWS property is not sufficient on its own to tightly bound mean response time, or to prove asymptotically optimal mean response time.

[3] Rank is the analogue of remaining size under the Gittins policy.

First attempt: Tagged job approach

One way to build on the RWS property to prove asymptotic optimality is to use the tagged job approach, employed by the SRPT-k [81] and monotonic-Gittins-k [204] results. The tagged job approach combines the RWS property with an additional property, which we call "relevant work implies response time":

Definition 5.5.2. *A policy π achieves* relevant work implies response time (RW$\rightarrow$RT) *if the following holds: If a generic tagged job of size r sees some amount x of r-relevant work in each of the policy π system and the optimal resource-pooled system OPT-1, then its expected response must be similar (within $O(r)$) in the two systems.*

If the RWS and RW$\rightarrow$RT properties can both be proven for some policy π, it is relatively straightforward to tightly bound mean response time and prove that the policy π has asymptotically optimal mean response time. Unfortunately, for our ServerFilling-SRPT policy, the RW$\rightarrow$RT property fails, meaning that the tagged-job approach cannot be used.

For a counterexample to the RW$\rightarrow$RT property for ServerFilling-SRPT, consider a scenario where the tagged job requires 1 server and has the smallest size of any job in the system, and where it sees many jobs on arrival, all of which require an even number of servers and have larger remaining sizes. Furthermore, assume that arriving jobs rarely require 1 server. The resource-pooled SRPT-1 system will quickly complete the tagged job, as it has the smallest remaining size of any job in the system.

In contrast, the ServerFilling-SRPT system will not quickly complete the tagged job, because ServerFilling-SRPT prioritizes the jobs of largest server need among the initial subset M, as defined in Section 5.4.2. The tagged job will need to wait until the system empties or additional 1-server jobs arrive to be served. Clearly, similar relevant work does not imply similar response time.

This is an inherent difficulty of the multiserver-job system: Serving the tagged job any earlier would require leaving at least one server empty, as the tagged job is the only job with an odd server need, given the power-of-two setting. This could endanger throughput-optimality. As a result, the tagged-job approach cannot be used to effectively analyze the multiserver-job system.

Second Attempt: Gittins-k

The analysis of the Gittins-k policy for the M/G/k [202] also relies on the RWS property, which again is insufficient alone to prove asymptotically optimal mean response time in their setting. As in our setting, for the Gittins-k system, the RW$\rightarrow$RT property fails, so the tagged-job approach cannot be employed.

The authors take a different approach: They introduce WINE [202, Theorem 6.3], our Lemma 5.5.1, a new identity that relates response time and relevant work in all systems.[4] WINE implies

$$\mathbb{E}[T^{\pi\text{-}k}] - \mathbb{E}[T^{OPT\text{-}1}] = \frac{1}{\lambda}\int_0^\infty \frac{\mathbb{E}[W_r^{\pi\text{-}k}] - \mathbb{E}[W_r^{OPT\text{-}1}]}{r^2}. \tag{5.3}$$

[4]The name "WINE", short for "work integral number equality" [198], is more recent than [202], but refers to their Theorem 6.3.

WINE is more general than the RW→RT property, because RW→RT only holds in certain systems.

We can see from (5.3) that the RWS property is almost enough to bound mean response time, but the $O(r)$ bound is too loose to show that the integral converges. The authors therefore prove a stronger version of the RWS property at sufficiently low and high ranks r. Combining their strengthened bounds with WINE, they prove that Gittins-k achieves asymptotically optimal mean response time in the M/G/k.

However, their proof of a stronger version of RWS at low ranks r relies on the fact that under Gittins-k in the M/G/k, the job of least rank is guaranteed to be served. This fails when applied to ServerFilling-SRPT, because in our multiserver-job system the job of least rank is not guaranteed to receive service. See the counterexample given in Section 5.5.2.

Our approach

Our key idea is to directly focus on the integrated relevant work difference given in (5.3). This circumvents the need to strengthen the RWS property (like in Section 5.5.2) or prove an RW→RT property (like in Section 5.5.2).

We start with a key property of the ServerFilling-SRPT system, which we call "relevant work efficiency" (RWE). RWE states that if there are k or more r-relevant jobs in the system, then all servers are occupied by r-relevant jobs. We prove in Section 5.4.2, specifically in Corollary 5.4.1, that ServerFilling-SRPT satisfies the RWE property.

While one can show that RWE implies RWS, RWS alone is not enough, as discussed in Section 5.5.2. Instead, we use the RWE property to directly bound the integrated relevant work difference given in (5.3), thereby directly bounding the mean response time difference. We prove this result in Theorem 5.5.3. This forms the core of our proof that ServerFilling-SRPT achieves asymptotically optimal mean response time.

Theorem 5.5.3 is our key technical theorem; it provides a novel bound on the *waste* in any system which satisfies RWE. By waste, we refer to the quantity $\mathbb{E}[W_r(1 - B_r)]$, the expected product of r-relevant work and the fraction of system capacity not working on r-relevant jobs. Note that the SRPT-1 system never has any waste: If any r-relevant job is present the entire system capacity is working on such a job.

To bound waste, we use a novel technique which we call MIAOW: Multiplicative Interval Analysis of Waste. Intuitively, MIAOW makes use of the fact that both W_r and B_r change slowly as a function of r. We use this fact to bound the integrated waste over a generic interval of remaining sizes $[r_\ell, r_h]$. This contrasts with the prior waste-based technique [202], which focused on bounding waste at individual remaining sizes r, an approach which does not imply a useful bound in the MSJ setting. We then carefully select a sequence of remaining size intervals with multiplicatively diminishing spare capacity $1 - \rho_r^A$. Applying our bound to each interval completes Theorem 5.5.3.

We note that MIAOW is stronger than the techniques used to prove asymptotically optimality in the M/G/k for SRPT-k and Gittins-k [81, 202]. In particular, one could use our technique to reprove all of the asymptotic optimality results in those papers. This follows from the fact that the multiserver-job model is a generalization of the M/G/k: A multiserver-job setting where all server needs are 1 is simply an M/G/k.

5.5.3 Proof of Main Results

Our goal is to bound the mean response time of the ServerFilling-SRPT policy, relative to the resource-pooled SRPT-1 policy.

To bound mean response time, we start by applying the "work integral number equality" (WINE) technique [198, 202] to write mean response time $\mathbb{E}[T^\pi]$ for a general policy π in terms of expected relevant work $\mathbb{E}[W_r^\pi]$. This technique was introduced in [202, Theorem 6.3], but we reprove it here for completeness.

Lemma 5.5.1 (WINE Identity [202])**.** *For an arbitrary scheduling policy π, in an arbitrary system,*

$$\mathbb{E}[T^\pi] = \frac{1}{\lambda}\mathbb{E}[N^\pi] = \frac{1}{\lambda}\int_{r=0}^{\infty} \frac{\mathbb{E}[W_r^\pi]}{r^2} dr.$$

Proof. We will prove that at every moment in time,

$$N^\pi(t) = \int_{r=0}^{\infty} \frac{W_r^\pi(t)}{r^2} dr. \tag{5.4}$$

Recall that r-relevant work $W_r^\pi(t)$ is simply a sum over the r-relevant jobs in the system. As a result, we can consider the integral in (5.4) as a sum over the jobs in the system.

Consider a general job j, with remaining size r_j. The contribution of j to the r-relevant work $W_r^\pi(t)$ is r_j, for thresholds r such that $r_j \le r$, and 0 otherwise.

Therefore, the contribution of job j to the integral in (5.4) is

$$\int_{r=0}^{\infty} \frac{r_j \mathbb{1}\{r_j \le r\}}{r^2} dr = \int_{r=r_j}^{\infty} \frac{r_j}{r^2} dr = r_j \int_{r=r_j}^{\infty} \frac{1}{r^2} dr = r_j \frac{1}{r_j} = 1.$$

Because the contribution of an arbitrary job is 1, the integral in (5.4) simply counts the number of jobs in the system at time t, giving $N^\pi(t)$ as desired.

Note that $\mathbb{E}[T^\pi] = \frac{1}{\lambda}\mathbb{E}[N^\pi]$, by Little's Law [104]. □

Now that we have written mean response time in terms of relevant work, we need to understand $\mathbb{E}[W_r^\pi] - \mathbb{E}[W_r^{SRPT\text{-}1}]$, the difference in r-relevant work between a general policy π and the resource pooled SRPT-1 system. To do so, we employ the work-decomposition law. This technique was introduced in [202], and we specialize it here to the SRPT setting.

Lemma 5.5.2. *[202, Theorem 7.2] For an arbitrary scheduling policy π, in an arbitrary known-size system,*

$$\mathbb{E}[W_r^\pi] - \mathbb{E}[W_r^{SRPT\text{-}1}] = \frac{\mathbb{E}[(1 - B_r^\pi)W_r^\pi] + \rho_r^R \mathbb{E}_r[W_r^\pi]}{1 - \rho_r^A}.$$

Proof. We will employ the rate conservation law, applied to the random variable $(W_r^\pi)^2$, the square of the stationary distribution of r-relevant work in the system. The rate conservation law states that, because $(W_r^\pi)^2$ is a stationary random variable, its expected rate of increase and decrease must be equal. This argument can be formalized further using Palm Calculus.

To find these rates of increase and decrease, let us first examine W_r^π. W_r^π decreases continuously as work completes, and increases by jumps whenever jobs arrive. W_r^π decreases at rate B_r^π, the fraction of servers that are occupied by r-relevant jobs. When a job arrives with size S, it contributes $[S\mathbb{1}\{S \le r\}]$ relevant work, increasing W_r^π by that amount. Such arrivals occur at rate λ. Finally, whenever a job recycles, by being served until its remaining size falls to r, it adds r relevant work to W_r^π.

Using these rates, we can calculate the expected rates of increase and decrease of $(W_r^\pi)^2$.

$$\begin{aligned}
\text{Increase due to arrivals:} &\quad \lambda\mathbb{E}[(S\mathbb{1}\{S \le r\})^2] + 2\rho_r^A\mathbb{E}[W_r^\pi] \\
\text{Increase due to recycling:} &\quad \lambda_r^R r^2 + 2\rho_r^R\mathbb{E}_r[W_r^\pi] \\
\text{Decrease due to service:} &\quad 2\mathbb{E}[B_r^\pi W_r^\pi]
\end{aligned}$$

Equating these rates, we find that

$$\begin{aligned}
2\mathbb{E}[B_r^\pi W_r^\pi] &= \lambda\mathbb{E}[(S\mathbb{1}\{S \le r\})^2] + 2\rho_r^A\mathbb{E}[W_r^\pi] + \lambda_r^R r^2 + 2\rho_r^R\mathbb{E}_r[W_r^\pi]. \\
\mathbb{E}[B_r^\pi W_r^\pi] &= \frac{\lambda}{2}\mathbb{E}[(S\mathbb{1}\{S \le r\})^2] + \rho_r^A\mathbb{E}[W_r^\pi] + \frac{\lambda_r^R}{2} r^2 + \rho_r^R\mathbb{E}_r[W_r^\pi]. \\
\mathbb{E}[W_r^\pi] &= \mathbb{E}[(1-B_r^\pi)W_r^\pi] + \frac{\lambda}{2}\mathbb{E}[(S\mathbb{1}\{S \le r\})^2] + \rho_r^A\mathbb{E}[W_r^\pi] + \frac{\lambda_r^R}{2} r^2 + \rho_r^R\mathbb{E}_r[W_r^\pi]. \\
\mathbb{E}[W_r^\pi](1-\rho_r^A) &= \mathbb{E}[(1-B_r^\pi)W_r^\pi] + \frac{\lambda}{2}\mathbb{E}[(S\mathbb{1}\{S \le r\})^2] + \frac{\lambda_r^R}{2} r^2 + \rho_r^R\mathbb{E}_r[W_r^\pi]. \\
\mathbb{E}[W_r^\pi](1-\rho_r^A) &= \mathbb{E}[(1-B_r^\pi)W_r^\pi] + \rho_r^R\mathbb{E}_r[W_r^\pi] + \frac{\lambda}{2}\mathbb{E}[(S\mathbb{1}\{S \le r\})^2] + \frac{\lambda_r^R}{2} r^2. \qquad (5.5)
\end{aligned}$$

Let us evaluate (5.5) in the case where the policy π is SRPT-1. The first two terms of the right-hand side are nonnegative terms depending on the policy π, while the second two terms are the same for all policies.

Let us start with the first term on the right-hand side, $\mathbb{E}[(1-B_r^\pi)W_r^\pi]$. Note that under SRPT-1, if W_r^π is nonzero, i.e. if a r-relevant job is present, then SRPT-1 will serve a r-relevant job on its single server, and so $B_r^{SRPT\text{-}1} = 1$. As a result, either $W_r^{SRPT\text{-}1}$ or $1 - B_r^{SRPT\text{-}1}$ must always be zero, so this term is equal to 0.

Next, consider the second term, $\rho_r^R\mathbb{E}_r[W_r^\pi]$. Recall that $\mathbb{E}_r[\cdot]$ is an expectation over system states at times when r-relevant jobs recycle. In the SRPT-1 system, if a job is recycling by falling down to remaining size r, there must be no jobs in the system with remaining size less than r. As a result, $\mathbb{E}_r[W_r^\pi] = 0$.

We therefore conclude that

$$\mathbb{E}[W_r^{SRPT\text{-}1}](1-\rho_r^A) = \frac{\lambda}{2}\mathbb{E}[(S\mathbb{1}\{S \le r\})^2] + \frac{\lambda_r^R}{2} r^2. \qquad (5.6)$$

As an aside, note that this argument shows that SRPT-1 has the least value of $\mathbb{E}[W_r^\pi]$ for any policy π. This fact, combined with Lemma 5.5.1, provides an alternative proof that SRPT-1 is the optimal scheduling policy in the M/G/1.

Subtracting (5.6) from (5.5), we find that

$$\mathbb{E}[W_r^\pi](1-\rho_r^A) - \mathbb{E}[W_r^{SRPT\text{-}1}](1-\rho_r^A) = \mathbb{E}[(1-B_r^\pi)W_r^\pi] + \rho_r^R \mathbb{E}_r[W_r^\pi]$$
$$\mathbb{E}[W_r^\pi] - \mathbb{E}[W_r^{SRPT\text{-}1}] = \frac{\mathbb{E}[(1-B_r^\pi)W_r^\pi] + \rho_r^R \mathbb{E}_r[W_r^\pi]}{1-\rho_r^A}.$$

□

Combining Lemma 5.5.1, and specifically its implication (5.3), with Lemma 5.5.2, we arrive at the following characterization of the mean response time difference between a general policy π and SRPT-1:

Lemma 5.5.3. *For any scheduling policy π, in any system,*

$$\mathbb{E}[T^\pi] - \mathbb{E}[T^{SRPT\text{-}1}] = \frac{1}{\lambda}\int_0^\infty \frac{\mathbb{E}[(1-B_r^\pi)W_r^\pi]}{r^2(1-\rho_r^A)}dr \tag{5.7}$$
$$+ \frac{1}{\lambda}\int_0^\infty \frac{\rho_r^R \mathbb{E}_r[W_r^\pi]}{r^2(1-\rho_r^A)}dr. \tag{5.8}$$

Intuitively, (5.7) and (5.8) measure the inefficiency of the policy π relative to the ideal SRPT-1 system, through the lens of W_r^π, the r-relevant work under policy π.

The first term (5.7) measures the extent to which r-relevant work is present, but not being worked on. In the multiserver-job system, not all of the system can be devoted to a single job, so the waste $\mathbb{E}[(1-B_r^\pi)W_r^\pi]$ will typically be nonzero.

The second term (5.8) measures the extent to which jobs r-recycle while r-relevant work is present in the system. In the multiserver-job system, not all of the system can be devoted to a single job, so jobs with remaining size above r will be worked on, and will r-recycle, while r-relevant work is present, so $\mathbb{E}_r[W_r^\pi]$ will also typically be nonzero.

Our goal is to bound the magnitude of (5.7) and (5.8) under the ServerFilling-SRPT policy, in the power-of-two setting. We do so by making use of the key property of ServerFilling-SRPT, relevant work efficiency (Corollary 5.4.1): If there are k or more r-relevant jobs in the system, then $B_r = 1$.

We bound (5.7) in Theorem 5.5.3 using our novel MIAOW technique, and we bound (5.8) in Theorem 5.5.4.

Theorem 5.5.3 (Bound waste)**.** *Under the ServerFilling-SRPT policy, in the power-of-two setting,*

$$\int_{r=0}^\infty \frac{\mathbb{E}[(1-B_r)W_r]}{r^2(1-\rho_r^A)}dr \le e(k-1)\left\lceil \ln\frac{1}{1-\rho}\right\rceil.$$

The same is true of DivisorFilling-SRPT in the divisible setting.

Proof. First, we make use of the key fact about ServerFilling-SRPT (and DivisorFilling-SRPT), relevant work efficiency: If there are at least k jobs with rank $\le r$ in the system, then $B_r = 1$. This is proven in Corollary 5.4.1 for ServerFilling-SRPT, and in Section 5.7 for DivisorFilling-SRPT.

Let us define W_r^* to be the r-relevant work of the $k-1$ jobs of least remaining size in the system. Note that if $B_r < 1$, then $W_r = W_r^*$, for ServerFilling-SRPT and DivisorFilling-SRPT. As a result,

$$\int_{r=0}^{\infty} \frac{\mathbb{E}[(1-B_r)W_r]}{r^2(1-\rho_r^A)}dr = \int_{r=0}^{\infty} \frac{\mathbb{E}[(1-B_r)W_r^*]}{r^2(1-\rho_r^A)}dr.$$

Next, we will break up the range of remaining sizes $r \in [0,\infty)$ into a finite set of buckets. Let $\{r_0, r_1, \ldots, r_m\}$ be a list of m different remaining sizes, where $r_0 = 0$. We will specify the list $\{r_i\}$ later. Implicitly, we will say that $r_{m+1} = \infty$. We can rewrite the above integral as:

$$\int_{r=0}^{\infty} \frac{\mathbb{E}[(1-B_r)W_r^*]}{r^2(1-\rho_r^A)}dr = \sum_{i=0}^{m} \int_{r=r_i}^{r_{i+1}} \frac{\mathbb{E}[(1-B_r)W_r^*]}{r^2(1-\rho_r^A)}dr. \tag{5.9}$$

Next, we replace r with either r_i or r_{i+1}, selectively, to simplify things. Note that B_r is increasing as a function of r, because as we increase the rank r, more servers are busy with r-relevant jobs. Likewise, ρ_r^A is increasing as a function of r. Thus, for any $r \in [r_i, r_{i+1}]$,

$$B_{r_i} \le B_r, \qquad \rho_r^A \le \rho_{r_{i+1}}^A.$$

Substituting into the integral from (5.9), we find that

$$\int_{r=r_i}^{r_{i+1}} \frac{\mathbb{E}[(1-B_r)W_r^*]}{r^2(1-\rho_r^A)}dr \le \int_{r=r_i}^{r_{i+1}} \frac{\mathbb{E}[(1-B_{r_i})W_r^*]}{r^2(1-\rho_{r_{i+1}}^A)}dr.$$

Next, let us perform some algebraic manipulation:

$$\int_{r=r_i}^{r_{i+1}} \frac{\mathbb{E}[(1-B_{r_i})W_r^*]}{r^2(1-\rho_{r_{i+1}}^A)}dr = E\left[\int_{r=r_i}^{r_{i+1}} \frac{(1-B_{r_i})W_r^*}{r^2(1-\rho_{r_{i+1}}^A)}dr\right] = E\left[\frac{1-B_{r_i}}{1-\rho_{r_{i+1}}^A}\int_{r=r_i}^{r_{i+1}} \frac{W_r^*}{r^2}dr\right]. \tag{5.10}$$

Now, let us make use of the definition of W_r^*. Recall that W_r^* is the total remaining size of the $k-1$ jobs of least remaining size in the system.

$$W_r^* = \sum_{j=1}^{k-1} r_j \mathbb{1}\{r_j \le r\}$$

Substituting this into (5.10), we find it is equal to

$$= E\left[\frac{1-B_{r_i}}{1-\rho_{r_{i+1}}^A}\sum_{j=1}^{k-1}\int_{r=r_i}^{r_{i+1}} \frac{r_j\mathbb{1}\{r_j \le r\}}{r^2}dr\right]. \tag{5.11}$$

Now, we will bound the integral in (5.11). As noted in Lemma 5.5.1, for an arbitrary remaining size r_j,

$$\int_{r=0}^{\infty} \frac{r_j\mathbb{1}\{r_j \le r\}}{r^2}dr = r_j\int_{r=r_j}^{\infty} \frac{1}{r^2}dr = r_j\frac{1}{r_j} = 1.$$

As a result,

$$\int_{r=r_i}^{r_{i+1}} \frac{r_j \mathbb{1}\{r_j \le r\}}{r^2} dr \le 1.$$

Substituting in this bound into (5.11), we find that

$$E\left[\frac{1-B_{r_i}}{1-\rho^A_{r_{i+1}}} \sum_{j=1}^{k-1} \int_{r=r_i}^{r_{i+1}} \frac{r_j \mathbb{1}\{r_j \le r\}}{r^2} dr\right] \le E\left[\frac{1-B_{r_i}}{1-\rho^A_{r_{i+1}}}(k-1)\right]$$
$$= (k-1)E\left[\frac{1-B_{r_i}}{1-\rho^A_{r_{i+1}}}\right] = (k-1)\frac{1-\rho_{r_i}}{1-\rho^A_{r_{i+1}}} \le (k-1)\frac{1-\rho^A_{r_i}}{1-\rho^A_{r_{i+1}}}.$$

Returning all the way back to the beginning, we find that

$$\int_{r=0}^{\infty} \frac{\mathbb{E}[(1-B_r)W_r]}{r^2(1-\rho^A_r)} dr \le (k-1)\sum_{i=0}^{m} \frac{1-\rho^A_{r_i}}{1-\rho^A_{r_{i+1}}}. \tag{5.12}$$

We are now ready to construct the list $\{r_i\}$. Our goal in doing so is to minimize the sum

$$\sum_{i=0}^{m} \frac{1-\rho^A_{r_i}}{1-\rho^A_{r_{i+1}}}.$$

Our only constrains are that $r_0 = 0$ and $r_{m+1} = \infty$. In particular,

$$1-\rho^A_{r_0} = 1-\rho^A_0 = 1, \qquad 1-\rho^A_{r_{m+1}} = 1-\rho^A_\infty = 1-\rho.$$

All other r_i thresholds are ours to choose.

We will set r_i such that the values $1-\rho^A_{r_i}$ form a geometric progression. In particular, define $r_1, r_2, \ldots$ to satisfy the following:

$$1-\rho^A_{r_1} = \frac{1}{e}, \qquad 1-\rho^A_{r_2} = \frac{1}{e^2}, \qquad \ldots \qquad 1-\rho^A_{r_i} = \frac{1}{e^i} \quad \forall i \le m. \tag{5.13}$$

If the size distribution S is continuous, we choose r_i to exactly satisfy (5.13). If S is discontinuous, then ρ^A_r is discontinuous, so exact equality is not necessarily possible. However, it suffices to choose r_i such that

$$\frac{1}{e^i} \in [1-\rho^A_{r_i^+}, 1-\rho^A_{r_i^-}] \quad \forall i \le m,$$

which is always possible. By $^+$ and $^-$, we refer to the one-sided limits.

We then set $m = \lceil \ln \frac{1}{1-\rho} \rceil - 1$. This choice of $\{r_i\}$ ensures that

$$\frac{1-\rho^A_{r_i}}{1-\rho^A_{r_{i+1}}} \le e \quad \forall i \le m \tag{5.14}$$

$$\sum_{i=0}^{m} \frac{1-\rho^A_{r_i}}{1-\rho^A_{r_{i+1}}} \le e(m+1) = e\left\lceil \ln \frac{1}{1-\rho} \right\rceil. \tag{5.15}$$

For $i \leq m-1$, (5.14) follows immediately from (5.13). For $i = m$, (5.14) follows from the fact that $1 - \rho^A_{r_{m+1}} = 1 - \rho$.

Applying (5.12), we find that

$$\int_{r=0}^{\infty} \frac{\mathbb{E}[(1-B_r)W_r]}{r^2(1-\rho_r^A)} dr \leq e(k-1)\left\lceil \ln \frac{1}{1-\rho} \right\rceil. \qquad \square$$

Now, it remains to bound (5.8):

Theorem 5.5.4 (Bound recycled work)**.** *Under the ServerFilling-SRPT policy, in the power-of-two setting,*

$$\int_{r=0}^{\infty} \frac{\rho_r^R \mathbb{E}_r[W_r]}{r^2(1-\rho_r^A)} dr \leq (k-1) \ln \frac{1}{1-\rho}.$$

The same is true of DivisorFilling-SRPT in the divisible setting.

Proof. First, recall the key property of ServerFilling-SRPT and DivisorFilling-SRPT, relevant work efficiency: If there are at least k jobs with remaining size $\leq r$ in the system, then $B_r = 1$. This is proven in Corollary 5.4.1 for ServerFilling-SRPT, and in Section 5.7 for DivisorFilling-SRPT.

When a job r-recycles, it must have been in service despite having remaining size $> r$. As a result, there are at most $k-1$ other jobs with remaining size $\leq r$ present in the system at an r-recycling moment. Each such job contributes at most r work to W_r. As a result, $\mathbb{E}_r[W_r] \leq (k-1)r$.

$$\int_{r=0}^{\infty} \frac{\rho_r^R \mathbb{E}_r[W_r]}{r^2(1-\rho_r^A)} dr \leq \int_{r=0}^{\infty} \frac{(k-1)r\rho_r^R}{r^2(1-\rho_r^A)} dr = (k-1) \int_{r=0}^{\infty} \frac{\rho_r^R}{1-\rho_r^A} \frac{1}{r} dr.$$

To bound the integrand, we will expand the definitions of ρ_r^R and ρ_r^A in the SRPT setting.

$$\rho_r^R = \lambda r \mathbb{P}\{S > r\}$$
$$\rho_r^A = \lambda \mathbb{E}[S \mathbb{1}\{S \leq r\}].$$

We therefore bound as follows:

$$\frac{\rho_r^R}{1-\rho_r^A} \frac{1}{r} = \frac{\lambda r \mathbb{P}\{S > r\}}{1 - \lambda \mathbb{E}[S\mathbb{1}\{S \leq r\}]} \frac{1}{r} = \frac{\lambda \mathbb{P}\{S > r\}}{1 - \lambda \mathbb{E}[S\mathbb{1}\{S \leq r\}]}.$$

Now, note that $\mathbb{P}\{S > r\} = \frac{d}{dr}\mathbb{E}[\min(S, r)]$, and that $\mathbb{E}[\min(S,r)] \geq \mathbb{E}[S\mathbb{1}\{S \leq r\}]$. As a result,

$$\frac{\lambda \mathbb{P}\{S > r\}}{1 - \lambda \mathbb{E}[S\mathbb{1}\{S \leq r\}]} \leq \frac{\lambda \mathbb{P}\{S > r\}}{1 - \lambda \mathbb{E}[\min(S,r)]} = \frac{\lambda \frac{d}{dr}\mathbb{E}[\min(S,r)]}{1 - \lambda \mathbb{E}[\min(S,r)]} = -\frac{d}{dr} \ln \frac{1}{1 - \lambda \mathbb{E}[\min(S,r)]}.$$

Integrating over all $r \in [0, \infty)$, we find that

$$\int_{r=0}^{\infty} \frac{\rho_r^R}{1-\rho_r^A} \frac{1}{r} dr \leq \left[-\ln \frac{1}{1 - \lambda \mathbb{E}[\min(S,r)]} \right]_{r=0}^{\infty} = \ln \frac{1}{1-\rho}. \qquad \square$$

Now, we're ready to put it all together. We derive a bound on mean response time:

Theorem 5.5.1. In any multiserver-job system, the difference in mean response time between ServerFilling-SRPT and SRPT-1 is at most

$$\mathbb{E}[T^{SFS\text{-}k}] - \mathbb{E}[T^{SRPT\text{-}1}] \leq \frac{(e+1)(k-1)}{\lambda} \ln\left(\frac{1}{1-\rho}\right) + \frac{e}{\lambda}.$$

The same is true of DivisorFilling-SRPT in the divisible setting.

Proof. From Lemma 5.5.3, we know that

$$\begin{aligned}&\mathbb{E}[T^{SFS\text{-}k}] - \mathbb{E}[T^{SRPT\text{-}1}] \\ &= \frac{1}{\lambda}\int_0^\infty \frac{\mathbb{E}[(1-B_r^{SFS\text{-}k})W_r^{SFS\text{-}k}]}{r^2(1-\rho_r^A)}dr + \frac{1}{\lambda}\int_0^\infty \frac{\rho_r^R \mathbb{E}_r[W_r^{SFS\text{-}k}]}{r^2(1-\rho_r^A)}dr.\end{aligned}$$

We apply Theorem 5.5.3 and Theorem 5.5.4 to bound the two terms:

$$\mathbb{E}[T^{SFS\text{-}k}] - \mathbb{E}[T^{SRPT\text{-}1}] \leq \frac{1}{\lambda}e(k-1)\left\lceil \ln\frac{1}{1-\rho}\right\rceil + \frac{1}{\lambda}(k-1)\ln\frac{1}{1-\rho}.$$

We use the bound $\lceil x \rceil \leq x+1$ to simplify the resulting expression. □

Now, we use this bound to prove asymptotic optimality:

Theorem 5.5.2. If $\mathbb{E}[S^2(\log S)^+] < \infty$,

$$\lim_{\rho\to 1}\frac{\mathbb{E}[T^{SFS\text{-}k}]}{\mathbb{E}[T^{SRPT\text{-}1}]} = \lim_{\rho\to 1}\frac{\mathbb{E}[T^{SFS\text{-}k}]}{\mathbb{E}[T^{OPT\text{-}k}]} = 1.$$

The same is true of DivisorFilling-SRPT in the divisible setting.

Proof. From Theorem 5.5.1, we know that the gap $\mathbb{E}[T^{SFS\text{-}k}] - \mathbb{E}[T^{SRPT\text{-}1}]$ grows as $O(\log\frac{1}{1-\rho})$ in the $\rho \to 1$ limit. It is known that if $\mathbb{E}[S^2(\log S)^+] < \infty$, then $\mathbb{E}[T^{SRPT\text{-}1}] = \omega(\log\frac{1}{1-\rho})$ in the $\rho \to 1$ limit. This is proven in [202, Appendix B.2], and specifically in the proof of [202, Theorem 1.3]. □

5.6 ServerFilling-Gittins: Asymptotic Optimality with Unknown Sizes

We generalize our results to the setting of unknown sizes or of partially known sizes (e.g. size estimates). To do so, we replace the SRPT job ordering with the Gittins job ordering, thus creating the ServerFilling-Gittins (SFG-k) and DivisorFilling-Gittins policies.

5.6.1 Background

The Gittins policy is the optimal scheduling policy for minimizing mean response time in the M/G/1 in the unknown and partially-known size settings [72, 199], filling the same role as SRPT in the known-size setting.

The Gittins policy is an age-based index policy, meaning that it assigns each job a rank according to the job's age and static characteristics (e.g. server need), as well as any other information the scheduler may have, and serves the job of least rank. In the blind MSJ setting, the Gittins rank function can be defined as follows: Let S_i be the job size distribution of jobs with server need i. Then a job with server need i and age a has rank:

$$\inf_{b>a} \frac{\mathbb{E}[\min(S_i, b) - a \mid S_i > a]}{P[S_i \leq b | S_i > a]}.$$

The definition of the Gittins rank in settings where the server has more information is similar, but more complicated. For more details, see [199, 202].

We define the ServerFilling-Gittins policy by ordering jobs in increasing order of Gittins rank, and then applying the same ServerFilling procedure as described in Section 5.4.2. We define DivisorFilling-Gittins similarly, based on the DivisorFilling procedure given in Section 5.7.

5.6.2 Notation

Our notation follows [202]. We start by defining a job state space X of all possible job states x. For instance, in the unknown size setting, a job's state is simply its age a. In the known-size setting, a job's state was its remaining size. Every state x is mapped to $\mathrm{rank}(x)$. We call a job in state x r-relevant if $\mathrm{rank}(x) < r$.

Next, we need to adjust the concept of "remaining size" slightly. We define $S_r(x)$, the *r-relevant remaining size* of a job in state x, to be the random variable denoting the amount of service the job needs in order to reach an r-irrelevant state or complete. In the known-size case, this amount of service was deterministic, but here it is a random variable.

We can now define W_r, the *r-relevant work* in the system, to be the total of all jobs' r-relevant remaining size in steady state. Likewise, B_r is the fraction of servers occupied by r-relevant jobs.

We also define two state distributions: X^A, the state of arriving jobs, and X_r^R, the state of jobs recycling relative to rank r. In the known-size case, X_r^R is deterministic, and in the unknown size case, X^A is deterministic, but in general both are random variables. We also define λ_r^R to be the rate at which jobs recycle relative to rank r. This is equal to λ times the expected number of r-recyclings per job.

We can now define the two constituents of r-relevant load, ρ_r^A and ρ_r^R.

$$\rho_r^A := \lambda \mathbb{E}[S_r(X^A)]$$
$$\rho_r^R := \lambda_r^R \mathbb{E}_r[S_r(X_r^R)]$$

Now, we are ready to state our main result for ServerFilling-Gittins.

5.6.3 ServerFilling-Gittins: Proof Outline

Our main result for ServerFilling-Gittins is an analogous bound on mean response time to Theorem 5.5.1, our bound on mean response time for ServerFilling-SRPT:

Theorem 5.6.1. *For all loads ρ, in the power-of-two setting, the mean response time gap between ServerFilling-Gittins and Gittins-1 is at most*

$$\mathbb{E}[T^{SFG\text{-}k}] - \mathbb{E}[T^{Gittins\text{-}1}] \leq \frac{(e+1)(k-1)}{\lambda} \ln \frac{1}{1-\rho} + \frac{e}{\lambda}.$$

The same is true of DivisorFilling-Gittins in the divisible setting.

Note that this bound is in some ways stronger than the bound on Gittins-k given in [202]. Our bound is the first uniform bound on multiserver Gittins, meaning that our bound doesn't depend on S except via $\mathbb{E}[S]$, unlike the bound on Gittins-k in [202]. Note also that the M/G/k is a special case of the multiserver-job system when server needs are all 1, and that in this special case, ServerFilling-Gittins specializes to Gittins-k. As a result, Theorem 5.6.1 is a strict improvement upon the bound given in [202].

We use this bound to prove that ServerFilling-Gittins yields optimal mean response time in the heavy-traffic limit:

Theorem 5.6.2. *If $\mathbb{E}[S^2(\log S)^+] < \infty$, then ServerFilling-Gittins is asymptotically optimal in the multiserver-job system:*

$$\lim_{\rho \to 1} \frac{\mathbb{E}[T^{SFG\text{-}k}]}{\mathbb{E}[T^{Gittins\text{-}1}]} = \lim_{\rho \to 1} \frac{\mathbb{E}[T^{SFG\text{-}k}]}{\mathbb{E}[T^{OPT\text{-}k}]} = 1.$$

The same is true of DivisorFilling-Gittins in the divisible setting.

Theorem 5.6.2 follows from Theorem 5.6.1 just as Theorem 5.5.2 follows from Theorem 5.5.1.

To prove Theorem 5.6.1, an analogous proof to the proof of Theorem 5.5.1 given in Section 5.5.3 suffices. We simply must replace certain quantities used in Section 5.5.3 with the equivalent quantities for the Gittins policy. Specifically, rather than thinking of a job as r-relevant if it has remaining size $\leq r$, we instead think of a job as r-relevant if it has rank $\leq r$ under the Gittins policy. We redefine W_r^π, B_r^π, and ρ_r, ρ_r^A, and ρ_r^R accordingly, as described in Section 5.6.2. For full details, see Section 5.6.4.

The recycling term of our key background lemma Lemma 5.6.1 is likewise slightly different:

Lemma 5.6.1. *For any scheduling policy π,*

$$\mathbb{E}[T^\pi] - \mathbb{E}[T^{Gittins\text{-}1}] = \frac{1}{\lambda} \int_0^\infty \frac{\mathbb{E}[(1 - B_r^\pi) W_r^\pi]}{r^2(1-\rho_r^A)} dr \tag{5.16}$$

$$+ \frac{1}{\lambda} \int_0^\infty \frac{\lambda_r^R \mathbb{E}_r[S_r(X_r^R) W_r^\pi]}{r^2(1-\rho_r^A)} dr. \tag{5.17}$$

Here $\rho_R^r \mathbb{E}_r[W_r^\pi]$ from Lemma 5.5.3 becomes $\lambda_r^R \mathbb{E}_r[S_r(X_r^R) W_r^\pi]$. Note that in the SRPT case, $S_r(X^R) = r$, because under SRPT, a job r-recycles when its remaining size is r. Lemma 5.6.1 follows from [202, Theorem 7.2].

Bounding the waste term involving $\mathbb{E}[(1 - B_r^\pi) W_r^\pi]$ proceeds completely analogously to Theorem 5.5.3. Bounding the recycled work term involving $\lambda_r^R \mathbb{E}_r[S_r(X_r^R) W_r^\pi]$ is likewise completely analogous to Theorem 5.5.4. For the full details, see Section 5.6.4.

5.6.4 ServerFilling-Gittins: Full Proof

Our results for ServerFilling-Gittins follow near-identical proofs as given in Section 5.5.3 for ServerFilling-SRPT. We give the proofs here for completeness.

Our starting point is the "work integral number equality" (WINE) identity [198, 202].

Theorem 5.6.3 (Theorem 6.3, [202]). *The mean number of jobs and mean response time in an arbitrary system, under an arbitrary scheduling policy, is*

$$\mathbb{E}[N] = \lambda \mathbb{E}[T] = \int_0^\infty \frac{\mathbb{E}[W_r]}{r^2} dr.$$

Now, we can state the work-decomposition law in a Gittins system.

Theorem 5.6.4 (Theorem 7.2, [202]). *For all $r \geq 0$, the mean r-relevant work gap between an arbitrary policy π and M/G/1/Gittins is*

$$\mathbb{E}[W_r^\pi] - \mathbb{E}[W_r^{Gittins\text{-}1}] = \frac{\mathbb{E}[(1 - B_r^\pi) W_r^\pi] + \lambda_r^R \mathbb{E}_r[S_r(X_r^R) W_r^\pi]}{1 - \rho_r^A}. \tag{5.18}$$

We will handle the two numerator terms of (5.18) separately. Let us start by combining Theorem 5.6.3 with Theorem 5.6.4, and try to bound the resulting integral.

We must bound

$$\mathbb{E}[T^\pi] - \mathbb{E}[T^{Gittins\text{-}1}] = \frac{1}{\lambda} \int_0^\infty \frac{\mathbb{E}[(1 - B_r^\pi) W_r^\pi]}{r^2 (1 - \rho_r^A)} + \frac{1}{\lambda} \int_0^\infty \frac{\lambda_r^R \mathbb{E}_r[S_r(X_r^R) W_r^\pi]}{r^2 (1 - \rho_r^A)}.$$

We bound the first term in Lemma 5.6.2 and the second term in Lemma 5.6.4.

Lemma 5.6.2.

$$\int_{r=0}^\infty \frac{\mathbb{E}[(1 - B_r) W_r]}{r^2 (1 - \rho_r^A)} dr \leq e(k-1) \lceil \ln \frac{1}{1 - \rho} \rceil.$$

Proof. First, we make use of the key fact about ServerFilling-Gittins (and DivisorFilling-Gittins): If there are at least k jobs with rank $\leq r$ in the system, then $B_r = 1$. Thus, we can replace W_r by W_r', the work of the $k-1$ jobs of least rank in the system:

$$\int_{r=0}^\infty \frac{\mathbb{E}[(1 - B_r) W_r]}{r^2 (1 - \rho_r^A)} dr = \int_{r=0}^\infty \frac{\mathbb{E}[(1 - B_r) W_r']}{r^2 (1 - \rho_r^A)} dr.$$

Next, we will break up the ranks $r \in [0, \infty)$ into a finite set of buckets. Let $R = [r_1, r_2, \ldots]$ be a list of ranks, where $r_1 = 0$. We will specify the list R later. Implicitly, we will say that $r_{|R|+1} = \infty$. We can rewrite the above integral as:

$$\int_{r=0}^\infty \frac{\mathbb{E}[(1 - B_r) W_r']}{r^2 (1 - \rho_r^A)} dr = \sum_{i=1}^{|R|} \int_{r=r_i}^{r_{i+1}} \frac{\mathbb{E}[(1 - B_r) W_r']}{r^2 (1 - \rho_r^A)} dr. \tag{5.19}$$

Next, we replace r with either r_i or r_{i+1}, selectively, to simplify things. Note that B_r is increasing as a function of r - as we increase the rank r, more servers are busy with r-relevant jobs. Likewise, ρ_r^A is increasing as a function of r. Thus,

$$\begin{aligned} B_{r_i} &\leq B_r \\ \rho_r^A &\leq \rho_{r_{i+1}}^A. \end{aligned}$$

Substituting into the integral from (5.9), we find that

$$\int_{r=r_i}^{r_{i+1}} \frac{\mathbb{E}[(1-B_r)W_r']}{r^2(1-\rho_r^A)}dr \leq \int_{r=r_i}^{r_{i+1}} \frac{\mathbb{E}[(1-B_{r_i})W_r']}{r^2(1-\rho_{r_{i+1}}^A)}dr.$$

Next, let us perform some algebraic manipulation:

$$\int_{r=r_i}^{r_{i+1}} \frac{\mathbb{E}[(1-B_{r_i})W_r']}{r^2(1-\rho_{r_{i+1}}^A)}dr = E\left[\int_{r=r_i}^{r_{i+1}} \frac{(1-B_{r_i})W_r'}{r^2(1-\rho_{r_{i+1}}^A)}dr\right] = E\left[\frac{1-B_{r_i}}{1-\rho_{r_{i+1}}^A}\int_{r=r_i}^{r_{i+1}} \frac{W_r'}{r^2}dr\right].$$

Note that B_{r_i} and W_r' are conditionally independent because given $\vec{X}$, the current states of the jobs in the system, the busyness B_{r_i} is deterministic. We can make this explicit:

$$E\left[\frac{1-B_{r_i}}{1-\rho_{r_{i+1}}^A}\int_{r=r_i}^{r_{i+1}} \frac{W_r'}{r^2}\right]dr = E\left[\frac{1-B_{r_i}}{1-\rho_{r_{i+1}}^A}\int_{r=r_i}^{r_{i+1}} \frac{\mathbb{E}[W_r' \mid \vec{X}]}{r^2}dr\right]. \tag{5.20}$$

Next, let us recall the definition of W_r':

$$\begin{aligned} W_r' &= \sum_{j=1}^{k-1} S_r(X_j), \\ \mathbb{E}[W_r' \mid \vec{X}] &= \sum_{j=1}^{k-1} \mathbb{E}[S_r(X_j)|X_j]. \end{aligned}$$

Following [202], let us define $\text{SERVICE}(X_j, r)$ to be $\mathbb{E}[S_r(X_j)|X_j]$, the expected r-relevant work of a job X_j.

Substituting this into (5.20), we find that

$$\begin{aligned} &E\left[\frac{1-B_{r_i}}{1-\rho_{r_{i+1}}^A}\int_{r=r_i}^{r_{i+1}} \frac{\mathbb{E}[W_r' \mid \vec{X}]}{r^2}dr\right] \\ =&E\left[\frac{1-B_{r_i}}{1-\rho_{r_{i+1}}^A}\sum_{j=1}^{k-1}\int_{r=r_i}^{r_{i+1}} \frac{\text{SERVICE}(X_j, r)}{r^2}dr\right]. \end{aligned} \tag{5.21}$$

Now, let us make use of the basic fact about $\text{SERVICE}(X_j, r)$ from [202] which underlies Theorem 5.6.3:

For any job state X_j which is not the empty job,

$$\int_{r=0}^{\infty} \frac{\text{SERVICE}(X_j, r)}{r^2} dr = 1.$$

For the empty job, service is 0.

This provides a loose bound on the integral in (5.11), which integrates over a smaller interval of ranks. Substituting in this bound, we find that

$$E\left[\frac{1-B_{r_i}}{1-\rho^A_{r_{i+1}}} \sum_{j=1}^{k-1} \int_{r=r_i}^{r_{i+1}} \frac{\text{SERVICE}(X_j, r)}{r^2} dr\right] \leq E\left[\frac{1-B_{r_i}}{1-\rho^A_{r_{i+1}}} \min\{N, k-1\}\right]$$
$$\leq (k-1)E\left[\frac{1-B_{r_i}}{1-\rho^A_{r_{i+1}}}\right] = (k-1)\frac{1-\rho^A_{r_i}-\rho^R_r}{1-\rho^A_{r_{i+1}}} \leq (k-1)\frac{1-\rho^A_{r_i}}{1-\rho^A_{r_{i+1}}}.$$

Returning all the way back to the beginning, we find that

$$\int_{r=0}^{\infty} \frac{\mathbb{E}[(1-B_r)W_r]}{r^2(1-\rho^A_r)} dr \leq (k-1) \sum_{r=0}^{|R|} \frac{1-\rho^A_{r_i}}{1-\rho^A_{r_{i+1}}}.$$

To optimize this bound, we need to choose R to minimize this sum. To do so, we set $|R| = \lceil \ln \frac{1}{1-\rho} \rceil$, and choose r_i such that

$$\frac{1-\rho^A_{r_i^+}}{1-\rho^A_{r_{i+1}^-}} \leq e.$$

for all $i < |R|$. By $^+$ and $^-$, we refer to the left and right limits, thereby handling the possibility that ρ^A_r is discontinuous as a function of r. We therefore find that

$$\int_{r=0}^{\infty} \frac{\mathbb{E}[(1-B_r)W_r]}{r^2(1-\rho^A_r)} dr \leq e(k-1)\left\lceil \ln \frac{1}{1-\rho} \right\rceil.$$

□

Now, it remains to bound the recyclings term in (5.18). Note that this term is identical to the one in [202], so we can use essentially the same approach - we just disentangle it from the other term. First, we use a basic theorem from [202]:

Lemma 5.6.3 (Lemma 8.2, [202])**.**

$$\lambda^R_r \mathbb{E}_r[S_r(X^R_r)W_r] \leq (k-1)r\rho^R_r.$$

Now, it remains to bound the recyclings-dependent term, plugged into Theorem 5.6.3.

Lemma 5.6.4.

$$\int_{r=0}^{\infty} \frac{(k-1)r\rho^R_r}{r^2(1-\rho^A_r)} dr \leq (k-1)\ln\frac{1}{1-\rho}$$

Proof. First, let us simplify:

$$\int_{r=0}^{\infty} \frac{(k-1)r\rho_r^R}{r^2(1-\rho_r^A)}dr = (k-1)\int_{r=0}^{\infty} \frac{\rho_r^R}{1-\rho_r^A}\frac{1}{r}dr.$$

To bound the integrand, we will explicitly consider the Gittins game. Using the definitions of $\text{UNDONE}_A(r)$, and $\text{GAME}_A(r)$ given in Appendix B.2 of [202], we bound as follows:

$$\begin{aligned}\frac{\rho_r^R}{1-\rho_r^A}\frac{1}{r} &\leq \frac{\lambda r\text{UNDONE}_A(r)}{1-\lambda(\text{GAME}_A(r)-r\text{UNDONE}_A(r))}\frac{1}{r} \\ &\leq \frac{\lambda\text{UNDONE}_A(r)}{1-\lambda\text{GAME}_A(r)} = \frac{\lambda\frac{d}{dr}\text{GAME}_A(r)}{1-\lambda\text{GAME}_A(r)} = \frac{d}{dr}\ln\frac{1}{1-\lambda\text{GAME}_A(r)}.\end{aligned}$$

Above, we make use of [202, Lemma 5.3].

Integrating over all $r \in [0,\infty)$, we find that

$$\begin{aligned}\int_{r=0}^{\infty} \frac{\rho_r^R}{1-\rho_r^A}\frac{1}{r}dr &\leq \left[\ln\frac{1}{1-\lambda\text{GAME}_A(r)}\right]_{r=0}^{\infty} \\ &= \ln\frac{1}{1-\lambda\text{GAME}_A(\infty)} - \ln\frac{1}{1-\lambda\text{GAME}_A(0)}.\end{aligned}$$

From the definition of the Gittins game, it is straightforward to prove that $\text{GAME}_A(0) = 0$, and that $\text{GAME}_A(\infty) = \mathbb{E}[S]$.

As a result,

$$\int_{r=0}^{\infty} \frac{\rho_r^R}{1-\rho_r^A}\frac{1}{r}dr \leq \ln\frac{1}{1-\rho}.$$

□

Now, we're ready to put it all together. We derive a bound on mean response time:

Theorem 5.6.1. In any multiserver-job system in the power-of-two setting the difference in mean response time between ServerFilling-Gittins and Gittins-1 (resource pooled) is at most

$$\mathbb{E}[T^{SFG\text{-}k}] - \mathbb{E}[T^{Gittins\text{-}1}] \leq \frac{(e+1)(k-1)}{\lambda}\ln\left(\frac{1}{1-\rho}\right) + \frac{e}{\lambda}.$$

The same is true of DivisorFilling-Gittins in the divisible setting.

Proof. Combine Theorem 5.6.3 with Theorem 5.6.4, using Lemma 5.6.2 and Lemma 5.6.4 to bound the two terms. □

Note that this bound is in some ways stronger than the bound on Gittins-k given in [202]. Our bound is the first uniform bound on multiserver Gittins, meaning that our bound doesn't depend on S except via $\mathbb{E}[S]$, unlike the bound on Gittins-k in [202]. Note also that the M/G/k is a special case of the multiserver-job system when server needs are all 1, and that in this special case, ServerFilling-Gittins specializes to Gittins-k. As a result, Theorem 5.6.1 is a strict improvement upon the bound given in [202].

Analogous to Theorem 5.5.2, we use our bound to prove that ServerFilling-Gittins (and DivisorFilling-Gittins) achieve asymptotically optimal mean response time.

Theorem 5.6.2. If $\mathbb{E}[S^2(\log S)^+] < \infty$,

$$\lim_{\rho \to 1} \frac{\mathbb{E}[T^{SFG\text{-}k}]}{\mathbb{E}[T^{Gittins\text{-}1}]} = 1.$$

Note that $\mathbb{E}[T^{SRPT\text{-}1}] \leq \mathbb{E}[T^{Gittins\text{-}1}]$ by the optimality of SRPT, so $\mathbb{E}[T^{Gittins\text{-}1}] = \omega(\log \frac{1}{1-\rho})$ whenever $\mathbb{E}[S^2(\log S)^+] < \infty$, just as $\mathbb{E}[T^{SRPT\text{-}1}] = \omega(\log \frac{1}{1-\rho})$ in this case.

5.7 DivisorFilling-SRPT

The DivisorFilling-SRPT policy is a scheduling policy for the divisible server needs setting of the multiserver-job system, where all server needs k_j perfectly divide the total number of servers k.

To implement DivisorFilling-SRPT, we order jobs in increasing order of remaining size r_j, and then apply a recursive procedure to select the jobs to serve, which we will specify in Section 5.7.1. DivisorFilling-Gittins is defined identically, replacing increasing remaining size order with increasing rank order.

DivisorFilling-SRPT achieves two key guarantees:

1. DivisorFilling-SRPT always serves a subset of the k jobs of least remaining size in the system.
2. If at least k jobs are present, DivisorFilling-SRPT serves jobs with total server need exactly k.

Item 1 is part of the definition of DivisorFilling-SRPT in Section 5.7.1. We prove Item 2 as Theorem 5.7.1 in Section 5.7.2.

DivisorFilling-SRPT is identical to the DivisorFilling policy defined in Section 4.7 [84, Appendix A], except that DivisorFilling orders jobs in the arrival ordering, while DivisorFilling-SRPT orders jobs in SRPT order. Note that [84] is the electronic companion to [86], and that Appendix A only appears in the electronic companion.

As a corollary of Items 1 and 2, we can prove the "relevant work efficiency" property for DivisorFilling-SRPT:

Corollary 5.7.1 (Relevant work efficiency)**.** *Under the DivisorFilling-SRPT policy, in the divisible setting, if there are k or more r-relevant jobs in the system, all servers are occupied by r-relevant jobs.*

The same is true for DivisorFilling-Gittins.

From Corollary 5.7.1, we can use the same techniques as were used for ServerFilling-SRPT to prove Theorems 5.5.1 and 5.5.2.

5.7.1 DivisorFilling-SRPT Definition

Order all jobs in the system in order of least remaining size. Let M be the set of k jobs with least remaining size, or all jobs if less than k are present.

We now split into three cases:

1. M contains at least $k/6$ jobs with server need $k_j = 1$.

2. $k = 2^a 3^b$ for some integers a, b, and M contains $< k/6$ jobs with $k_j = 1$.
3. k has largest prime factor $p \geq 5$, and M contains $< k/6$ jobs with $k_j = 1$.

Case 1

If M contains at least $k/6$ jobs with server need 1, we initially parallel the ServerFilling-SRPT policy: we order jobs in M by server need (tiebroken by least remaining size), and place jobs into service in that order. However, because server needs are not powers of two, we may reach a point where no more jobs fit into service, but servers are still unoccupied. In this case, we place jobs from M with server need 1 into service, again tiebroken by least remaining size. We continue doing so until all k servers are full or no more server need 1 jobs remain.

Case 2

Suppose that k is of the form $2^a 3^b$, and that Case 1 does not apply.

We will recurse on one of two subsets of M: the set of jobs with even server need, or the set of jobs of odd server need greater than 1. Note that all jobs in the latter subset have server needs divisible by 3. We call the former subset M_2 and the latter subset M_3. To decide which subset to recurse on, we compare the values $2|M_2|$ and $3|M_3|$, and recurse on the subset whose value is larger. In the case of a tie, we arbitrarily select M_2.

If $2|M_2|$ is larger, we will only serve jobs from among M_2. To decide which jobs to serve, imagine that we combine pairs of servers. Doing so reduces k by a factor of 2, and reduces the server need of each job in M_2 by a factor of 2. We now recursively compute which jobs from M_2 the DivisorFilling-SRPT policy would serve in this subproblem, and serve those same jobs. If $3|M_3|$ is larger, we combine triples of servers, and then perform the same recursion.

Case 3

Suppose that k has largest prime factor $p \geq 5$, and that Case 1 does not apply.

Let M_p be the set of jobs in M with server need divisible by p. If $p|M_p| \geq k$, we recurse as in Case 2 by combining groups of p servers.

Otherwise, we will only serve jobs from M whose server need is *not* divisible by p, and also greater than 1. Let M_r be this subset of M. Note that all jobs in M_r have server needs which are divisors of k/p. We therefore construct a set M' consisting of the k/p jobs of M_r with least remaining size. If less than k/p jobs are in M_r, M' is all of M_r. We then apply the DivisorFilling-SRPT procedure to M', setting the total number of servers $k' = k/p$ in the subproblem. We extract the subset of jobs that DivisorFilling-SRPT serves in the subproblem from M_r. We repeat this process by extracting subsets from the remaining jobs in M_r, repeating until we have extracted p subsets from M_r, or M_r contains no jobs. DivisorFilling-SRPT serves all jobs that were served in any of the p subproblems.

Note that this set of jobs served is valid to serve, with total server need at most k, because each of the p subproblems have total server need at most k/p.

5.7.2 DivisorFilling-SRPT Fills All Servers

Our proof mirrors the proof in [84, Appendix A], which we reprove to make this chapter self-contained.

Theorem 5.7.1. *If at least k jobs are present, DivisorFilling-SRPT serves a set of jobs with total server need exactly k.*

The same is true for DivisorFilling-Gittins.

Proof. We will prove that if M contains k jobs, DivisorFilling-SRPT serves all k jobs. Our proof proceeds by strong induction on k. Specifically, assume that for all $k' < k$, if M' consists of at least k' jobs whose server needs divide k', then DivisorFilling-SRPT run on M' serves a set of jobs with total server need k'. We will show that this assumption implies the desired result for k servers.

Again, we split into three cases:

1. M contains at least $k/6$ jobs with server need k_j=1.
2. $k = 2^a 3^b$ for some integers a, b, and M contains $< k/6$ jobs with $k_j = 1$.
3. k has largest prime factor $p \geq 5$, and M contains $< k/6$ jobs with $k_j = 1$.

Case 1

Suppose M contains at least $k/6$ jobs with server need 1.

Let us label the jobs in M as $m_1, m_2, \ldots$ in decreasing order of server need:

$$k_{m_1} \geq k_{m_2} \geq \ldots.$$

Let i^* be defined as

$$i^* = \arg\max_i \sum_{\ell=1}^{i} k_{m_\ell} \leq k.$$

In Case 1, DivisorFilling-SRPT serves jobs $m_1, \ldots m_{i^*}$, as well as any jobs with $k_j = 1$ that fit in the remaining servers. Let us write $\text{SUM}_i := \sum_{\ell=1}^{i} k_{m_\ell}$. Because M contains at least $k/6$ jobs with server need 1, to prove Theorem 5.7.1 in this case, it suffices to show that $\text{SUM}_{i^*} \geq 5k/6$. The remaining servers are filled by the jobs with server need 1.

First, note that $\text{SUM}_k \geq k$, because M contains k jobs, each with server need at least 1. Next, note that $k - \text{SUM}_{i^*} < k_{m_{i^*+1}}$, by the definition of i^*. Because the labels $m_1, m_2, \ldots$ are in decreasing order of server need, $k - \text{SUM}_{i^*} < k_{m_{i^*}}$.

We will now proceed by enumerating the possible sequences of the i^* largest server needs in M. To prove that $k - \text{SUM}_{i^*} \leq k/6$, we need only consider such sequences where all server needs are greater than $k/6$. Such sequences consist only of the elements $k, k/2, k/3, k/4, k/5$. We enumerate all possible such sequences in Table 5.2. Note that if k is not divisible by all of $\{2, 3, 4, 5\}$, some entries will not apply. This only tightens the resulting bound on $k - \text{SUM}_{i^*}$ for such k.

As shown in Table 5.2, in all cases $k - \text{SUM}_{i^*} \leq k/6$. The remaining servers are filled with jobs with server need 1. DivisorFilling-SRPT serves a set of jobs with total server need exactly k, as desired. As a result, Theorem 5.7.1 holds in this case.

Sequence $k_{m_1}, \ldots, k_{m_{i^*}}$	$k - \text{SUM}_{i^*}$	Sequence $k_{m_1}, \ldots, k_{m_{i^*}}$	$k - \text{SUM}_{i^*}$
k	0	$k/2, k/2$	0
$k/2, k/3$	$k/6$	$k/2, k/4, k/4$	0
$k/2, k/4, k/5$	$k/20$	$k/2, k/5, k/5$	$k/10$
$k/3, k/3, k/3$	0	$k/3, k/3, k/4$	$k/12$
$k/3, k/3, k/5$	$2k/15$	$k/3, k/4, k/4$	$k/6$
$k/3, k/4, k/5, k/5$	$k/60$	$k/3, k/5, k/5, k/5$	$k/15$
$k/4, k/4, k/4, k/4$	0	$k/4, k/4, k/4, k/5$	$k/20$
$k/4, k/4, k/5, k/5$	$k/10$	$k/4, k/5, k/5, k/5$	$3k/20$
$k/5, k/5, k/5, k/5, k/5$	0		

Table 5.2: All possible sequences of the i^* largest server needs in M in which all server needs exceed $k/6$.

Case 2

Suppose that $k = 2^a 3^b$ for integers a, b, and that Case 1 does not apply.

Recall that M_2 is the set of jobs in M with even server need, and that M_3 is the set of jobs with odd server need, with server need greater than 1. We recurse on one of these subsets, by comparing $2|M_2|$ and $3|M_3|$. Note that if M_2 is recursed on, the total number of servers in the subproblem is $k/2$, and all server needs are divisors of $k/2$. For M_3, the same is true of $k/3$.

For Theorem 5.7.1 to hold inductively, we must show that if M_2 is recursed on, then $|M_2| \geq k/2$, and that if M_3 is recursed on, then $|M_3| \geq k/3$. Because we select a subset by comparing $2|M_2|$ and $3|M_3|$, if either set is large enough, the set recursed on will be large enough.

Because there are $< n/6$ jobs with server need 1, $|M_2| + |M_3| \geq 5k/6$. Therefore, either $|M_2| \geq k/2$ or $|M_3| \geq k/3$.

Suppose that $2|M_2| \geq 3|M_3|$. Call M_2' the set of jobs in M_2, but with all server needs reduced by a factor of 2. M_2' is the subset that DivisorFilling recurses on. Because $|M_2'| = |M_2| \geq k/2$ in this case, by our inductive hypothesis the recursive call returns a subset of M_2' with total server need $k/2$. The corresponding jobs in M_2 have total server need k, so DivisorFilling-SRPT serves a set of jobs with total server need exactly k, completing the inductive step in this case. If $3|M_3| \geq 2|M_2|$, then $|M_3| \geq k/3$, and the same argument applies.

Case 3

Suppose that k has largest prime factor $p \geq 5$, and that Case 1 does not apply.

If $p|M_p| \geq k$, Theorem 5.7.1 holds inductively, by the same argument as in Case 2.

Let us therefore focus on the extraction procedure. We must show that the extraction procedure always extracts p subsets with total server need exactly k/p, to ensure that the overall set served has total server need k.

Note that $|M_p| < k/p \leq k/5$, and that there are $\leq k/6$ jobs with server need 1. M_r consists of the remaining jobs. As a result,

$$|M_r| \geq k - k/6 - k/5 = \frac{19k}{30}.$$

Note also that every job in $|M_r|$ has server need at least 2. The total server need extracted in each step is at most k/p, so the number of jobs extracted is at most $k/2p$. To prove that p subsets each with total server need k/p can be extracted, it suffices to show that at least k/p jobs remain after the first $p-1$ subsets have been extracted.

The number of jobs remaining at this point is at least:

$$\frac{19k}{30} - \frac{(p-1)k}{2p} = \frac{19k}{30} - \frac{k}{2} + \frac{k}{2p} = \frac{2k}{15} + \frac{k}{2p}.$$

To prove that the number of jobs remaining is at least k/p, we just need to show that $2k/15 \geq k/2p$. But $p \geq 5$, so $2k/15 > k/10 \geq k/2p$.

Therefore, by induction, each of the p subsets extracted from M_r has total server need k/p. Combining these subsets gives a total server need of k. Therefore, DivisorFilling-SRPT serves a set of jobs with total server need exactly k, as desired. □

5.8 Empirical Results

We have proven that ServerFilling-SRPT yields asymptotically optimal mean response time in the heavy-traffic limit (as $\rho \to 1$). To empirically validate our theoretical results and broaden our comparison to general ρ, we use simulation to compare the mean response time of ServerFilling-SRPT to that of several previously proposed policies:

MaxWeight: A throughput optimal policy which considers all possible sets of jobs that can be served at a time. Each job is given a weight equal the number of jobs in the system with the same server need. The set of jobs with the maximum total weight is served [148]. Note that this policy requires solving a NP-hard Bin Packing problem for each service.

ServerFilling: A policy which orders jobs in arrival order, then uses the same procedure to place jobs onto servers as our ServerFilling-SRPT policy specified in Section 5.4.2. ServerFilling is throughput-optimal in the power-of-two setting [86].

We also compare against resource-pooled SRPT-1, our lower bound on the optimal policy.

In Fig. 5.5, we show the ratio of mean response time between the multiserver-job policies and SRPT-1. As proven in Theorem 5.5.2, for ServerFilling-SRPT, this ratio converges to 1, implying that ServerFilling-SRPT yields asymptotically optimal mean response time. In contrast, for MaxWeight and ServerFilling, the ratio is far from one, and appears to diverge. ServerFilling-SRPT has superior mean response time at all ρ.

In Fig. 5.6, we show a setting with higher variance job sizes, where $C^2 = 10$. In high-variance settings, making effective use of job size information is at its most important. Here, the ratio for ServerFilling-SRPT again converges smoothly to 1, while the ratios for MaxWeight and ServerFilling diverge rapidly.

In Section 5.2, Fig. 5.4, we also compared ServerFilling-SRPT against two size-based heuristic policies:

GreedySRPT: Order jobs in increasing order of remaining size. As long as sufficient servers are available, place jobs into service. When a job has higher server need than the remaining number of servers available, stop.

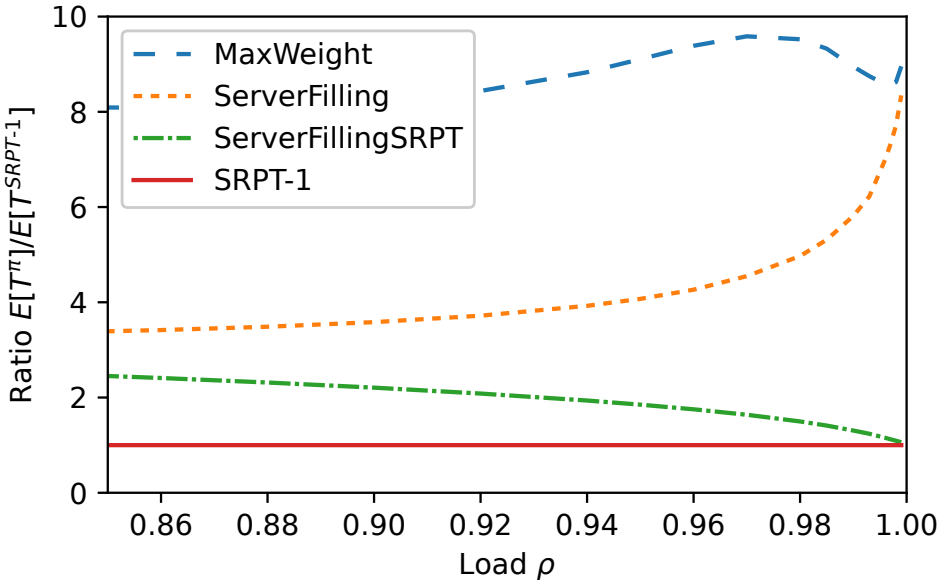


Figure 5.5: Ratio of mean response time between several multiserver-job policies and SRPT-1. K uniformly sampled from $\{1, 2, 4, 8\}$. S exponentially distributed, independent of K. Each simulation consists of 10^7 arrivals. Loads up to $\rho = 0.999$ simulated.

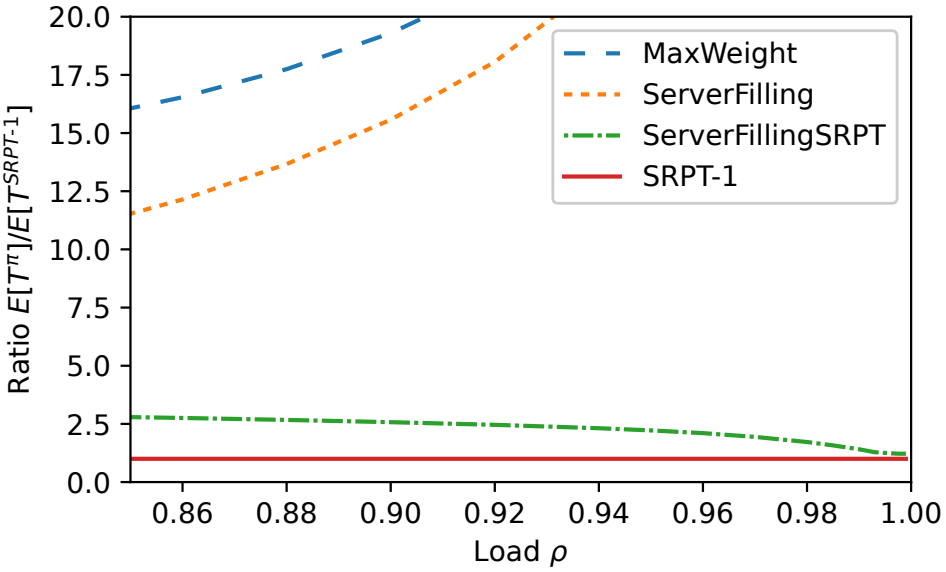


Figure 5.6: Ratio of mean response time between several multiserver-job policies and SRPT-1 under high variance. K uniformly sampled from $\{1, 2, 4, 8\}$. S hyperexponentially distributed, $C^2 = 10$, independent of K. Each simulation consists of 10^7 arrivals. Loads up to $\rho = 0.999$ simulated.

FirstFitSRPT: Order jobs in increasing order of remaining size. As long as sufficient servers are available, place jobs into service. If a job has higher server need than the remaining number of servers available, skip that job. Continue through the list of jobs, placing jobs into service if sufficient servers are available, until all servers are full, or all jobs are exhausted. This policy was studied under the name "Smallest Area First" [33].

GreedySRPT makes no effort to pack jobs efficiently onto servers, while FirstFitSRPT is unreliable at doing so. For both of these policies, the stability region is significantly smaller than the optimal stability region. This is why neither policy is depicted in Fig. 5.5 or Fig. 5.6, as both are unstable for all loads $\rho \geq 0.85$, and hence have infinite mean response time on this domain.

We summarize our experiments as follows: In all experiments, at all ρ, ServerFilling-SRPT has minimal mean response time.

5.9 Conclusion

We introduce the ServerFilling-SRPT scheduling policy for the multiserver-job system. We prove a tight bound on the mean response time of ServerFilling-SRPT in the power-of-two setting, which applies for all loads ρ. We use that bound to prove that ServerFilling-SRPT achieves asymptotically optimal mean response time in heavy traffic. We also show that ServerFilling-SRPT empirically achieves the best mean response time of any policy simulated, across all loads ρ. Finally, we introduce the DivisorFilling-SRPT policy, in the more general divisible setting, and the ServerFilling- and DivisorFilling-Gittins policies, in the settings of unknown- and partially-known job sizes, proving similar asymptotic optimality results for each.

One of the major insights of this chapter is that achieving asymptotically optimal mean response time requires prioritizing jobs of small remaining size without sacrificing the throughput of the system. ServerFilling-SRPT is the first policy to achieve both goals simultaneously.

The MIAOW analysis technique introduced in this chapter extends beyond ServerFilling-SRPT and the multiserver-job setting. In fact, it allows the analysis of any system and any policy in which the relevant work efficiency property (Corollary 5.4.1) can be proven.

One direction of future work is to study multiserver-job scheduling policies outside of the divisible setting. No mean response time analysis is currently known for any scheduling policy in this more general setting, much less any optimality results, so new techniques will likely be needed. In particular, no policy with the remaining work efficiency property can exist in this setting.

5.10 General Conclusion

5.10.1 Summary

We start by summarizing the results proven in this chapter, as well as the key techniques behind these results.

Results: ServerFilling-SRPT is optimal We prove that out novel ServerFilling-SRPT scheduling policy, which we define in Section 5.4.2, achieves the best possible mean response

time when the load on the system is high and queue lengths become long (See Theorem 5.5.2).

We also prove an upper bound on the mean response time of ServerFilling-SRPT, in the form of a clean mathematical formula (See Theorem 5.5.1). This upper bound proves that ServerFilling-SRPT achieves similar mean response time to that of resource-pooled SRPT, in which all k servers are combined into a single ultra-fast server, which runs k times faster than each of the original servers. This bound becomes tight as load becomes high.

Simulation result: ServerFilling-SRPT is near-optimal under moderate traffic While our results focus on heavy traffic, we show via simulation in Fig. 5.6 that ServerFilling-SRPT is also very good under moderate traffic. In particular, we see that the ratio between ServerFilling-SRPT's mean response time and that of resource-pooled SRPT is near 1 even at moderate loads, unlike other MSJ scheduling policies. This implies that ServerFilling-SRPT is near-optimal even at moderate loads.

Results: Further optimal policies We prove similar optimality results and upper bounds for several more MSJ scheduling policies. While the ServerFilling-SRPT policy focuses on workloads where all server needs are powers of 2, and the total number of servers k is a power of 2, the DivisorFilling-SRPT policy can handle any workload where all server needs are perfect divisors of the total number of servers. We define the DivisorFilling-SRPT policy in Section 5.7. We prove DivisorFilling-SRPT's optimality in Theorem 5.5.2 and bound its response time in Theorem 5.5.1.

The ServerFilling-SRPT and DivisorFilling-SRPT results focus on the setting where the scheduling policy knows the job sizes. If job sizes are unknown or estimated, the appropriate analogue to SRPT scheduling is the Gittins index scheduling policy [72, 73, 199], Correspondingly, we define the ServerFilling-Gittins and DivisorFilling-Gittins policies in Section 5.6. We prove in Theorem 5.6.2 that these policies each achieve optimal mean response time in the limit as load approaches capacity, and prove upper bounds on each policy's mean response time in Theorem 5.6.1. These proofs follow essentially the same approach as our results on ServerFilling-SRPT and DivisorFilling-SRPT.

Key challenge: Single straggler The key challenge that we overcome in proving our bounds and optimality results is the "single straggler" problem. Suppose that, out of many jobs in the system, there is only a single 1-server job, and suppose that job has a very small size. Note that $1 = 2^0$ is a power of 2, and that 1 is the only odd power of 2. All other jobs have much larger sizes, and have even server needs. Now the scheduling policy is in a bind: It must either deviate from SRPT scheduling, by leaving the 1-server job in the queue, or it must waste a server, by scheduling the 1-server job. Because k is even and all other server needs are even, the 1-server job cannot be served without leaving at least one server empty.

ServerFilling-SRPT chooses to leave the 1-server job in the queue. As a result, despite the fact that the 1-server job has a very small size, it can have a very long response time. However, note that this unfortunate scenario can only happen to a single 1-server job at a time. Therefore, there is hope that even though the 1-server job has a long response time, the overall mean response time of the system may still be short.

This single-straggler scenario blocks us from using many previous techniques that we have developed to prove optimality results in multiserver scheduling settings, including the techniques used to prove optimality for one-server-per-job SRPT in Chapter 2 and one-server-per-job Gittins [204]. We discuss the single-straggler obstacle further in Section 5.5.2

Overcoming single straggler: MIAOW To overcome the single-straggler obstacle, we develop a new analysis technique called Multiplicative Interval Analysis of Waste (MIAOW). Rather than focusing on a single job at a time or a single size of jobs at a time, MIAOW bounds the harmful mean-response-time impact of an entire interval of sizes simultaneously. For instance, MIAOW might be used to analyze all inefficiency involving jobs whose sizes are in the "interval" from 1 to 2 server-seconds. MIAOW might then show that all jobs in that interval collectively only cause a 1-second increase in mean response time in ServerFilling-SRPT as compared to resource-pooled SRPT, regardless of the load on the system.

We use the MIAOW technique to prove Theorem 5.5.3, the key step towards our optimality result and our upper bound on ServerFilling-SRPT. Using the MIAOW technique, we make use of the fact that only one job can be caught in the single straggler scenario at a time. This implies that the single straggler scenario cannot increase the overall mean response time of ServerFilling-SRPT by a non-negligible amount, in the limit as load on the system approaches the capacity of the system.

5.10.2 Future Direction: General MSJ Scheduling

This chapter invents the ServerFilling-SRPT scheduling policy and proves that it achieves optimal mean response time in the limit as load approaches the capacity of the system. However, the ServerFilling policy is limited to the setting where all jobs' server needs are powers of 2, and where k, the total number of servers, is also a power of 2. Even the more general DivisorFilling policy is limited to the setting where all jobs' server needs are divisors of k.

As we discuss in Section 8.3.4, much less is known about achieving optimal mean response time in the general MSJ scheduling setting. This is true even in settings where 100% server utilization (i.e. *full* server utilization) is achievable. For instance, consider a system with $k = 3$ servers, server needs 1, 2, and 3, and where the load of 1-server jobs exceeds that of 2-server jobs. Full server utilization can be achieved by pairing 1-server jobs with 2-server jobs for service, to ensure that servers are never wasted. However, the response time under such a policy may not resemble that of a resource-pooled SRPT system. For instance, suppose that 1-server jobs typically have smaller size than 2-server jobs. We need to pair up 1-server and 2-server jobs to maintain full utilization, but by doing so we must diverge drastically from the ideal SRPT service ordering.

We pose the following open problem:

> What MSJ scheduling policy achieves optimal mean response time under general workloads?

5.10.3 Potential Impact

We now explore potential directions in which the ServerFilling-SRPT and DivisorFilling-SRPT scheduling policies could be applied.

Adopting ServerFilling-SRPT into Modern Computing Systems

Our analysis of the ServerFilling-SRPT scheduling policy in the MSJ model shows that ServerFilling-SRPT can dramatically lower response times compared to other scheduling policies, and that ServerFilling-SRPT is in fact optimal for mean response time under sufficiently heavy load. Our hope is that our results will lead computing systems operators to adopt ServerFilling-SRPT scheduling in their systems. However, the road from theoretical results to adoption requires overcoming several hurdles. These include:

Judicious use of preemption. In real systems, preemption can be difficult or expensive, either from an implementation perspective or because of performance overheads. ServerFilling-SRPT preempts jobs for two reasons: Upon completions, to keep servers filled, and upon arrivals, if a smaller job arrives. To keep completion-triggered preemptions to a minimum, one could expand the pool of jobs that are under consideration for service, to increase the chance that all servers can be filled without preemption. To lessen arrival-triggered preemptions, one could decide to only remove a job from service if a much smaller job arrives. If the newly-arrived job is only slightly smaller, then one might err on the side of avoiding preemption.

Multidimensional resources. In real systems, jobs often require a variety of resources, including CPU cores, GPUs, memory, network bandwidth, disk IO, and more. These can be modeled as multidimensional resource requirements, in contrast to the single-dimensional server need model considered in this chapter. In a general multidimensional setting, full utilization is not attainable. There is no equivalent of the power-of-2 server need assumption which can ensure that any assortment of jobs will always be able to use all of the system's resources. Jobs will use more of one resource or another, inevitably wasting some resources.

Instead of full utilization, one should aim for *optimal stability region*. A policy achieves optimal stability if it can handle as large an arrival rate as possible without jobs backing up indefinitely and queue lengths diverging to infinity. Several optimal-stability scheduling policies are known in the multidimensional setting [148, 181], However, these policies do not incorporate size information, and do not achieve particularly low mean response times. To achieve better mean response time in the multidimensional setting, one might start with an optimal-stability policy, and then tilt its decisions towards favoring small jobs. We discuss scheduling under multidimensional resource constraints further in Section 8.3.4.

ServerFilling-SRPT for Scheduling Jobs Involving People

When scheduling jobs involving people, the duration of service and the number of people (i.e. server need) necessary to fulfil that job are not the only important qualities of the job. Jobs can be differentiated both by their sizes, summarizing duration and server need, and by the time-sensitivity or *value* of that service. Value might represent a customer paying extra for rush service, or a task that needs to be completed before other work can begin. An appropriate metric for this is scenario is the mean product of value and response time, also known as *weighted response time*.

In the single-server setting, if both value and size are known, the policy which minimizes weighted mean response time is the "$c\mu$-rule", which serves the job with maximal ratio of value to size. We would therefore use the ServerFilling-$c\mu$ policy, ordering jobs by their ratio of value to size before performing the ServerFilling procedure. In fact, thanks to prior work on the Gittins policy [199], our optimal mean response time result for ServerFilling-Gittins, Theorem 5.6.2, immediately implies a similar optimal *weighted* mean response time result for ServerFilling-$c\mu$.

Chapter 6

Analyzing Multiserver-job First-Come First-Served

This chapter is based on the paper "The RESET and MARC Techniques, with Application to Multiserver-Job Analysis", currently under submission, written with my coauthors Yige Hong, Mor Harchol-Balter, and Alan Scheller-Wolf.

An extended abstract of this chapter has been published under the title "The RESET Technique for Multiserver-Job Analysis", and was presented at the SIGMETRICS 2023 Student Research Competition [79].

6.1 General Introduction

Modeling large-scale computing Modern computing systems scale in two important ways. First, they contain huge numbers of servers, allowing them to process many jobs at once. Second, the *jobs* in modern computing systems require vastly different amount of resources, ranging from a single CPU core to thousands of machines. The scheduling decision in this system consists of selecting a combination of jobs to serve at once. It is vital to understand the performance of existing scheduling policies, so we can make informed decisions about changing workloads, capacity provisioning, and other aspects of system design.

To capture the behavior of these computing systems, we use a *multiserver-job* queueing model, depicted in Fig. 6.1, which was introduced in Chapter 4. Jobs arrive randomly over time, and wait in a central queue. Each job has a *server need*, which specifies the number of servers the jobs requires in order to enter service. All servers are identical, so the scheduling policy can decide to serve any set of jobs with total server need at most k, where k is the total number of servers in the system.

FCFS scheduling One important MSJ scheduling policy, which is the focus of this chapter, is the First-Come First-Served (FCFS) policy. We depict FCFS in Fig. 6.1. FCFS is a natural and practical scheduling policy that is the default in both cloud computing [58, 146, 208] and supercomputing [61, 120]. FCFS places the oldest jobs in the system into service one-by-one, until a job is reached whose server need exceeds the number of currently available servers. At this point, the blocked job, and all jobs behind it in the queue, must wait until some of the jobs in

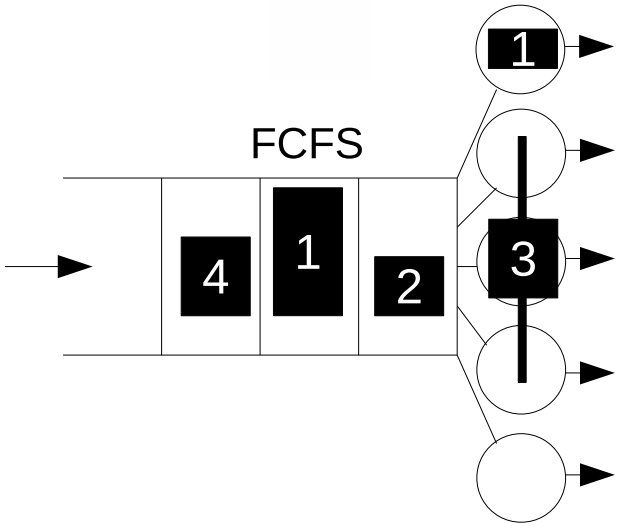


Figure 6.1: The FCFS scheduling policy in the multiserver-job setting. Each job has two characteristics: A *server need*, the number of servers it requires in order to run, and a *duration*, the amount of service time it requires. FCFS serves jobs in arrival order. In the figure, the job with server need 2 cannot receive service yet, because there is only 1 server available. The 2-server job blocks all jobs behind it in the queue from receiving service either.

service complete and more servers become available. To measure FCFS's performance, we are interested in its *mean response time*, the mean time from when a job arrives to when it completes.

Prior analysis requires assumptions In Chapters 4 and 5, we analyzed the mean response time of ServerFilling and ServerFilling-SRPT, two scheduling policies that we invented. These are the only two mean response time analyses of any MSJ scheduling policies. While both policies have advantageous properties, they both make assumptions about the behavior of the MSJ system and its workload. In particular, both policies assume that jobs may be *preempted*, meaning that the jobs may be paused and put back in the queue, to be returned to later. In addition, both policies assume that all jobs' server needs are powers of two, and that k is a power of two. While the assumptions are valid in some settings, they are not valid in all settings. No mean response time analysis is known for any non-preemptive scheduling policies, nor for any scheduling policy under a general server need distribution.

Key question: Analyzing FCFS FCFS does not use preemption, and it does not make any assumptions on the server need distribution. The key question of this chapter is:

What is the mean response time of FCFS in the multiserver-job model?

Key challenge: Wasted servers As shown in Fig. 6.1, FCFS can leave some servers empty, even when there are lots of jobs in the queue. As a result, the system is not completing work at a constant rate. At some times, more servers are busy, and work is completed at a faster rate, and at some times, fewer servers are busy, and work is completed slower. This is a major obstacle to many prior approaches to analyzing mean response time. In particular, the fact that FCFS wastes servers means that FCFS does *not* resemble a *resource-pooled* system. A resource-pooled system is one in which all k servers are combined into one gigantic server which runs k times faster than each of the original servers. Resource-pooled analysis is at the heart of our mean response time

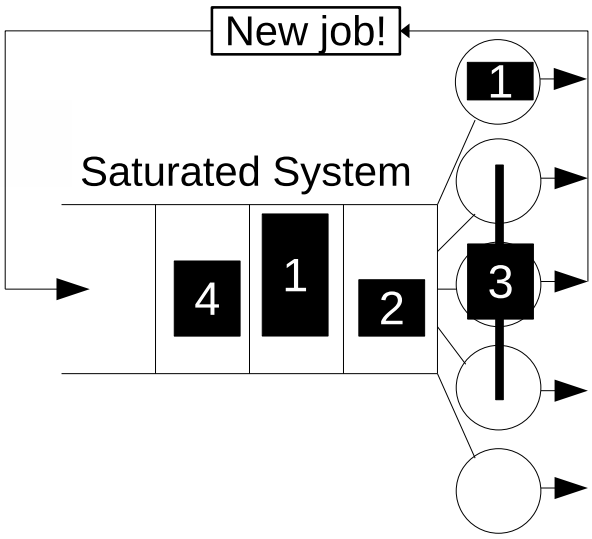


Figure 6.2: The saturated system for the MSJ FCFS scheduling policy. The saturated system is a closed system in which completions trigger new arrivals. As a result, there are always a fixed number of jobs present in the system. Specifically, we set the number of jobs to always be exactly k, the number of servers. In this case, there are always $k = 5$ jobs in the system.

analysis in each of Chapters 2 to 5. Unfortunately, it cannot handle systems where servers are wasted, because no capacity can be wasted in the resource-pooled system.

Key technique: Saturated system In order to analyze the MSJ system, we need to get a grip on its pattern of server use and waste over time, under a given workload. Our key tool for understanding this behavior is the *saturated system*. The saturated system, shown in Fig. 6.2, is a variant of the MSJ system where, instead of jobs arriving due to an external arrival process, job arrivals are instead triggered by job completions. As a result, the number of jobs in the saturated system is always constant. Nothing else changes: Jobs still have random server needs and durations. The server need and duration of an arriving job are unrelated to the characteristics of the job whose completion triggered that arrival.

As the external arrivals to the original MSJ system get faster and faster, and the MSJ system's queue gets longer and longer, it more and more closely approaches the saturated system's behavior, in the following sense: We show that the distribution of jobs in service in the two systems converges in this limit (See Lemma 6.7.5). This property has previously been used to characterize the MSJ system's *stability region*, the exact arrival where the MSJ system's queue length diverges to infinity [13, 65, 83, 88, 187].

Saturated system analysis to response time By analyzing the saturated system, we understand the patterns of server utilization and wastage. This in turn allows us to analyze the mean response time of the original MSJ FCFS system.

6.2 Technical Introduction

Multiserver queueing theory predominantly emphasizes models in which each job utilizes only one server (one-server-per-job models), such as the M/G/k. For decades, such models were popular in the study of computing systems, where they provided a faithful reflection of the behavior of such systems while remaining conducive to theoretical analysis. However, one-server-per-job models no longer reflect the behavior of many modern computing systems.

Multiserver jobs: In modern datacenters, such as those of Google, Amazon, and Microsoft, each job now requests many servers (cores, processors, etc.), which the job holds simultaneously. A job's "server need" refers to the number of servers requested by the job. In Google's recently published trace of its "Borg" computation cluster [86, 218], the server needs vary by a factor of 100,000 across jobs. Throughout this chapter, we will focus on this "multiserver-job model" (MSJ), in which each job requests some number of servers, and concurrently occupies that many servers throughout its time in service (its "duration").

FCFS service: We specifically study the first-come first-served (FCFS) service ordering for the MSJ model, a natural and practical policy that is the default in both cloud computing [58, 146, 208] and supercomputing [61, 120]. Currently, little is known about FCFS service in MSJ models.

Stability under FCFS: Even the stability region under FCFS scheduling is not generally understood. Some papers characterize the stability region under restrictive assumptions on the job duration distributions [5, 161, 187, 188]. A key technique in these papers is the *saturated system* approach [13, 65]. The saturated system is a closed system in which completions trigger new arrivals, so that the number of jobs in the system is always constant. We are the first to use the saturated system for analysis beyond characterizing the stability region.

Response time for FCFS: Even less is known about mean response time $\mathbb{E}[T]$ in MSJ FCFS systems: The only MSJ FCFS system in which mean response time has been analytically characterized is the simpler case of 2 servers and exponentially distributed durations [31, 63]. Mean response time is much better understood under more complex scheduling policies such as ServerFilling and ServerFilling-SRPT [86, 87], but these policies require assumptions on both preemption and the server need distribution, and do not capture current practices, which emphasize non-preemptive policies. Mean response time is also better understood in MSJ FCFS scaling regimes, where the number of servers and the arrival rate both grow asymptotically [113, 115, 226]. We are the first to analyze MSJ FCFS mean response time under a fixed number of servers.

Why FCFS is hard to analyze: One source of difficulty in studying the FCFS policy is the lack of work conservation. In simpler one-server-per-job models, a work-conservation property holds: If enough jobs are present, no servers will be idle. The same is true under ServerFilling and ServerFilling-SRPT [86]. Work conservation is key to the mean response time analysis of those systems, as one can often reduce the analysis of response time to the analysis of work. In contrast, the multiserver-job model under FCFS service is not work conserving: a job must wait if it demands more servers than are currently available, leaving those servers idle.

First response time analysis: We derive the first characterization of mean response time in the MSJ FCFS system. We allow any phase-type duration distribution, and any correlated distribution of server need and duration. Our result holds at all loads up to an additive error, which becomes negligible as the arrival rate λ approaches λ^*, the threshold of stability.

Figure 6.3: The structure of our main results: RESET (Theorem 6.5.2) and MARC (Theorem 6.5.1).

Proof structure: We illustrate the structure of our proof in Fig. 6.3. We first use our RESET technique (REduction to Saturated for Expected Time) to reduce from the MSJ FCFS system to a M/M/1 with Markovian service rate (MMSR), where the service rate is based on the saturated system. We next use our MARC technique (MArkovian Relative Completions) to prove Theorem 6.5.1, the first characterization of mean response time in the MMSR. Both steps are novel, hard, and of independent interest. We prove our MARC result first because it is a standalone result, characterizing mean response time for any MMSR system up to an additive constant. We then prove Theorem 6.5.2, our characterization of mean response time in the MSJ FCFS system, by layering our RESET technique on top of MARC.

Breadth of the RESET technique: Our RESET technique is very broad, and applies to a variety of generalizations of the MSJ model and beyond (See Section 6.10). For instance, RESET can handle cases where a job's server need varies throughout its time in service, and where the service rates at the servers can depend on the job. Finally, we can analyze scheduling policies that are close to FCFS but allow limited reordering, such as some backfilling policies.

Breadth of the MARC technique: Our MARC technique is also very broad, and applies to any MMSR system. For example, we can handle systems in which machine breakdowns lead to reduced service rate, or where servers are taken away by higher-priority customers.

This chapter is organized as follows:

- Section 6.3: We discuss prior work on the MSJ model.
- Section 6.4: We define the MSJ model, the MMSR, the saturated system, relative completions, and related concepts.
- Section 6.5: We state our main results, and walk through an example of applying our results to a specific MSJ FCFS system.
- Section 6.6: We characterize mean response time in the MMSR using our MARC technique.
- Section 6.7: We build upon Section 6.6 to characterize MSJ FCFS mean response time using our RESET technique.
- Section 6.10: Our results apply to a very broad class of models which we call "finite skip" models, and which we define in this section.
- Section 6.11: We empirically validate our theoretical results.

6.3 Prior work

The bulk of the prior work we discuss is in Section 6.3.1, which focuses on specific results in the multiserver-job model. In Section 6.3.2, we briefly discuss prior work on the saturated system, an important tool in our analysis. Finally, in Section 6.3.3, we discuss prior work on the M/M/1 with Markovian service rate.

6.3.1 Multiserver-job model

Theoretical results in the multiserver-job model are limited. We first discuss the primary setting of this chapter: a fixed number of servers and FCFS service.

Fixed number of servers, FCFS service

In this setting, most results focus on characterizing the stability region. Rumyantsev and Morozov characterize stability for an MSJ system with an arbitrary distribution of server needs, where the duration distribution is exponential and independent of server need [187]. This result can implicitly be seen as solving the saturated system, which has a product-form stationary distribution in this setting. A setting with two job classes, each with distinct server needs and exponential duration distributions has also been considered [83, 88, 186]. In this setting, the saturated system was also proven to have a product-form stationary distribution, which was also used to characterize the stability region.

The only setting in which mean response time $\mathbb{E}[T]$ is known is in the case of $k = 2$ servers and exponential duration independent of server need [31, 63]. In this setting, the exact stationary distribution is known. Mean response time is open in all other settings, including whenever $k > 2$.

Advanced scheduling policies

More advanced scheduling policies for the MSJ system have been investigated, in order to analyze and optimize the stability region and mean response time.

The MaxWeight policy was proven to achieve optimal stability region in the MSJ setting [147]. However, its implementation requires solving an NP-hard optimization problem upon every transition, and it performs frequent preemption. It is also too complex for response time analysis to be tractable. The Randomized Timers policy achieves optimal throughput with no preemption [71, 181], but has very poor empirical mean response time, and no response time analysis.

In some settings, it is possible for a scheduling policy to ensure that all servers are busy whenever there is enough work in the system, which we call "work conservation." Work conservation enables the optimal stability region to be achieved and mean response time to be characterized. Two examples are ServerFilling and ServerFilling-SRPT scheduling policies [86, 87]. However, the work-conservation-based techniques used in these papers cannot be used to analyze non-work-conserving policies such as FCFS.

Scaling number of servers

The MSJ FCFS model has also been studied in settings where the number of servers, the arrival rate, and the server need distribution all grow in unison to infinity. Analogues of the Halfin-Whitt and non-diminishing-slowdown regimes have been established, proving bounds on the probability of queueing and mean waiting time [113, 115, 226]. These results focus on settings where an *approximate* work conservation property holds, and there is enough excess capacity that this approximate work conservation is sufficient to determine the first-order behavior of the system. These results do not apply to the $\lambda \to \lambda^*$ limit.

6.3.2 Prior work on the saturated system

The *saturated system* is a queueing system which is used as analysis tool to understand the behavior of an underlying non-saturated queueing system [13, 65]. Baccelli and Foss state that it is a "folk theorem" that the stability region of the original open queueing system is equivalent to the completion rate of the saturated system: If the completion rate of the saturated system is μ, then the original system is stable for arrival rate λ if and only if $\lambda < \lambda^* = \mu$ [13]. Baccelli and Foss give sufficient conditions for this folk theorem, known as the "saturation rule," to hold rigorously. These conditions are mild, and are easily shown to hold for the MSJ FCFS system. The strongest stability results in the MSJ FCFS system have either been proven by characterizing the steady state of the saturated system, or are equivalent to such a characterization [83, 88, 187].

Our novel contribution is characterizing the *mean response time* behavior of an original system by reducing its analysis to the analysis of a saturated system. All previous uses of the saturated system focused on characterizing stability.

6.3.3 M/M/1 with Markovian Service Rate

The M/M/1 with Markovian service rate (MMSR) has been extensively studied since the 50's, often alongside Markovian arrival rates [39, 47, 95, 131, 151, 166]. A variety of mathematical tools have been applied to the MMSR, including generating function methods, matrix-analytic and matrix-geometric methods, and spectral expansion methods [39, 47, 145, 166]. However, these methods primarily result in *numerical results*, rather than theoretical insights [47, 155].

More is known for special cases of the MMSR system [49, 179]. For instance, the case where arrival rates alternate between a high and low completion rate at some frequency has received specific study. In this case, the generating function can be explicitly solved as the root of a cubic equation [239], but the resulting expression is too complex for analytical insights. In this simplified setting, scaling results [167–169, 223] and monotonicity results [95] have been derived, but those results do not extend to more complex MMSR systems.

By contrast, our MARC technique provides the first explicit characterization of mean response time for the general MMSR system, up to an additive constant.

6.4 Model

In this section, we introduce five queueing models: the multiserver-job (MSJ) model, the M/M/1 with Markovian service rate (MMSR), the At-least-k (Ak) model, the saturated system, and the simplified saturated system (SSS). The MSJ is the main focus of this chapter. Our RESET technique reduces its analysis to analyzing the Ak system. The Ak system is equivalent to a MMSR system whose completion process is controlled by the saturated system. Our MARC technique allows us to analyze this MMSR system. The SSS is a simpler equivalent of the saturated system. We also introduce the concepts of relative completions and the generator approach, which are key to our analysis.

6.4.1 Multiserver-job Model

The MSJ model is a queueing model in which each job requests an integer number of servers, the *server need*, for some duration of time, the *service duration*. Each job requires concurrent service on all of its servers throughout its duration. Let k denote the total number of servers in the system.

We assume that each job's server need and service duration are drawn i.i.d. from some joint distribution. The duration distribution is phase type, and it may depend on the job's server need. This assumption can likely be generalized, which we leave to future work. We assume a Poisson(λ) arrival process.

We focus on the first-come first-served (FCFS) service discipline. Our RESET technique also applies to many other scheduling policies, as we discuss in Section 6.10. Under FCFS, jobs are placed into service one by one in arrival order as long as the total server need of the jobs in service is at most k. If a job is reached whose server need would push the total over k, that job does not receive service until sufficient completions occur. We consider head-of-the-line blocking, so no subsequent jobs in arrival order receive service. It has been shown that in the MSJ FCFS setting, there exists a threshold λ^*, such that the system is stable if and only if $\lambda < \lambda^*$ [13, 65]. We assume that $\lambda < \lambda^*$.

Note that the only jobs eligible for service are the k oldest jobs in arrival order. We conceptually divide the system into two parts: the *front*, which consists of the (up to) k oldest jobs in arrival order in the system, and the *queue*, which consists of all other jobs in the system.

6.4.2 Running Example

Throughout this section, we will use a running example to clarify notation and concepts. Consider a MSJ setting with $k = 2$ servers, and two classes of jobs: $2/3$ of jobs have server need 1 and duration $Exp(1)$, and the other $1/3$ of jobs have server need 2 and duration $Exp(1/2)$.

6.4.3 M/M/1 with Markovian Service Rate

The MMSR is a queueing system where jobs arrive to the system according to a Poisson process, and complete at a variable rate, where the completions are determined by the transitions of a finite-state Markov chain. When a job arrives, it stays in the queue until it reaches the head of

the line, entering service. The job then completes when the service-process Markov chain next undergoes a transition associated with a completion. Jobs are identical until they reach service. The service process is unaffected by the number of jobs in the queue.

6.4.4 At-least-k System

To connect the MSJ FCFS and MMSR systems, we define two systems: the "At-least-k" (Ak) system, and the "saturated system" in Section 6.4.5. The Ak model mimics the MSJ model, except that the front of the Ak model always has exactly k jobs. Specifically, in addition to the primary Poisson(λ) arrival process, whenever there are exactly k jobs in the system, and a job completes, a new job immediately arrives. The server need and service duration of this job are sampled i.i.d. from the same distribution as the primary arrivals. Due to these extra arrivals, the Ak system always has at least k jobs present.

Intuitively, the Ak system should have about k more jobs present in steady state than the MSJ system. We thus expect the Ak and MSJ systems to have the same asymptotic mean response time, up to an $O_\lambda(1)$ term. We make this intuition rigorous by using our RESET technique to prove Theorem 6.5.2.

6.4.5 Saturated System

The saturated system is a closed multiserver-job system, where completions trigger new arrivals.[1] Jobs are served according to the same FCFS service discipline. There are always exactly k jobs in the system. Whenever a job completes, a new job with i.i.d. server need and service duration is sampled. The state descriptor is just an ordered list of exactly k jobs.

In our running example with $k = 2$ servers, the state space of the saturated system consists of all orderings of 2 jobs:

$$\mathbb{Y}^{\text{Sat}} = \{[1,1], [1,2], [2,1], [2,2]\}$$

The leftmost entry in each of the lists is the oldest job in FCFS order. In state $[1,2]$, a 1-server job is in service and a 2-server job is in the queue, while in state $[2,1]$, a 2-server job is in service and a 1-server job is in the queue.

6.4.6 Equivalence between MMSR-Sat and At-least-k

Now we are ready to connect the MMSR and At-least-k (Ak) systems. Consider the subsystem consisting only of the front of the Ak system. This subsystem is stochastically identical to the saturated system. Whenever a job completes at the front of the Ak system, a new job enters the front, either from the queue or from the auxiliary arrival process. This matches the saturated system's completion-triggered arrival process.

As a result, the Ak system is stochastically equal to an MMSR system whose service process is controlled by the saturated system. To clarify this equivalency, assume the Ak system starts in

[1] Baccelli and Foss [13] consider a system with an infinite number of jobs in the queue, which is equivalent to our closed system.

a certain front state y with an empty queue. Then equivalently the MMSR system starts empty, with its service process in state y. If a job completes in the Ak system then a new job is generated, and the same transition occurs in the service process in the MMSR system. Next, assume a job arrives to the Ak system, and sits in the queue. At the same time, a job arrives in the MMSR and sits in the queue. Through this mapping, the two systems are sample-path equivalent.

6.4.7 Notation

A state of the MSJ system consists of a pair $(q^{\text{MSJ}}, y^{\text{MSJ}})$. The queue length q^{MSJ} is a nonnegative integer. A job state consists of a server need and a phase of its phase-type duration. The front state y^{MSJ} is a list of up to k job states. If $q^{\text{MSJ}} > 0$, then y^{MSJ} must consist of exactly k job states, while if $q^{\text{MSJ}} = 0$, y^{MSJ} may consist of anywhere from 0 to k job states. Let $\mathbb{Y}^{\text{MSJ}}$ denote the set of all possible front states y^{MSJ} of the MSJ system. For instance, in our running example, $\mathbb{Y}^{\text{MSJ}} = \{[], [1], [2], [1,1], [1,2], [2,1], [2,2]\}$. Note that in the first three states, q^{MSJ} must equal 0.

In the MMSR system, let π denote the Markov chain that modulates the service rate. As a superscript, it signifies "the MMSR system controlled by the Markov chain π." A state of the MMSR-π system consists of a pair (q^π, y^π). The queue length q^π is a nonnegative integer. The state y^π is a state of the service process π, and $\mathbb{Y}^\pi$ is the state space of π.

Because the MMSR-Sat system is stochastically equal to the Ak system, we use the superscripts $^{\text{Sat}}$ and $^{\text{Ak}}$ interchangeably. A state of the Ak system is a pair $(q^{\text{Ak}}, y^{\text{Ak}})$. In contrast to the MSJ system, y^{Ak} always consists of exactly k job states. In particular, $\mathbb{Y}^{\text{Ak}} \subset \mathbb{Y}^{\text{MSJ}}$.

When the service process π transitions from state y to y', there are two possibilities: Either a completion occurs, which we write as $a = 1$, or no completion occurs, which we write as $a = 0$. We therefore define $\mu^\pi_{y,y',a}$ to denote the system's transition rate from front state y to front state y', accompanied by a completions, where $a \in \{0, 1\}$. For instance, in our running example $\mu^{\text{Sat}}_{[1,1],[1,2],1} = 2/3$. Let the total completion rate from state y be denoted by $\mu^\pi_{y,\cdot,1} = \sum_{y'} \mu^\pi_{y,y',1}$. For instance, in our running example $\mu^{\text{Sat}}_{[1,1],\cdot,1} = 2$.

Let $\mu^{\text{MSJ}}_{y,y',a,b}$ denote a transition rate in the Multiserver-job system, where y, y', and a have the same meaning as in $\mu^{\text{Ak}}_{y,y',a}$. Let $b = \mathbb{1}_{q>0}$ denote whether this transition is associated with an empty queue ($b = 0$), or an occupied queue ($b = 1$). Note that if $y \notin \mathbb{Y}^{\text{Ak}}$, then $b = 0$ for all nonzero $\mu^{\text{MSJ}}_{y,y',a,b}$, while if $y \in \mathbb{Y}^{\text{Ak}}$, then both values of b are possible. Note that $\forall y \in \mathbb{Y}^{\text{Ak}}, \mu^{\text{MSJ}}_{y,y',a,1} = \mu^{\text{Ak}}_{y,y',a}$.

If a job arrives to the MSJ system and finds that the front state y has fewer than k jobs ($y \notin \mathbb{Y}^{\text{Ak}}$), a fresh job state is sampled and appended to y. Let S be the distribution over fresh job states, let i be a particular fresh job state, let p_i be the probability $\mathbb{P}\{S = i\}$, and let $y \cdot i$ be the new front state with a job in state i appended. For instance, in the running example, $p_1 = 2/3, p_2 = 1/3$.

We will study the time-average steady states of each of these systems, which we write $(Q^{\text{MSJ}}, Y^{\text{MSJ}})$, (Q^π, Y^π), etc. Let Y^π_d denote the departure-average steady state of the MMSR service process π: the steady-state distribution of the embedded DTMC which samples states after each departure from π.

Let X^π denote the long-term throughput of the service process π. Let λ^*_π denote the stability region of the MMSR-π system. Note that $X^\pi = \lambda^*_\pi$ by prior results relating the saturated system

to the stability region of the original system [13, 65]. In particular, $X^{\text{Sat}} = \lambda^*_{\text{Sat}} = \lambda^*$, where λ^* denotes the stability region of the MSJ FCFS system. We will typically write λ^* to avoid confusion between X^{Sat} and a random variable.

A concrete example of this notation is provided in Section 6.5.1.

6.4.8 Relative completions

Key to our MARC technique is the novel idea of *relative completions*, which we define for a general MMSR-π system. Let y_1 and y_2 be two states of the service process π. The difference in relative completions between two states y_1 and y_2 is the long-term difference in expected completions between an instance of the service process starting in state y_1 and one starting in y_2. Specifically, let $C_\pi(y, t)$ denote the number of completions up to time t of the service process of π initialized in state y at time $t = 0$. Then let $\Delta_\pi(y_1, y_2)$ denote the relative completions between states y_1 and y_2.

$$\Delta_\pi(y_1, y_2) = \lim_{t\to\infty} \mathbb{E}[C_\pi(y_1, t) - C_\pi(y_2, t)].$$

We prove that $\Delta_\pi(y_1, y_2)$ always exists and is always finite in Lemma 6.9.1. We also allow y_1 and/or y_2 to be distributions over states, rather than single states. Specifically, we will often focus on the case where y_2, rather than being a single state, is the steady state distribution Y^π. In this case, note that $\mathbb{E}[C_\pi(Y^\pi, t)] = X^\pi t = \lambda^*_\pi t$. When it is clear from context, we write $\Delta_\pi(y)$ to denote $\Delta_\pi(y, Y^\pi)$. The relative completions formula for this case simplifies:

$$\Delta_\pi(y) = \Delta_\pi(y, Y^\pi) = \lim_{t\to\infty} \mathbb{E}[C_\pi(y, t)] - \lambda^*_\pi t. \tag{6.1}$$

6.4.9 Generator

We also make use of the *instantaneous generator* of each of our queueing systems, which is the stochastic equivalent of the derivative operator. The instantaneous generator is an operator which takes a function from system states to real values, and returns a function from system states to real values. The generator operator is specific to a given Markov chain. Let η be a Markov chain, and let G^η denote the generator operator for η, which is defined as follows:

For any real-valued function of the state of η, $f(q, y)$,

$$G^\eta \circ f(q, y) := \lim_{t\to 0} \frac{1}{t} \mathbb{E}[f(Q^\eta(t), Y^\eta(t)) - f(q, y) | Q^\eta(0) = q, Y^\eta(0) = y].$$

Importantly, the expected value of the generator in steady state is zero:

Lemma 6.4.1. *Let f be a real-valued function of the state of a Markov chain η. Assume that the transition rates of the Markov chain η are uniformly bounded, and $\mathbb{E}[f(Q^\eta, Y^\eta)] < \infty$. Then*

$$\mathbb{E}_{(q,y)\sim(Q^\eta, Y^\eta)}[G^\eta \circ f(q, y)] = 0. \tag{6.2}$$

Proof deferred to Section 6.9. □

We show in 6.9 that (6.2) holds for the MSJ, MMSR, At-least-k, and Saturated systems, for any $f(q, y)$ with polynomial dependence on q.

6.4.10 Asymptotic notation

We use the notation $O_\lambda(f(\lambda))$ to represent a function $g(\lambda)$ such that

$$\exists \text{ a constant } M \text{ such that } |g(\lambda)| \leq M|f(\lambda)| \quad \forall \lambda, 0 < \lambda < \lambda^*.$$

6.4.11 Simplified saturated system

While the saturated system is a finite-state system, it can have a very large number of possible states. However, many of the states have identical behavior, and can be combined to reduce the state space. For instance, in our running example, the states $[2, 1]$ and $[2, 2]$ are nearly identical: in both states just a 2-server job is in service. We therefore simplify the system by combining the two states into the state $[2]$, and delaying sampling the next job until needed.

We refer to the resulting system as the "simplified saturated system" (SSS), in contrast to the "main saturated system," defined in Section 6.4.5, which is the focus of the bulk of this chapter.

The simplified saturated system is a closed system which always contains jobs with total server need $\geq k$, and contains the minimal number of jobs to reach that threshold. Whenever a job completes, the system admits new jobs until the total server need is $\geq k$. Jobs are served in FCFS order. Note that at most one job in the system is not in service.

In particular, a state of the SSS consists of a multiset of job states for the jobs in service, plus the server need of the job in the queue, if any. The total server need of these jobs is just enough to be $\geq k$.

For instance, consider a system with $k = 30$ jobs, and server needs either 3 or 10, and exponential durations. The main saturated system has state space $\mathbb{Y}^{\text{Sat}} = \{3, 10\}^{30}$, with over a billion states. In contrast, the simplified saturated system has 13 states. We will write each state as a triple, consisting of the server need of the job in the queue, and the number of 3-server and 10-server jobs in service. Then the state space of the SSS is:

$$\mathbb{Y}^{\text{SSS}} = \{[\emptyset, 0, 3], [10, 1, 2], [10, 2, 2], [3, 3, 2], [10, 3, 2], [10, 4, 1], [10, 5, 1], \\ [3, 6, 1], [10, 6, 1], [10, 7, 0], [10, 8, 0], [10, 9, 0], [\emptyset, 10, 0]\}$$

Despite its much smaller state space, the SSS has essentially identical behavior to the main saturated system:

Lemma 6.4.2. *There exists a coupling under which the main saturated system and simplified saturated system have identical completions.*

Proof. To form the coupling, let us sample in advance the entire arrival sequence: For each arrival, we pre-sample which initial state it will arrive in.

Next, we initialize both systems based on this arrival sequence: For the main saturated system, the first k jobs are initially present, while for the simplified saturated system, a subset of those jobs are initially present. Note that the set of jobs in service in the main saturated system is identical to the set of jobs in service in the simplified saturated system, because the total server need of jobs in service is at most k. Note that the ordering of the jobs in service does not affect any transitions, so the fact that SSS does not track this information poses no obstacle. We will ensure that the set of jobs in service in the two systems is identical throughout time.

Next, we couple the two systems' completions and job state transitions to be identical. Jobs' states can only change while those jobs are in service, so this coupling is valid as long as the set of jobs in service is identical in both systems. Finally, whenever a pair of jobs completes, new jobs are generated according to the shared global arrival sequence. This ensures that the jobs that enter service are identical in the two systems.

By construction, the set of jobs in service is always identical in the two systems. Under this coupling, the completion moments are also identical in the two systems. □

6.5 Results

In this chapter, we give the first analysis of mean response time in the MSJ FCFS system. To do so, we reduce the problem to the analysis of mean response time in an M/M/1 with Markovian service rate (MMSR) in which the saturated system controls the service process (i.e. the At-least-k system). We call this reduction the RESET technique. Before applying the RESET technique, we start by analyzing the general MMSR-π system.

We prove the first explicit characterization of mean response time in the MMSR. To do so, we use our MARC technique, which is based on the novel concept of *relative completions* (See Section 6.4.8).

Theorem 6.5.1 (Mean response time asymptotics of MMSR systems)**.** *In the MMSR-π system, the expected response time in steady state satisfies*

$$\mathbb{E}[T^\pi] = \frac{1}{\lambda^*_\pi} \frac{1 + \Delta_\pi(Y^\pi_d, Y^\pi)}{1 - \lambda/\lambda^*_\pi} + O_\lambda(1).$$

To understand this formula, first note that the dominant term has order $\Theta(\frac{1}{1-\lambda/\lambda^*_\pi})$. This is the equivalent of the $\Theta(\frac{1}{1-\rho})$ behavior seen in simpler systems such as the M/G/1/FCFS. Next, to understand the numerator, examine the $\Delta_\pi(Y^\pi_d, Y^\pi)$ term. Δ_π, the relative completions function, smooths out the irregularities in completion times, so that the function $q - \Delta_\pi(y)$ has a constant negative drift. Δ_π is the analog of the remaining size of the job in service in the M/G/1. When a generic job arrives, it sees a time-average state of the service process, namely Y^π. When it departs, it leaves behind a departure-average state of the service process, namely Y^π_d. The difference in relative completions between these states captures the asymptotic behavior of mean response time. The overall numerator, $1 + \Delta_\pi(Y^\pi_d, Y^\pi)$, is analogous to the $\mathbb{E}[S_e]$ term in the M/G/1/FCFS mean response time formula.

Now that we have characterized the mean response time of the MMSR system, we can use this result to characterize the MSJ FCFS system. With our RESET technique, we show that the MSJ FCFS system has the same mean response time, up to an $O_\lambda(1)$ term, as the MMSR system whose service rate is controlled by the saturated system, or equivalently the At-least-k system.

Theorem 6.5.2 (Mean response time asymptotics of MSJ systems)**.** *In the multiserver-job system, the expected response time in steady state satisfies*

$$\mathbb{E}[T^{\text{MSJ}}] = \frac{1}{\lambda^*} \frac{1 + \Delta_{\text{Sat}}(Y^{\text{Sat}}_d, Y^{\text{Sat}})}{1 - \lambda/\lambda^*} + O_\lambda(1). \tag{6.3}$$

Empirically, the $O_\lambda(1)$ term is very small, as seen in Fig. 6.4a in Section 6.11.

Rather than calculating $\Delta_{\text{Sat}}(Y_d^{\text{Sat}}, Y^{\text{Sat}})$ in Theorem 6.5.2, we can calculate the equivalent value in the simplified saturated system (SSS) (See Section 6.4.11). Define $\Delta_{\text{SSS}}, Y_d^{\text{SSS}}$, and Y^{SSS} analogously to the primary saturated system.

Corollary 6.5.1. *In the MSJ FCFS model,*

$$\mathbb{E}[T^{\text{MSJ}}] = \frac{1}{\lambda^*}\frac{1+\Delta_{\text{SSS}}(Y_d^{\text{SSS}}, Y^{\text{SSS}})}{1-\lambda/\lambda^*} + O_\lambda(1).$$

This holds because $\Delta_{\text{Sat}}(y_1, y_2)$ is defined based on the completion times in the primary saturated system, and by Lemma 6.4.2, the SSS can be coupled to have the same completion times as the primary saturated system.

6.5.1 Example for demonstration

We now demonstrate applying Theorem 6.5.2 and Corollary 6.5.1 to characterize the asymptotic mean response time of our running example from Section 6.4.2.

We start with the MSJ system. First, we convert to the Ak system, whose front has state space $\mathbb{Y}^{\text{Ak}} = \{[1,1],[1,2],[2,1],[2,2]\}$. By the RESET technique, this only increases mean response time by $O_\lambda(1)$. As discussed in Section 6.4.6, the Ak system is identical to a MMSR-Sat system. As discussed in Section 6.4.11, the Sat system is equivalent to Simplified Saturated System (SSS), which has state space $\mathbb{Y}^{\text{SSS}} = \{[1,1],[1,2],[2]\}$.

For the rest of this section, we focus on the SSS, leaving the superscript implicit. Transitions between these states only happen as a result of completions, leading to the following transition rates:

$$\mu_{[1,1],[1,1],1} = 2\cdot\frac{2}{3} = \frac{4}{3}, \quad \mu_{[1,1],[1,2],1} = 2\cdot\frac{1}{3} = \frac{2}{3}, \quad \mu_{[1,2],[2],1} = 1,$$
$$\mu_{[2],[1,1],1} = \frac{1}{2}\frac{2}{3}\frac{2}{3} = \frac{2}{9}, \quad \mu_{[2],[1,2],1} = \frac{1}{2}\frac{2}{3}\frac{1}{3} = \frac{1}{9}, \quad \mu_{[2],[2],1} = \frac{1}{2}\frac{1}{3} = \frac{1}{6}.$$

Now, we can calculate the steady states Y^{SSS} and Y_d^{SSS} of the SSS's CTMC and DTMC respectively, and calculate the throughput $X^{\text{SSS}} = X^{\text{Sat}} = \lambda^*$. The vectors are in the order $\{[1,1],[1,2],[2]\}$:

$$Y = \left[\frac{1}{5}, \frac{1}{5}, \frac{3}{5}\right], \quad Y^d = \left[\frac{4}{9}, \frac{2}{9}, \frac{1}{3}\right], \quad X^{\text{SSS}} = X^{\text{Sat}} = \lambda^* = \frac{9}{10}.$$

Now, we can solve for $\Delta(y)$, defined in (6.1). To do so, we split up the completions $\mathbb{E}[C(y,t)]$ into the time until the first completion, and the time after the first completion. For example, starting in state $y = [1,1]$, the first completion takes an expected $\frac{1}{2}$ second, during which 1 completion occurs, compared to the long-term average rate $\frac{1}{2}\lambda^* = \frac{9}{20}$ completions. The system then transitions to a new state, with corresponding $\Delta(y)$. This gives rise to the following equation:

$$\Delta([1,1]) = 1 - \frac{9}{20} + \frac{2}{3}\Delta([1,1]) + \frac{1}{3}\Delta([1,2])$$

We use the same process to derive a system of equations that uniquely determines $\Delta(y)$, given in Corollary 6.9.1. We solve for $\Delta(y)$ for each state y:

$$\Delta([1,1]) = 1.38, \quad \Delta([1,2]) = -0.27, \quad \Delta([2]) = -0.37. \tag{6.4}$$

All decimals are exact. We can then average over the distribution Y^d to find that $\Delta(Y^d) = 0.43$. Recall that $\Delta(Y^d)$ is just shorthand for $\Delta(Y^d, Y)$.

We can therefore apply Theorem 6.5.2 and Corollary 6.5.1 to characterize the asymptotic mean response time of the original system:

$$\mathbb{E}[T^{\text{MSJ}}] = \frac{10}{9}\frac{1.43}{1 - \frac{\lambda}{9/10}} + O_\lambda(1).$$

6.6 MARC Proofs

We start by analyzing the M/M/1 with Markovian service rate π (MMSR-π). Our main result in this section is the proof of Theorem 6.5.1, a characterization of the asymptotic mean response time of the MMSR-π system.

The main challenge is choosing an appropriate test function $f(q, y)$, to leverage (6.2), the fact that $\mathbb{E}[G^\pi \circ f(Q^\pi, Y^\pi)] = 0$, to give an expression for $\mathbb{E}[Q^\pi]$. To gain information about $\mathbb{E}[Q^\pi]$ via this approach, it is natural to choose a function f which is quadratic in q, because G^π is effectively a derivative. However, if we choose $f_1(q, y) = \frac{1}{2}q^2$, the expression $G^\pi \circ f_1(q, y)$ will have cross-terms in which both q and y appear, preventing further progress.

Instead, our key idea is to use relative completions Δ_π in our test function:

Definition 6.6.1. *Let* $f^\pi_\Delta(q, y) = \frac{1}{2}(q - \Delta_\pi(y))^2$.

The $\Delta_\pi(y)$ term smooths out the fluctuations in the system's service rate, so that the quantity $q - \Delta_\pi(y)$ has a constant drift of $-\lambda^*_\pi$ whenever $q > 0$.

This choice of test function ensures that $G^\pi \circ f^\pi_\Delta(q, y)$ separates into a linear term dependent only on q and a term dependent only on y. The separation allows us to characterize $\mathbb{E}[Q^\pi]$, and hence $\mathbb{E}[T^\pi]$, in Theorem 6.5.1.

Let $u = \mathbb{1}\{q = 0 \wedge a = 1\}$ denote the unused service caused by a given transition. Only completion transitions ($a = 1$) can cause unused service.

We start by decomposing $G^\pi \circ f^\pi_\Delta(q, y)$, into a term linearly dependent on q, and terms dependent only on y, a, and u:

Lemma 6.6.1. *For any state* (q, y) *of the MMSR-*π *system,*

$$\begin{aligned} G^\pi \circ f^\pi_\Delta(q, y) &= (\lambda - \lambda^*_\pi)q \\ &\quad - \lambda\Delta_\pi(y) + \frac{1}{2}\lambda + \sum_{y',a} \mu^\pi_{y,y',a}\left(\frac{1}{2}(-a + u - \Delta_\pi(y'))^2 - \frac{1}{2}\Delta_\pi(y)^2\right). \end{aligned} \tag{6.5}$$

Proof. In this proof we omit π in the subscript of $\Delta_\pi(y)$ and in the superscript of $\mu^\pi_{y,y',a}$ for readability.

To calculate $G^\pi \circ f^\pi_\Delta(q,y)$, we begin by applying Lemma 6.9.5:

$$G^\pi \circ f^\pi_\Delta(q,y) = \lambda(q - \Delta(y) + \frac{1}{2}) \tag{6.6}$$

$$+ \sum_{y',a} \mu_{y,y',a} \left(\frac{1}{2} \left((q-a)^+ - \Delta(y') \right)^2 - \frac{1}{2} \left(q - \Delta(y) \right)^2 \right) \tag{6.7}$$

Recall that the unused service $u = \mathbb{1}\{q = 0 \wedge a = 1\}$, so $(q-a)^+ = q - a + u$. We can decompose (6.7) into two terms, with and without q:

$$(6.7) = q \sum_{y',a} \mu_{y,y',a} \left(-a + u - \Delta(y') + \Delta(y) \right) \tag{6.8}$$

$$+ \sum_{y',a} \mu_{y,y',a} \left(\frac{1}{2}(-a + u - \Delta(y'))^2 - \frac{1}{2}\Delta(y)^2 \right).$$

The coefficient of q in (6.8) can be simplified considerably using Lemma 6.9.4.

$$\begin{aligned}
&\sum_{y',a} \mu_{y,y',a} \left(-a + u - \Delta(y') + \Delta(y) \right) \\
&= \sum_{y',a} \mu_{y,y',a}(-a) + \sum_{y',a} \mu_{y,y',a} u - \sum_{y',a} \mu_{y,y',a}(\Delta(y') - \Delta(y)) \\
&= -\mu_{y,\cdot,1} - G^{\text{Ak}} \circ \Delta(y) + \sum_{y',a} \mu_{y,y',a} u \\
&= -\mu_{y,\cdot,1} - (\lambda^*_\pi - \mu^{\text{Ak}}_{y,\cdot,1}) + \sum_{y',a} \mu_{y,y',a} u \\
&= -\lambda^*_\pi + \sum_{y',a} \mu_{y,y',a} u.
\end{aligned}$$

Note that either $u = 0$ or $q = 0$, because new jobs are only generated if the queue is empty. As a result, $qu = 0$. We can therefore further simplify the q-term in (6.8):

$$q(\sum_{y',a} \mu_{y,y',a} u - \lambda^*_\pi) = -q\lambda^*_\pi \tag{6.9}$$

Substituting (6.9) into (6.8), (6.8) into (6.7), and performing some rearrangement, we find that

$$G^\pi \circ f^\pi_\Delta(q,y) = (\lambda - \lambda^*_\pi)q - \lambda\Delta(y) + \frac{1}{2}\lambda$$

$$+ \sum_{y',a} \mu_{y,y',a} \left(\frac{1}{2}(-a + u - \Delta(y'))^2 - \frac{1}{2}\Delta(y)^2 \right). \qquad \square$$

We can now characterize the mean response time of the MMSR-π system. To do so, we define three functions related to (6.5), the y-dependent portion of $G^\pi \circ f^\pi_\Delta(q,y)$:

Definition 6.6.2. *Define* $c_0(y,q), c_1(y)$, *and* $c_2(y)$ *as follows:*

$$\begin{aligned}
c_0(y,q) &= G^\pi \circ f^\pi_\Delta(q,y) - (\lambda - \lambda^*_\pi)q \\
&= -\lambda\Delta_\pi(y) + \frac{1}{2}\lambda + \sum_{y',a} \mu^\pi_{y,y',a}\left(\frac{1}{2}(-a+u-\Delta_\pi(y'))^2 - \frac{1}{2}\Delta_\pi(y)^2\right), \\
c_1(y) &= -\lambda\Delta_\pi(y) + \frac{1}{2}\lambda + \sum_{y',a} \mu^\pi_{y,y',a}\left(\frac{1}{2}(-a-\Delta_\pi(y'))^2 - \frac{1}{2}\Delta_\pi(y)^2\right), \\
c_2(y) &= c_1(y) - G^\pi \circ h(y), \text{ where } h(y) = \frac{1}{2}\Delta_\pi(y)^2 \\
&= -\lambda\Delta_\pi(y) + \frac{1}{2}\lambda + \sum_{y',a} \mu^\pi_{y,y',a}\left(\frac{1}{2}a^2 + a\Delta_\pi(y')\right).
\end{aligned}$$

We will show that the expected values of $c_0(y,q), c_1(y)$, and $c_2(y)$ in steady state are all equal up to a $O_\lambda(1-\frac{\lambda}{\lambda^*_\pi})$ error, which is crucial to our proof.

Theorem 6.5.1 (Mean response time asymptotics of MMSR systems). In the MMSR-π system, the expected response time in steady state satisfies

$$\mathbb{E}[T^\pi] = \frac{1}{\lambda^*_\pi}\frac{1+\Delta_\pi(Y^\pi_d, Y^\pi)}{1-\lambda/\lambda^*_\pi} + O_\lambda(1). \tag{6.10}$$

Proof. In this proof we omit π in the subscript of $\Delta_\pi(y)$ and in the superscript of $\mu^\pi_{y,y',a}$. We start from Lemma 6.6.1, which states that $G^\pi \circ f(q,y) = (\lambda - \lambda^*_\pi)q + c_0(y,q)$, so by (6.2), $0 = (\lambda - \lambda^*_\pi)\mathbb{E}[Q^\pi] + \mathbb{E}[c_0(Y^\pi, Q^\pi)]$, which implies that $\mathbb{E}[Q^\pi] = \mathbb{E}[c_0(Y^\pi, Q^\pi)]/(\lambda^*_\pi - \lambda)$. We therefore focus on $c_0(q,y)$: By characterizing $\mathbb{E}[c_0(Y^\pi, Q^\pi)]$, we will characterize $\mathbb{E}[Q^\pi]$.

Let us expand $c_0(y,q)$ and separate out the terms where u appears:

$$c_0(y,q) - c_1(y) = \sum_{y',a} \mu_{y,y',a} u\left(\frac{1}{2}u - a - \Delta(y')\right). \tag{6.11}$$

Note that in the time-average steady state Y^π, the fraction of service-process completions that occur while the queue is empty (i.e. where $u=1$) is $1-\frac{\lambda}{\lambda^*_\pi}$, because λ jobs arrive per second, and λ^* service-process completions occur per second. As a result, $E_{y\sim Y^\pi}\left[\sum_{y',a}\mu_{y,y',a}u\right] = 1 - \frac{\lambda}{\lambda^*_\pi}$.

Note that $a \le 1$ and $u \le 1$, because at most 1 job completes at a time. Note that $\Delta(y')$ is bounded by a constant over all y', because $y' \in \mathbb{Y}^\pi$, which is a finite state space. Thus, the $u/2 - a - \Delta(y')$ term in (6.11) is bounded by a constant. As a result, (6.11) contributes $O_\lambda(1-\frac{\lambda}{\lambda^*_\pi})$ to $\mathbb{E}[c_0(Y^\pi, Q^\pi)]$:

$$\mathbb{E}[c_1(Y^\pi)] = \mathbb{E}[c_0(Y^\pi, Q^\pi)] + O_\lambda(1 - \lambda/\lambda^*_\pi).$$

By (6.2), $\mathbb{E}[G^\pi \circ h(Y^\pi)] = 0$, so $\mathbb{E}[c_2(Y^\pi)] = \mathbb{E}[c_1(Y^\pi)]$. Let us now simplify $c_2(y)$, using the fact that $a = 0$ or 1:

$$c_2(y) = -\lambda\Delta(y) + \frac{1}{2}\lambda + \frac{1}{2}\mu_{y,\cdot,1} + \sum_{y'} \mu_{y,y',1}\Delta(y').$$

By Lemma 6.9.7, the last term in $c_2(y)$ is closely related to Y_d^π. Taking expectations over $y \sim Y^\pi$ and applying Lemma 6.9.7,

$$\mathbb{E}[\sum_{y'} \mu_{Y^\pi,y',1}\Delta(y')] = \lambda_\pi^* \sum_{y'} \mathbb{P}\left\{Y_d^\pi = y'\right\}\Delta(y') = \lambda_\pi^*\Delta(Y_d^\pi),$$

$$\mathbb{E}[c_2(Y^\pi)] = \mathbb{E}[-\lambda\Delta(Y^\pi) + \frac{1}{2}(\mu_{Y^\pi,\cdot,1} + \lambda) + \lambda_\pi^*\Delta(Y_d^\pi)].$$

Now note that $\mathbb{E}[\Delta(Y^\pi)] = 0$, $\mathbb{E}[\mu_{Y^\pi,\cdot,1}] = \lambda_\pi^*$, and $\lambda = \lambda_\pi^* + O_\lambda(1 - \frac{\lambda}{\lambda_\pi^*})$:

$$\mathbb{E}[c_1(Y^\pi)] = \mathbb{E}[c_2(Y^\pi)] = \lambda_\pi^* + \lambda_\pi^*\Delta(Y_d^\pi) + O_\lambda(1 - \frac{\lambda}{\lambda_\pi^*}). \tag{6.12}$$

$$\mathbb{E}[Q^\pi] = \frac{\mathbb{E}[c_0(Y^\pi, Q^\pi)]}{\lambda_\pi^* - \lambda^*} = \frac{\lambda_\pi^* + \lambda_\pi^*\Delta(Y_d^\pi)}{\lambda_\pi^* - \lambda} + O_\lambda(1) = \frac{\Delta(Y_d^\pi) + 1}{1 - \lambda/\lambda_\pi^*} + O_\lambda(1).$$

Now, we apply Little's Law, which states that $\mathbb{E}[T^\pi] = \frac{1}{\lambda}\mathbb{E}[Q^\pi]$.

$$\mathbb{E}[T^\pi] = \frac{1}{\lambda}\frac{1 + \Delta(Y_d^\pi)}{1 - \lambda/\lambda_\pi^*} + O_\lambda\left(\frac{1}{\lambda}\right) = \frac{1}{\lambda_\pi^*}\frac{1 + \Delta(Y_d^{\text{Sat}})}{1 - \lambda/\lambda_\pi^*} + O_\lambda\left(\frac{1}{\lambda}\right).$$

For the second equality, note that for any x, $\frac{1}{\lambda}\frac{x}{1-\lambda/\lambda^*} = \frac{1}{\lambda^*}\frac{x}{1-\lambda/\lambda^*} + \frac{x}{\lambda}$, which is absorbed by the $O_\lambda(\frac{1}{\lambda})$ term. Note that in the $\lambda \to \lambda_\pi^*$ limit, $O_\lambda(\frac{1}{\lambda}) = O_\lambda(1)$. Consider the $\lambda \to 0$ limit: $\mathbb{E}[T^\pi]$ is bounded for small λ. Likewise, $\frac{1}{\lambda_\pi^*}\frac{1+\Delta(Y_d^\pi)}{1-\lambda/\lambda_\pi^*}$ is bounded for small λ. As a result, the two differ by $O_\lambda(1)$:

$$\mathbb{E}[T^{\text{MSJ}}] = \frac{1}{\lambda_\pi^*}\frac{1 + \Delta(Y_d^\pi)}{1 - \lambda/\lambda_\pi^*} + O_\lambda(1). \quad \square$$

6.7 RESET Proofs

To characterize the asymptotic behavior of mean response time of the MSJ system, we use the At-least-k (Ak) system, which is stochastically equal to the MMSR-Sat system. The MARC results from Section 6.6 allow us to characterize the MMSR-Sat system. To prove that the MSJ FCFS and Ak systems have the same asymptotic mean response time behavior, our key idea is to show that Y^{MSJ} and Y^{Ak}, the steady states of their fronts, are "almost identical."

To formalize and prove the relationship between Y^{MSJ} and Y^{Ak}, we design a coupling in Section 6.7.1 between the MSJ system and the Ak system. We use a renewal-reward argument based on busy periods to prove Lemma 6.7.5, which states that under the coupling, $\mathbb{P}\left\{Y^{\text{MSJ}} \neq Y^{\text{Ak}}\right\} = O_\lambda(1 - \frac{\lambda}{\lambda^*})$.

Then, in Section 6.7.2, we combine Theorem 6.5.1 and Lemma 6.7.5 to prove Theorem 6.5.2, our main result, in which we give the first analysis of the asymptotic mean response time in the MSJ system, by reduction to the saturated system. Theorem 6.5.2 parallels the proof steps that Theorem 6.5.1 uses to characterize the MMSR system, using Lemma 6.7.5 to prove that the equivalent proof steps hold for the MSJ system.

We will make use of a test function $f_\Delta^{\text{MSJ}}(q, y)$ for the multiserver-job system which is similar to $f_\Delta^\pi(q, y)$, which was defined in Definition 6.6.1.

Definition 6.7.1. *For states* $y \in \mathbb{Y}^{\text{Ak}}$, $f_\Delta^{\text{MSJ}}(q,y) := f_\Delta^{\text{Ak}}(q,y) = f_\Delta^{\text{Sat}}(q,y)$. *Otherwise,* $f_\Delta^{\text{MSJ}}(q,y) := 0$.

Importantly, $G^{\text{MSJ}} \circ f_\Delta^{\text{MSJ}}(q,y)$ is similar to $G^{\text{Ak}} \circ f_\Delta^{\text{Ak}}(q,y)$:

Lemma 6.7.1. $G^{\text{MSJ}} \circ f_\Delta^{\text{MSJ}}(q,y) = \mathbb{1}_{q>0} G^{\text{Ak}} \circ f_\Delta^{\text{Ak}}(q,y) + \mathbb{1}_{q=0} O_\lambda(1)$.

Proof deferred to Section 6.9.1.

6.7.1 Coupling between At-least-k and MSJ

To show that the Ak system and the MSJ system have identical asymptotic mean response time, we define the following coupling of the two systems. We let the arrivals of the two systems happen at the same time. We couple the transitions of their front states based on their joint state $(q^{\text{MSJ}}, y^{\text{MSJ}}, q^{\text{Ak}}, y^{\text{Ak}})$. If $y^{\text{MSJ}} = y^{\text{Ak}}$, $q^{\text{MSJ}} > 0$, and $q^{\text{Ak}} > 0$, the completions happen at the same time in both systems, the same jobs complete, the same job phase transitions occur, and the jobs entering the front are the same. We call the two systems "merged" during such a time period. Note that under this coupling, if the two systems become merged, they will stay merged until $q^{\text{MSJ}} = 0$ or $q^{\text{Ak}} = 0$. If the systems are not merged, the two systems have independent completions and phase transitions, and independently sampled jobs.

The two systems transition according to synchronized Poisson timers whenever they are merged, and independent Poisson timers otherwise. Because all transitions are exponentially distributed, this poses no obstacle to the coupling.

We want to show that under this coupling, the two systems spend almost all of their time merged, in the limit as $\lambda \to \lambda^*$. Specifically, we will show that the fraction of time in which the two systems are *unmerged* is $O_\lambda(1 - \frac{\lambda}{\lambda^*})$. This implies Lemma 6.7.5, which is the key lemma we need for our main RESET result, Theorem 6.5.2.

To prove Lemma 6.7.5, we prove two key lemmas:

- Lemma 6.7.2: Whenever the two systems are unmerged, the expected time until the systems become merged is $O_\lambda(1)$.
- Lemma 6.7.3: Whenever the two systems are merged, the expected time for which they stay merged is $\Omega_\lambda(\frac{1}{1-\lambda/\lambda^*})$.

We then use a renewal-reward approach to prove Lemma 6.7.5.

Lemma 6.7.2 (Quick merge)**.** *From any joint MSJ, Ak state, for any* $\epsilon > 0$*, under the coupling above, the expected time until* $y^{\text{MSJ}} = y^{\text{Ak}}$, $q^{\text{MSJ}} \geq k+1$, *and* $q^{\text{Ak}} \geq k+1$ *is at most* $m_1(\epsilon)$ *for some* $m_1(\epsilon)$ *independent of the arrival rate* λ *and initial joint states, given that* $\lambda \in [\epsilon, \lambda^*)$.

Proof. We call the period of time until $y^{\text{MSJ}} = y^{\text{Ak}}$, $q^{\text{MSJ}} \geq k+1$, and $q^{\text{Ak}} \geq k+1$ the "bad period." We wish to show that the expected length of the bad period is upper bounded by some constant $m_1(\epsilon)$ for all λ such that $\lambda \in [\epsilon, \lambda^*)$.

Consider the possibility that the following sequence of events occurs: over a period of 1/2 second, at least $2k+1$ jobs arrive. Then, over another 1/2 second, k completions occur in each of the MSJ and At-least-k systems, which is sufficient to clear out every job initially present in the fronts and replace them with freshly sampled jobs. Finally, the sampled jobs in the fronts

of the two systems are the same, in the same order. After this sequence of events, $y^{\text{MSJ}} = y^{\text{Ak}}$, $q^{\text{MSJ}} \geq k+1$, and $q^{\text{Ak}} \geq k+1$, which ends the bad period.

Recall that as long as the front states of the two systems are distinct, their completions are independent. As a result, the probability of this sequence of events is positive, for any $\lambda > 0$ and for any initial states $y^{\text{MSJ}}, y^{\text{Ak}}$. We call the probability of this sequence of events $\text{pGood}(\lambda, y^{\text{MSJ}}, y^{\text{Ak}})$.

Moreover, $\text{pGood}(\lambda, y^{\text{MSJ}}, y^{\text{Ak}})$ is monotonically increasing in λ, as λ only affects the probability that at least $2k+1$ jobs arrive in the first half second.

Therefore, the least value of $\text{pGood}(\lambda, y^{\text{MSJ}}, y^{\text{Ak}})$ is achieved when $\lambda = \epsilon$. Because there are only finitely many possible front states $y^{\text{MSJ}} \in \mathbb{Y}^{\text{MSJ}}, y^{\text{Ak}} \in \mathbb{Y}^{\text{Ak}}$, there must be some lowest value of $\text{pGood}(\epsilon, y^{\text{MSJ}}, y^{\text{Ak}})$. We call this value $\text{pGood}^*(\epsilon)$. Note that for all $\lambda \geq \epsilon$ and for all $y^{\text{MSJ}} \in \mathbb{Y}^{\text{MSJ}}, y^{\text{Ak}} \in \mathbb{Y}^{\text{Ak}}$,

$$\text{pGood}(\lambda, y^{\text{MSJ}}, y^{\text{Ak}}) \geq \text{pGood}^*(\epsilon) > 0. \tag{6.13}$$

In the first second, there is at least a $\text{pGood}^*(\epsilon)$ chance of the desired sequence of events happening and the bad period completing. In the next second, the same is true. In general, the time until the bad period completes is upper bounded by a geometric distribution with completion probability $\text{pGood}^*(\epsilon)$. Taking $m_1(\epsilon) = 1/\text{pGood}^*(\epsilon)$, the mean time until the bad period completes is upper bounded by $m_1(\epsilon)$, which is independent of the arrival rate λ and initial joint states, as desired. □

Lemma 6.7.3 (Long merged period)**.** *From any joint MSJ, Ak state such that $y^{\text{MSJ}} = y^{\text{Ak}}$, $q^{\text{MSJ}} \geq k+1$, and $q^{\text{Ak}} \geq k+1$, the expected time until $q^{\text{MSJ}} = 0$, $q^{\text{Ak}} = 0$, or $y^{\text{MSJ}} \neq y^{\text{Ak}}$, is at least $\frac{m_2}{1-\lambda/\lambda^*}$ for some m_2 independent of the arrival rate λ and initial joint states, given that $\lambda < \lambda^*$.*

Before we prove Lemma 6.7.3, we prove a lemma about busy periods in the At-least-k system:

Lemma 6.7.4. *In the At-least-k system, for all $\lambda < \lambda^*$*

$$\mathbb{E}[B^{\text{Ak}}] = \Omega_\lambda\left(\frac{1}{1-\lambda/\lambda^*}\right) \tag{6.14}$$

Proof. In this proof we omit Ak in the subscript of $\mu^{\text{Ak}}_{y,y',a}$ for readability.

To prove Lemma 6.7.4, it suffices to show that $\mathbb{P}\left\{Q^{\text{Ak}} = 0\right\} = O_\lambda(1 - \frac{\lambda}{\lambda^*})$, and that the non-busy periods (periods when $Q^{\text{Ak}} = 0$) have expected duration $\Omega_\lambda(1)$. The latter follows from the fact that all transitions have expected duration $\Omega_\lambda(1)$.

To prove the former, let $u(q^{\text{Ak}}, y^{\text{Ak}})$ be the rate at which new jobs are generated due to completions in a particular state $(q^{\text{Ak}}, y^{\text{Ak}})$ of the At-least-k system. Note that $u(q^{\text{Ak}}, y^{\text{Ak}})$ is positive only if $q^{\text{Ak}} = 0$. The time-average value of $u(q^{\text{Ak}}, y^{\text{Ak}})$ is the difference between the completion rate of the system and the Poisson arrival rate, because in steady state the total completion rate and total arrival rate must match. Thus,

$$\mathbb{E}[u(Q^{\text{Ak}}, Y^{\text{Ak}})] = \lambda^* - \lambda. \tag{6.15}$$

Note that $u(q, y) = \mu_{y,\cdot,1}\mathbb{1}_{\{q=0\}}$, so

$$\mathbb{E}[\mu_{Y^{\text{Ak}},\cdot,1}\mathbb{1}_{\{Q^{\text{Ak}}=0\}}] = \lambda^* - \lambda.$$

Note that

$$\mathbb{P}\left\{Q^{\text{Ak}}=0\right\}=\frac{\mathbb{E}[\mu_{Y^{\text{Ak}},\cdot,1}\mathbb{1}_{\{Q^{\text{Ak}}=0\}}]}{\mathbb{E}[\mu_{Y^{\text{Ak}},\cdot,1}|Q^{\text{Ak}}=0]}=\frac{\lambda^*-\lambda}{\mathbb{E}[\mu_{Y^{\text{Ak}},\cdot,1}|Q^{\text{Ak}}=0]} \tag{6.16}$$

It therefore suffices to show that there exists a constant $c>0$ not dependent on λ such that $\mathbb{E}[\mu_{Y^{\text{Ak}},\cdot,1}|Q^{\text{Ak}}=0]\geq c$.

From an arbitrary state y^{Ak} with $q^{\text{Ak}}=0$, the distribution of time until a completion next occurs does not depend on λ. Consider the probability of a completion happening in the next second, with no arrivals happening before that completion. This probability is nonzero, and only dependent only λ via the arrival process. The probability can be lower bounded away from zero by substituting a Poisson(λ^*) process instead. We can thus lower bound the completion rate over the next second with an empty queue away from 0. This therefore provides a lower bound on the completion rate conditional on the queue being empty, $\mathbb{E}[\mu_{Y^{\text{Ak}},\cdot,1}|Q^{\text{Ak}}=0]$, as desired. Calling that lower bound c, we have:

$$\mathbb{P}\left\{Q^{\text{Ak}}=0\right\}=\frac{\mathbb{E}[\mu_{Y^{\text{Ak}},\cdot,1}\mathbb{1}_{\{Q^{\text{Ak}}=0\}}]}{\mathbb{E}[\mu_{Y^{\text{Ak}},\cdot,1}|Q^{\text{Ak}}=0]}\leq\frac{\lambda^*-\lambda}{c}=O_\lambda(1-\frac{\lambda}{\lambda^*}). \tag{6.17}$$

This completes the proof. □

Now we are ready to prove Lemma 6.7.3.

Lemma 6.7.3 (Long merged period). From any joint MSJ, At-least-k state such that $y^{\text{MSJ}}=y^{\text{Ak}}$, $q^{\text{MSJ}}\geq k+1$, and $q^{\text{Ak}}\geq k+1$, the expected time until $q^{\text{MSJ}}=0$, $q^{\text{Ak}}=0$, or $y^{\text{MSJ}}\neq y^{\text{Ak}}$, is at least $\frac{m_2}{1-\lambda/\lambda^*}$ for some m_2 independent of the arrival rate λ and initial joint states, given that $\lambda<\lambda^*$.

Note that the time until $q^{\text{MSJ}}=0$ or $q^{\text{Ak}}=0$ is a lower bound on the time until $y^{\text{MSJ}}\neq y^{\text{Ak}}$.

Proof. In this proof we omit Ak in the subscript of $\mu^{\text{Ak}}_{y,y',a}$ for readability.

Let's call the period of interest the "good period." Note that throughout the good period, $y^{\text{MSJ}}=y^{\text{Ak}}$. Let us introduce a new lower-bounding MSJ system, M', beginning in a general state $y^{M'}=y^{\text{MSJ}}=y^{\text{Ak}}$ and beginning with $q^{M'}=k+1$. Let us define a coupling between M' and the original MSJ and At-least-k systems in the same synchronized/independent fashion defined at the start of Section 6.7.1. As a result, for all time until $q^{M'}=0$, $y^{M'}=y^{\text{MSJ}}=y^{\text{Ak}}$, and $q^{M'}\leq q^{\text{MSJ}}$, and $q^{M'}\leq q^{\text{Ak}}$. In particular, the duration until $q^{M'}=0$ is a lower bound on the length of the good period.

Let us set up a new coupled system, M''. The M'' system is an At-least-k system initialized in a specific front state distribution to be specified later and with $q^{M''}=1$. Let us define a coupling between the M' and M'' systems in the same synchronized/independent fashion defined at the start of Section 6.7.1. Note however that M'' is a new system, distinct from all of the previous systems.

Let $B^{M'}$ be the length of the first busy period of the M' system, which is the time in M' until $q^{M'}=0$; similarly, let $B^{M''}$ be the length of the first busy period of the M'' system. We want to

show that

$$\mathbb{E}[B^{M'}] \geq m_3 \mathbb{E}[B^{M''}] \tag{6.18}$$

$$\mathbb{E}[B^{M''}] \geq \frac{m_4}{1-\lambda/\lambda^*}. \tag{6.19}$$

for some positive numbers m_3 and m_4 independent of the arrival rate λ and the initial front state of the M' system $y^{M'}$.

We will choose the front state distribution of the M'' system in order to guarantee that (6.19) holds. To do so, we will make use of Lemma 6.7.4, which states that the At-least-k system has long busy periods:

$$\mathbb{E}[B^{\text{Ak}}] = \Omega_\lambda(\frac{1}{1-\lambda/\lambda^*}) \tag{6.20}$$

Let $Y^{\text{Ak}-BP}$ denote the long-term-average distribution of the front state in the At-least-k system at the start of a busy period. We let the initial state distribution of the M'' system be $y^{M''} \sim Y^{\text{Ak}-BP}$ and $q^{M''} = 1$. As a result, $\mathbb{E}[B^{M''}] = \mathbb{E}[B^{\text{Ak}}]$, the expected busy period length of the At-least-k system. By Lemma 6.7.4, we have $\mathbb{E}[B^{M''}] \geq \frac{m_4}{1-\lambda/\lambda^*}$ for some positive number m_4 independent of λ and $y^{M'}$.

Now, we wish to show (6.18): that the length of the first busy period in M', initialized in an arbitrary initial front state $y^{M'}$ and $q^{M'} = k+1$, is also long in expectation.

To prove this, let us introduce a very fast Poisson process with a rate μ^* given by

$$\mu^* = \lambda^* + \max_{y \in \mathbb{Y}^{\text{Ak}}} \sum_{y',a} \mu_{y,y',a}.$$

Note that μ^* is at least as fast as the transition rate of M'' in any state, and μ^* is independent of λ. Let us define a coupling between the Poisson(μ^*) process and the M'' system. Transitions in the M'' system only occur when the Poisson(μ^*) increment occurs, where with some probability sampled on each Poisson increment a transition happens, and otherwise no transition occurs. In state y, a transition happens with probability

$$\frac{\lambda + \sum_{y',a} \mu_{y,y',a}}{\mu^*}$$

Note that this probability is always less than 1, by the definition of μ^*.

To lower bound $\mathbb{E}[B^{M'}]$, the expected busy period length in the M' system, let us consider $\mathbb{E}[B^{M'} \mathbb{1}_{\{A_1 \wedge A_2\}}]$, where A_1 and A_2 are the following two events:

1. Event A_1: the first increment of the Poisson(μ^*) process takes at least 1 second.
2. Event A_2: during the first second M' has exactly k completions, after each of which the job entering the front of the M' system is sampled to have the same server need as the corresponding job of the M'' system, and then all the jobs transition to the same phase as in the M'' system. At the end of the first second, M' and M'' have identical front states y and queue lengths $q = 1$ after exactly k completions in the M' system.

First, note that

$$\mathbb{E}[B^{M'}] \geq \mathbb{E}[B^{M'} \mathbb{1}_{\{A_1 \wedge A_2\}}] = \mathbb{E}[B^{M'} | A_1 \wedge A_2] \mathbb{P}\{A_1 \wedge A_2\}. \tag{6.21}$$

Note that $\mathbb{P}\{A_1 \wedge A_2\}$ is lower bounded by a positive constant for every λ such that $\epsilon \leq \lambda \leq \lambda^*$, so we can focus on $\mathbb{E}[B^{M'} | A_1 \wedge A_2]$.

Note that if events A_1 and A_2 occur, the M' and M'' systems have the same busy period length, because after 1 second, the two systems have identical states. Specifically, both systems become empty at the same time, which is the first time after each is initialized when each becomes empty.

As a result,

$$\mathbb{E}[B^{M'} | A_1 \wedge A_2] = \mathbb{E}[B^{M''} | A_1 \wedge A_2] \tag{6.22}$$

Note that Event A_2 is conditionally independent of the behavior of the M'' system, given that Event A_1 occurs. As a result,

$$\mathbb{E}[B^{M''} | A_1 \wedge A_2] = \mathbb{E}[B^{M''} | A_1] \tag{6.23}$$

Notice that event A_1 is independent of the state of M''. Conditioning on the event A_1 merely increases the time of the first transition in M'', without altering the future updates of M'' at all. As a result,

$$\mathbb{E}[B^{M''} | A_1] \geq \mathbb{E}[B^{M''}] \tag{6.24}$$

Thus, $\mathbb{E}[B^{M'}]$ is lower bounded by a constant multiple of $\mathbb{E}[B^{M''}]$. Recall that by construction, $\mathbb{E}[B^{M''}] = \Omega_\lambda(\frac{1}{1-\lambda/\lambda^*})$. Combining (6.18) and (6.19) and letting $m_2 = m_3 m_4$, we get the desired lower bound on the expected length of the good period. □

Now we are ready to prove our main result of this section:

Lemma 6.7.5 (Tight coupling)**.** *In the MSJ system, for any $\lambda < \lambda^*$, we have the following two properties:*

1. *Property 1:* $\mathbb{P}\left\{Q^{\text{MSJ}} = 0\right\} = O_\lambda(1 - \frac{\lambda}{\lambda^*})$
2. *Property 2:* $\mathbb{P}\left\{Y^{\text{MSJ}} \neq Y^{\text{Ak}}\right\} = O_\lambda(1 - \frac{\lambda}{\lambda^*})$.

where property 2 holds under the coupling in Section 6.7.1.

Proof. Let $\epsilon = \frac{\lambda^*}{2}$. Note that if $\lambda < \epsilon$, the properties are trivial: $O_\lambda(1 - \frac{\lambda}{\lambda^*}) \equiv O_\lambda(1)$, and probabilities are bounded. Therefore, we will focus on the case where $\lambda \geq \epsilon$, where we can apply Lemmas 6.7.2 and 6.7.3.

Let us define a *good period* to begin when $Y^{\text{MSJ}}(t) = Y^{\text{Ak}}(t)$, $Q^{\text{MSJ}}(t) \geq k+1$ and $Q^{\text{Ak}}(t) \geq k+1$, and end when $Q^{\text{MSJ}}(t) = 0$ or $Q^{\text{Ak}}(t) = 0$. Let a *bad period* be the time between two good periods. Note that throughout a good period, the front states are merged ($Y^{\text{MSJ}}(t) = Y^{\text{Ak}}(t)$) and both queues are nonempty.

To bound the fraction of time that the joint system is in a good period, we introduce the concept of a "y^*-cycle." Let y^* be an arbitrary state in $\mathbb{Y}^{\text{Ak}}$. Let a y^*-cycle be a renewal cycle

whose renewal points are moments when a bad period begins, and $Y^{\text{MSJ}}(t) = Y^{\text{Ak}}(t) = y^*$, and $Q^{\text{MSJ}}(t) = Q^{\text{Ak}}(t) = 0$, for some designated state y^*. We will show that a y^*-cycle has finite mean time. Given that fact, we can apply renewal reward to derive the equations below:

$$\mathbb{P}\left\{Q^{\text{MSJ}} = 0\right\} = \frac{\mathbb{E}[Q^{\text{MSJ}}(t) = 0 \text{ time per } y^*\text{-cycle}]}{\mathbb{E}[\text{total time per } y^*\text{-cycle}]}, \tag{6.25}$$

$$\mathbb{P}\left\{Y^{\text{MSJ}} \neq Y^{\text{Ak}}\right\} = \frac{\mathbb{E}[Y^{\text{MSJ}}(t) \neq Y^{\text{Ak}}(t) \text{ time per } y^*\text{-cycle}]}{\mathbb{E}[\text{total time per } y^*\text{-cycle}]}. \tag{6.26}$$

Note that $Q^{\text{MSJ}}(t) = 0$ or $Y^{\text{MSJ}}(t) \neq Y^{\text{Ak}}(t)$ only during a bad period, so the two probabilities in (6.25) and (6.26) are both bounded by the fraction of time spent in bad periods. By Lemma 6.7.2 and Lemma 6.7.3, the expected length of a bad period is at most m_1 and the expected length of a good period is at least $\frac{m_2}{1-\lambda/\lambda^*}$, conditioned on any initial joint state. Let Z be a random variable denoting the number of good periods in a y^* cycle. Note that good and bad periods alternate.

$$\mathbb{E}[\text{total time per } y^*\text{-cycle}] \geq \frac{m_2}{1-\lambda/\lambda^*}\mathbb{E}[Z],$$
$$\mathbb{E}[\text{bad period time per } y^*\text{-cycle}] \leq m_1\mathbb{E}[Z].$$

If a y^*-cycle has finite mean time, then we also have $\mathbb{E}[Z] < \infty$ because each good period and bad period take a positive time. Plugging the above inequalities into (6.25) and (6.26), we derive Properties 1 and 2:

$$\mathbb{P}\left\{Q^{\text{MSJ}} = 0\right\} \leq \frac{m_1}{m_2}\left(1 - \frac{\lambda}{\lambda^*}\right), \qquad \mathbb{P}\left\{Y^{\text{MSJ}} \neq Y^{\text{Ak}}\right\} \leq \frac{m_1}{m_2}\left(1 - \frac{\lambda}{\lambda^*}\right).$$

It remains to show that a y^*-cycle has finite mean time. We first use a Lyapunov argument to show that the joint states of the two systems return to a bounded set in a finite mean time. Consider the Lyapunov function $f_\Delta^{\text{MSJ}}(q^{\text{MSJ}}, y^{\text{MSJ}}) + f_\Delta^{\text{Ak}}(q^{\text{Ak}}, y^{\text{Ak}})$. Its drift is $G^{\text{MSJ}} \circ f_\Delta^{\text{MSJ}}(q^{\text{MSJ}}, y^{\text{MSJ}}) + G^{\text{Ak}} \circ f_\Delta^{\text{Ak}}(q^{\text{Ak}}, y^{\text{Ak}})$. Applying Lemma 6.6.1 to the Ak system,

$$G^{\text{Ak}} \circ f_\Delta^{\text{Ak}}(q^{\text{Ak}}, y^{\text{Ak}}) = (\lambda - \lambda^*)q^{\text{Ak}} + c_0(y^{\text{Ak}}, q^{\text{Ak}}),$$

where $c_0(y, q)$ is defined in Definition 6.6.2. Note that $c_0(y, q)$ is a bounded function because $\Delta(y)$ is bounded, by Lemma 6.9.1. Let $c_{\max}^{\text{Ak}}$ be the maximum of $c_0(y, q)$. For all $y^{\text{Ak}}, q^{\text{Ak}}$, $G^{\text{Ak}} \circ f_\Delta^{\text{Ak}}(q^{\text{Ak}}, y^{\text{Ak}}) \leq (\lambda - \lambda^*)q^{\text{Ak}} + c_{\max}^{\text{Ak}}$. By similar reasoning, applying Lemma 6.7.1, there exists a $c_{\max}^{\text{MSJ}}$ such that

$$G^{\text{MSJ}} \circ f_\Delta^{\text{MSJ}}(q^{\text{MSJ}}, y^{\text{MSJ}}) \leq (\lambda - \lambda^*)q^{\text{Ak}} + c_{\max}^{\text{MSJ}}$$

Let $c_{\max} = \max(c_{\max}^{\text{Ak}}, c_{\max}^{\text{MSJ}})$. Consider any $q^{\text{Ak}} \geq \frac{2c_{\max}+1}{\lambda^*-\lambda}$. Then for any y^{Ak},

$$G^{\text{Ak}} \circ f_\Delta^{\text{Ak}}(q^{\text{Ak}}, y^{\text{Ak}}) \leq -c_{\max} - 1.$$

Similarly, $G^{\text{MSJ}} \circ f_\Delta^{\text{MSJ}}(q^{\text{MSJ}}, y^{\text{MSJ}}) \leq -c_{\max} - 1$ whenever $q^{\text{MSJ}} \geq \frac{2c_{\max}+1}{\lambda^*-\lambda}$.

Let $c_{\text{cap}} = \max\{\frac{2c_{\max}+1}{\lambda^*-\lambda}, k+1\}$. We define the bounded set $\mathbb{S}$ as

$$\mathbb{S} = \left\{(q^{\text{MSJ}}, q^{\text{Ak}}, y^{\text{MSJ}}, y^{\text{Ak}}) \colon q^{\text{MSJ}} \leq c_{\text{cap}}, q^{\text{Ak}} \leq c_{\text{cap}}\right\}.$$

By the calculation above, the drift of the Lyapunov function $f_\Delta^{\text{MSJ}}(q^{\text{MSJ}}, y^{\text{MSJ}}) + f_\Delta^{\text{Ak}}(q^{\text{Ak}}, y^{\text{Ak}})$ is at most -1 outside $\mathbb{S}$. In particular, outside of $\mathbb{S}$, either $q^{\text{MSJ}} > c_{\max}$ or $q^{\text{Ak}} > c_{\max}$, yielding a drift term $\leq -c_{\max} - 1$, outweighing the term where q is small. Thus, by the Foster-Lyapunov theorem [153, Theorem A.4.1], the system returns to $\mathbb{S}$ in finite mean time.

We call a period of time inside the bounded set $\mathbb{S}$ an $\mathbb{S}$-*visit*. Each $\mathbb{S}$-visit has a finite mean time because there is a positive probability of having a lot of arrivals in the next second and leaving $\mathbb{S}$. Moreover, as proved above using the Lyapunov argument, the time between two $\mathbb{S}$-visits has finite mean.

Each $\mathbb{S}$-visit has a positive probability of ending the y^*-cycle. To prove this, we construct a positive probability sample path of beginning a good period with $q^{\text{MSJ}} = q^{\text{Ak}}$ and ending the good period in $(0, 0, y^*, y^*)$, while remaining in $\mathbb{S}$.

- First, we have a lot of completions in the two systems, completely emptying both. $q^{\text{MSJ}} = q^{\text{Ak}} = |y^{\text{MSJ}}| = 0$. Next, k jobs arrive. Now $q^{\text{Ak}} = k$ and $q^{\text{MSJ}} = 0$. During this time $y^{\text{MSJ}} \neq y^{\text{Ak}}$.
- Then k jobs complete in the Ak system, no jobs complete in the MSJ system, and the newly generated Ak jobs are sampled such that $y^{\text{MSJ}} = y^{\text{Ak}}$, while $q^{\text{MSJ}} = q^{\text{Ak}} = 0$.
- Next, $k + 1$ jobs arrive, and a good period begins.
- Finally, $k + 1$ jobs complete in both systems, ending with $y^{\text{MSJ}} = y^{\text{Ak}} = y^*$, and $q^{\text{MSJ}} = q^{\text{Ak}} = 0$. Now a y^*-cycle ends, and the next begins.

All of these events have strictly positive probability and are independent of each other, so their joint occurrence has strictly positive probability as well. Thus, the length of a y^*-cycle is bounded by a geometric number of $\mathbb{S}$-visits, each of which has finite mean time, completing the proof. □

6.7.2 Proof of Theorem 6.5.2

We now are ready to prove our main theorem, Theorem 6.5.2, progressing along similar lines as Theorem 6.5.1 and making use of Lemmas 6.6.1 and 6.7.5. Throughout this section, whenever we make use of results from Section 6.6, we set $\pi = \text{Sat}$. In particular, we make use of $c_0(y, q)$ and $c_1(y)$, from Definition 6.6.2.

Theorem 6.5.2. In the multiserver-job system, the expected response time in steady state satisfies

$$\mathbb{E}[T^{\text{MSJ}}] = \frac{1}{\lambda^*} \frac{1 + \Delta(Y_d^{\text{Sat}}, Y^{\text{Sat}})}{1 - \lambda/\lambda^*} + O_\lambda(1)$$

Proof. We will show that the MSJ model has the same asymptotic mean response time as the Ak system. We will make use of the test function $f_\Delta^{\text{MSJ}}(q, y)$, from Definition 6.7.1. Recall from Lemma 6.7.1 that

$$G^{\text{MSJ}} \circ f_\Delta^{\text{MSJ}}(q, y) = G^{\text{Ak}} \circ f_\Delta^{\text{Ak}}(q, y) \mathbb{1}_{q>0} + \mathbb{1}_{q=0} O_\lambda(1)$$

We will next use (6.2), the fact that the expected value of a generator function in steady state is zero, which implies that

$$0 = \mathbb{E}[G^{\text{Ak}} \circ f_\Delta^{\text{Ak}}(Q^{\text{MSJ}}, Y^{\text{MSJ}}) \mathbb{1}\{Q^{\text{MSJ}} > 0\}] + \mathbb{P}\left\{Q^{\text{MSJ}} = 0\right\} O_\lambda(1) \tag{6.27}$$

By Lemma 6.7.5, $\mathbb{P}\left\{Q^{\text{MSJ}} = 0\right\} = O_\lambda(1 - \frac{\lambda}{\lambda^*})$. Next, we apply Lemma 6.6.1 to the Ak system, finding that $G^{\text{Ak}} \circ f_\Delta^{\text{Ak}}(q, y) = (\lambda - \lambda^*)q + c_0(y, q)$. Recall that $c_0(y, q)\mathbb{1}_{q>0} = c_1(y)\mathbb{1}_{q>0}$. Combining with (6.27) and invoking Lemmas 6.6.1 and 6.7.5 and the fact that $c_1(y)$ is bounded, we have

$$\begin{aligned}(\lambda - \lambda^*)\mathbb{E}[Q^{\text{MSJ}}] + \mathbb{E}[c_0(Y^{\text{MSJ}}, Q^{\text{MSJ}})\mathbb{1}\{Q^{\text{MSJ}} > 0\}] &= O_\lambda(1 - \lambda/\lambda^*)\\ (\lambda - \lambda^*)\mathbb{E}[Q^{\text{MSJ}}] + \mathbb{E}[c_1(Y^{\text{MSJ}})] &= O_\lambda(1 - \lambda/\lambda^*)\\ \mathbb{E}[Q^{\text{MSJ}}] = \frac{\mathbb{E}[c_1(Y^{\text{MSJ}}])}{\lambda^* - \lambda} &+ O_\lambda(1) \qquad (6.28)\end{aligned}$$

Next, specializing (6.12) in the proof of Theorem 6.5.1 to the Ak system, we know that $\mathbb{E}[c_1(Y^{\text{Ak}})] = \lambda^* + \lambda^*\Delta(Y_d^{\text{Sat}}) + O_\lambda(1 - \frac{\lambda}{\lambda^*})$. By Lemma 6.7.5, we know that $\mathbb{P}\left\{Y^{\text{Ak}} \neq Y^{\text{MSJ}}\right\} = O_\lambda(1 - \frac{\lambda}{\lambda^*})$. Again because $c_1(y)$ is bounded,

$$\mathbb{E}[c_1(Y^{\text{MSJ}})] = \mathbb{E}[c_1(Y^{\text{Ak}})] + O_\lambda\left(1 - \frac{\lambda}{\lambda^*}\right) = \lambda^* + \lambda^*\Delta(Y_d^{\text{Sat}}) + O_\lambda\left(1 - \frac{\lambda}{\lambda^*}\right).$$

Therefore, applying (6.28), we find that $\mathbb{E}[Q^{\text{MSJ}}] = \frac{1+\Delta(Y_d^{\text{Sat}})}{1-\lambda/\lambda^*} + O_\lambda(1)$.

Now, we apply Little's Law, which states that $\mathbb{E}[T^{\text{Ak}}] = \frac{1}{\lambda}\mathbb{E}[N^{\text{Ak}}]$. Note that Q^{Ak} and N^{Ak} differ by the number of jobs in the front, which is $O_\lambda(1)$.

$$\mathbb{E}[T^{\text{MSJ}}] = \frac{1}{\lambda}\frac{1 + \Delta(Y_d^{\text{Sat}})}{1 - \lambda/\lambda^*} + O_\lambda\left(\frac{1}{\lambda}\right) = \frac{1}{\lambda^*}\frac{1 + \Delta(Y_d^{\text{Sat}})}{1 - \lambda/\lambda^*} + O_\lambda\left(\frac{1}{\lambda}\right)$$

For the second equality, note that for any x, $\frac{1}{\lambda}\frac{x}{1-\lambda/\lambda^*} = \frac{1}{\lambda^*}\frac{x}{1-\lambda/\lambda^*} + \frac{x}{\lambda}$. Here x is a constant, so the extra term is absorbed by the $O_\lambda(1/\lambda)$.

By the same bounding argument as in the last step of Theorem 6.5.1,

$$\mathbb{E}[T^{\text{MSJ}}] = \frac{1}{\lambda^*}\frac{1 + \Delta(Y_d^{\text{Sat}})}{1 - \lambda/\lambda^*} + O_\lambda(1). \quad \square$$

6.8 Deferred lemmas

We now present proofs of several subsidiary lemmas which were deferred to improve the clarity of the presentation of the mean results.

6.9 Finiteness of Δ, and the conditions for drift lemma

Lemma 6.9.1. *The relative completion function*

$$\Delta_\pi(y_1, y_2) = \lim_{t\to\infty} \mathbb{E}[C_\pi(y_1, t) - C_\pi(y_2, t)]$$

is well-defined and finite for any pair of states y_1 and y_2 of the service process π.

Proof. Throughout this proof, we leave the subscript π implicit.

To characterize $\mathbb{E}[C(y_1, t) - C(y_2, t)]$, we construct a coupling between the two service processes π with initial states y_1 and y_2. We let each system transition independently when their states are different, and let them transition identically once their states become the same. Let τ be the time that the states of the two systems become the same. Because the two systems remain identical after τ, for any $t \geq 0$,

$$C(y_1, t) - C(y_2, y) = C(y_1, \min(t, \tau)) - C(y_2, \min(t, \tau)).$$

We assume that the system π is irreducible. Because each system is irreducible, the joint Markov chain of two systems is also irreducible and $\tau < \infty$ almost surely. Therefore,

$$\lim_{t\to\infty} \mathbb{E}[C(y_1, t) - C(y_2, t)] = \mathbb{E}[C(y_1, \tau) - C(y_2, \tau)].$$

The RHS of the above equality is clearly finite. □

Now we show that for any Markov chain η,

$$\mathbb{E}[G^\eta \circ f(Q^\eta, Y^\eta)] = 0. \tag{6.2}$$

The lemma below is implied by [74, Proposition 3]:

Lemma 6.4.1. Let f be a real-valued function of the state of a Markov chain η. Assume that the transition rates of the Markov chain η are uniformly bounded, and $\mathbb{E}[f(Q^\eta, Y^\eta)] < \infty$. Then

$$\mathbb{E}_{(q,y)\sim(Q^\eta, Y^\eta)}[G^\eta \circ f(q, y)] = 0. \tag{6.2}$$

To check that the conditions of Lemma 6.4.1 hold for the At-least-k and MSJ systems, first notice that their transitions rates are both uniformly bounded. In particular, the transition rates of the At-least-k system are uniformly bounded by $\lambda + \max_y \sum_{y',a} \mu^{\text{Ak}}_{y,y',a}$, and the transition rates of the MSJ system are uniformly bounded by $\lambda + \max_{y,b} \sum_{y',a} \mu^{\text{MSJ}}_{y,y',a,b}$. Therefore we only need to check that each f used in this chapter has finite steady-state expectations in At-least-k and MSJ systems, i.e.

$$\mathbb{E}[f(Q^{\text{Ak}}, Y^{\text{Ak}})] < \infty$$
$$\mathbb{E}[f(Q^{\text{MSJ}}, Y^{\text{MSJ}})] < \infty.$$

The following lemma shows that a function f has finite expectations in the At-least-k and MSJ system as long as it grows at a polynomial rate in q, which is true for all f which we will apply Lemma 6.4.1 to.

Lemma 6.9.2. *Consider the MMSR system controlled by the Markov chain π and the MSJ system. For any positive integer m,*

$$\mathbb{E}[(Q^\pi)^m] < \infty$$
$$\mathbb{E}[(Q^{\text{MSJ}})^m] < \infty.$$

To prove the lemma, we need [98, Theorem 2.3], restated as below:

Lemma 6.9.3. *Consider a Markov chain η and a Lyapunov function V that satisfies the conditions below: $V(q,y) \geq 0$; there exists a constant $b, \gamma > 0$ such that whenever $V(q,y) \geq b$,*

$$G^{\eta} \circ V(q,y) \leq -\gamma, \tag{6.29}$$

and there exists $d > 0$ such that

$$\max_{\text{next state } (q',y')} |V(q',y') - V(q,y)| \leq d, \tag{6.30}$$

then there exists $\theta > 0$ such that

$$\mathbb{E}[e^{\theta V(Q^{\eta}, Y^{\eta})}] < \infty. \tag{6.31}$$

Now we prove Lemma 6.9.2.

Proof of Lemma 6.9.2. We first prove the lemma for the MMSR system controlled by the Markov chain π.

Let $\Delta_{\max}$ be the maximal absolute value of $\Delta_{\pi}(y)$ for any y in the state spaces of $\mathbb{Y}^{\pi}$, which must be finite due to Lemma 6.9.1 and the fact that there are only finitely many possible y.

We construct the Lyapunov function $V(q,y) = (q - \Delta(y))^{+}$. We first check the conditions of Lemma 6.9.3 for the MMSR system controlled by π. To check (6.29), we let $b = 1 + 2\Delta_{\max}$ and $\gamma = \lambda^* - \lambda$. If $V(q,y) \geq b$, we must have $q \geq 1 + \Delta_{\max}$; for any state (q', y') that the system can jump to after one transition, $V(q',y') \geq q' - \Delta(y') \geq q - 1 - \Delta_{max} \geq 0$, so $V(q',y') = q' - \Delta(y')$. Therefore,

$$G^{\pi} \circ V(q,y) = G^{\pi} \circ (q - \Delta(y)) = \lambda - \lambda^* = -\gamma.$$

It is also easy to see that (6.30) holds with $d = 2\Delta_{\max}$. Therefore, by Lemma 6.9.3, there exists $\theta > 0$ such that

$$\mathbb{E}[e^{\theta V(Q^{\pi}, Y^{\pi})}] < \infty.$$

Observe that $e^{\theta V(q,y)}$ grows with q exponentially fast. Therefore, for any positive integer m,

$$\begin{aligned} q^m &= O(e^{\theta V(q,y)}), \\ \mathbb{E}[(Q^{\pi})^m] &< \infty. \end{aligned}$$

The analysis of the MSJ system is similar to the analysis of the At-least-k system, which is a special case of the MMSR system with $\pi = \text{Sat}$. We consider the Lyapunov function

$$V(q,y) = \begin{cases} \text{if } q > 1 & (q - \Delta_{\text{Sat}}(y))^{+} \\ \text{otherwise} & 0, \end{cases}$$

and check the conditions of Lemma 6.9.3. Notice that $G^{\text{MSJ}} \circ V(q,y) = G^{\text{Ak}} \circ V(q,y)$ for any $q \geq 1$, so the rest of the argument is verbatim. □

6.9.1 Lemmas about G^π and G^{MSJ}

Lemma 6.9.4. *Consider the MMSR system controlled by the Markov chain π. For any state $y \in \mathbb{Y}^\pi$,*

$$G^\pi \circ \Delta_\pi(y, Y^\pi) = \lambda_\pi^* - \mu_{y,\cdot,1}^\pi \tag{6.32}$$

Proof. Recall that by the definition of the generator, $G^\pi \circ \Delta_\pi(y, Y^\pi)$ is given by

$$G^\pi \circ \Delta_\pi(y, Y^\pi) = \lim_{t\to 0} \frac{1}{t}\mathbb{E}[\Delta_\pi(Y^\pi(t), Y^\pi) - \Delta_\pi(y, Y^\pi)|Y^\pi(0) = y]. \tag{6.33}$$

To figure out $\mathbb{E}[\Delta_\pi(Y^\pi(t), Y^\pi) - \Delta_\pi(y, Y^\pi)|Y^\pi(0) = y]$, recall the definition that

$$\Delta_\pi(y, Y^\pi) = \lim_{t'\to\infty} \mathbb{E}[C_\pi(y, t') - \lambda_\pi^* t'],$$

where recall that $C_\pi(y, t')$ is the expected number of completion up to time t' of the MMSR system whose service process is controlled by the Markov chain π initializing in state y. Therefore, if we replace y by $Y^\pi(t)$ on the LHS of the above definition and take the expectation, we have

$$\begin{aligned}
&\mathbb{E}[\Delta_\pi(Y^\pi(t), Y^\pi)|Y^\pi(0) = y] \\
&= \lim_{t'\to\infty} \mathbb{E}[C_\pi(Y^\pi(t), t') - \lambda_\pi^* t'|Y^\pi(0) = y] \\
&= \lim_{t'\to\infty} \mathbb{E}[C_\pi(y, t + t') - C_\pi(y, t) - \lambda_\pi^* t'|Y^\pi(0) = y],
\end{aligned}$$

where in the second equality we have used the fact that

$$\begin{aligned}
\mathbb{E}[C_\pi(y, t + t')] &= \mathbb{E}[C_\pi(y, t) + C_\pi([Y_\pi(t) \mid Y_\pi(0) = y], t')] \\
\mathbb{E}[C_\pi(Y_\pi(t), t') \mid Y_\pi(0) = y] &= \mathbb{E}[C_\pi(y, t + t')] - \mathbb{E}[C_\pi(y, t)].
\end{aligned} \tag{6.34}$$

(6.34) simply splits up the completions from time 0 to $t + t'$ into the completions from time 0 to t, and the completions from time t to $t + t'$.

Therefore,

$$\begin{aligned}
&\mathbb{E}[\Delta_\pi(Y^\pi(t), Y^\pi)) - \Delta_\pi(y, Y^\pi)|Y^\pi(0) = y] \\
&= \lim_{t'\to\infty} \mathbb{E}[C_\pi(y, t + t') - C_\pi(y, t) - \lambda_\pi^* t'] - \lim_{t'\to\infty} \mathbb{E}[C_\pi(y, t') - \lambda_\pi^* t'] \\
&= \lim_{t'\to\infty} \mathbb{E}[C_\pi(y, t + t') - C_\pi(y, t) - \lambda_\pi^* t'] - \lim_{t'\to\infty} \mathbb{E}[C_\pi(y, t + t') - \lambda_\pi^* t - \lambda_\pi^* t'] \\
&= \mathbb{E}[-C_\pi(y, t) + \lambda_\pi^* t],
\end{aligned}$$

where in the second inequality we replace t' with $t + t'$ in the second term, which will not change the limit because t' and $t + t'$ are both going to infinity. Plugging the above calculations into (6.33), we get

$$G^\pi \circ \Delta_\pi(y, Y^\pi) = \lim_{t\to 0} \frac{1}{t}\mathbb{E}[-C_\pi(y, t) + \lambda_\pi^* t] = -\mu_{y,\cdot,1}^\pi + \lambda_\pi^*,$$

where in the last inequality we use the fact that $\lim_{t\to 0} \frac{1}{t}\mathbb{E}[C_\pi(y, t)] = \mu_{y,\cdot,1}^\pi$ (the instantaneous completion rate at state y). □

Lemma 6.9.5. *For any $f(q, y)$ which is a real-valued function of the state of the MMSR-π system,*

$$G^\pi \circ f(q,y) = \lambda \left(f(q+1,y) - f(q,y)\right) + \sum_{\substack{y' \in \mathbb{Y}^\pi, \\ a \in \{0,1\}}} \mu^\pi_{y,y',a} \left(f((q-a)^+, y') - f(q,y)\right).$$

Proof. In this proof we omit π in the subscript of $\Delta_\pi(y)$ and in the superscript of $\mu^\pi_{y,y',a}$ for readability.

Recall the definition of the generator

$$G^\pi \circ f(q,y) = \lim_{t \to 0} \frac{1}{t} \mathbb{E}[f(Q^\pi(t), Y^\pi(t)) - f(q,y) | Q^\pi(0) = q, Y^\pi(0) = y],$$

which can be interpreted as the instantaneous rate of change of the function $f(Q^\pi(t), Y^\pi(t))$ when $(Q^\pi(t), Y^\pi(t))$ is initialized in (q, y). Note that $(Q^\pi(t), Y^\pi(t))$ can change either due to an arrival event, or a transition event of the Markov chain π. An arrival event happens with rate λ, and causes $Q^\pi(t)$ to change from q to $q+1$, so arrival events contribute

$$\lambda \left(f(q+1,y) - f(q,y)\right)$$

to $G^\pi \circ f(q, y)$. A transition event of the Markov chain π from y to $y' \in \mathbb{Y}^\pi$ accompanied by $a \in \{0, 1\}$ completions happens with rate $\mu_{y,y',a}$. Such a event causes $(Q^\pi(t), Y^\pi(t))$ to change from (q, y) to $((q-a)^+, y')$, so it contributes

$$\mu_{y,y',a} \left(f((q-a)^+, y') - f(q,y)\right)$$

to $G^\pi \circ f(q, y)$, for each $y' \in \mathbb{Y}^{\text{Ak}}$ and $a \in \{0, 1\}$. This proves the expression in the lemma statement. □

As a corollary of Lemma 6.9.4 and Lemma 6.9.5, we can derive a forward recurrence for $\Delta_\pi(y) := \Delta_\pi(y, Y^\pi)$. Solving the resulting system of equations, together with the fact that $\Delta_\pi(Y^\pi) = 0$, gives the value of $\Delta_\pi(y)$.

Corollary 6.9.1. *For any MMSR-π system and any state $y \in \mathbb{Y}^\pi$,*

$$\Delta_\pi(y) = \frac{\mu_{y,\cdot,1} - \lambda^*_\pi}{\mu_{y,\cdot,\cdot}} + \sum_{y',a} \frac{\mu_{y,y',a}}{\mu_{y,\cdot,\cdot}} \Delta(y'),$$

where $\mu_{y,\cdot,\cdot}$ is the total transition rate out of state y.

Moreover, if all transitions in π are associated with completions (if a always equals 1), then the recurrence simplifies:

$$\Delta_\pi(y) = 1 - \frac{\lambda^*_\pi}{\mu_{y,\cdot,1}} + \sum_{y'} \frac{\mu_{y,y',1}}{\mu_{y,\cdot,1}} \Delta(y').$$

Proof. Start with Lemma 6.9.4:

$$G^\pi \circ \Delta_\pi(y) = \lambda^*_\pi - \mu^\pi_{y,\cdot,1} \tag{6.35}$$

Here we write $\Delta_\pi(y)$ as a shorthand for $\Delta_\pi(y, Y^\pi)$.

Expand the left-hand side of (6.35) using Lemma 6.9.5:

$$G^\pi \circ \Delta_\pi(y) = \sum_{y',a} \mu^\pi_{y,y',a}(\Delta_\pi(y') - \Delta_\pi(y))$$

Note that Lemma 6.9.5 simplifies because $\Delta_\pi(y)$ does not depend on q.

Now we can perform algebraic manipulation to complete the proof:

$$\begin{aligned}
\lambda^*_\pi - \mu^\pi_{y,\cdot,1} &= \sum_{y',a} \mu^\pi_{y,y',a}(\Delta_\pi(y') - \Delta_\pi(y)) \\
&= -\mu^\pi_{y,\cdot,\cdot}\Delta_\pi(y) + \sum_{y',a} \mu^\pi_{y,y',a}\Delta_\pi(y') \\
\mu^\pi_{y,\cdot,\cdot}\Delta_\pi(y) &= \mu^\pi_{y,\cdot,1} - \lambda^*_\pi + \sum_{y',a} \mu^\pi_{y,y',a}\Delta_\pi(y') \\
\Delta_\pi(y) &= \frac{\mu^\pi_{y,\cdot,1} - \lambda^*_\pi}{\mu^\pi_{y,\cdot,\cdot}} + \sum_{y',a} \frac{\mu^\pi_{y,y',a}}{\mu^\pi_{y,\cdot,\cdot}}\Delta_\pi(y')
\end{aligned}$$

Note that if all transitions are associated with completions, e.g. if $a = 1$, then $\mu^\pi_{y,\cdot,1} = \mu^\pi_{y,\cdot,\cdot}$ □

Now, we focus on the MSJ system:

Lemma 6.9.6. *For any $f(q, y)$ which is a real-valued function of the state of the MSJ system,*

$$\begin{aligned}
G^{\text{MSJ}} \circ f(q,y) = {} & \lambda \left(f(q+1,y) - f(q,y)\right) \mathbb{1}_{\{y \in \mathbb{Y}^{\text{Ak}}\}} && (6.36)\\
& + \mathbb{1}_{q=0, y \notin \mathbb{Y}^{\text{Ak}}} \lambda \sum_{i \in S} p_i (f(0, y \cdot i) - f(0,y)) && (6.37)\\
& + \mathbb{1}_{q>0} \sum_{\substack{y' \in \mathbb{Y}^{\text{Ak}}, \\ a \in \{0,1\}}} \mu^{\text{Ak}}_{y,y',a} \left(f((q-a)^+, y') - f(q,y)\right). && (6.38)\\
& + \mathbb{1}_{q=0} \sum_{\substack{y' \in \mathbb{Y}^{\text{MSJ}}, \\ a \in \{0,1\}}} \mu^{\text{MSJ}}_{y,y',a,0} \left(f((q-a)^+, y') - f(q,y)\right). && (6.39)
\end{aligned}$$

Proof. Recall the definition of the generator

$$G^{\text{MSJ}} \circ f(q,y) = \lim_{t \to 0} \frac{1}{t} \mathbb{E}[f(Q^{\text{MSJ}}(t), Y^{\text{MSJ}}(t)) - f(q,y) | Q^{\text{MSJ}}(0) = q, Y^{\text{MSJ}}(0) = y],$$

which can be interpreted as the instantaneous rate of change of the function $f(Q^{\text{MSJ}}(t), Y^{\text{MSJ}}(t))$ when $(Q^{\text{MSJ}}(t), Y^{\text{MSJ}}(t))$ is initialized in (q, y). Note that $(Q^{\text{MSJ}}(t), Y^{\text{MSJ}}(t))$ can change either due to an arrival event, or a transition event of the front state. An arrival event happens with rate λ, and its effect depends on whether $y \in \mathbb{Y}^{\text{Ak}}$: if $y \in \mathbb{Y}^{\text{Ak}}$, there are k jobs in the front, so $Q^{\text{MSJ}}(t)$ changes from q to $q+1$, $Y^{\text{MSJ}}(t)$ remains unchanged; if $y \notin \mathbb{Y}^{\text{Ak}}$, there are strictly fewer than k jobs in the front, so $Q^{\text{MSJ}}(t)$ remains zero after the arrival, and $Y^{\text{MSJ}}(t)$ changes from y to $y \cdot i$

with probability p_i (append a fresh job in state i to the front state with probability p_i). Therefore, arrival events contribute

$$\lambda \left(f(q+1,y) - f(q,y)\right) \mathbb{1}_{\{y \in \mathbb{Y}^{\text{Ak}}\}} + \mathbb{1}_{q=0, y \notin \mathbb{Y}^{\text{Ak}}} \lambda \sum_{i \in S} p_i (f(0, y \cdot i) - f(0,y))$$

to $G^{\text{MSJ}} \circ f(q,y)$, which are the terms in (6.36) and (6.37) in the lemma statement. As for the transition events of the front, a transition from state y to state y' accompanied by a completions causes $(Q^{\text{MSJ}}(t), Y^{\text{MSJ}}(t))$ to change from (q,y) to $((q-a)^+, y')$. Such a transition happens with the rate $\mu^{\text{MSJ}}_{y,y',a,1} = \mu^{\text{Ak}}_{y,y',a}$ if $q > 0$ and $y' \in \mathbb{Y}^{\text{Ak}}$, and happens with rate $\mu^{\text{MSJ}}_{y,y',a,0}$ if $q = 0$. Therefore, the transition events of the front contribute

$$\mathbb{1}_{q>0} \sum_{\substack{y' \in \mathbb{Y}^{\text{Ak}}, \\ a \in \{0,1\}}} \mu^{\text{Ak}}_{y,y',a} \left(f((q-a)^+, y') - f(q,y)\right) + \mathbb{1}_{q=0} \sum_{\substack{y' \in \mathbb{Y}^{\text{MSJ}}, \\ a \in \{0,1\}}} \mu^{\text{MSJ}}_{y,y',a,0} \left(f((q-a)^+, y') - f(q,y)\right)$$

to $G^{\text{MSJ}} \circ f(q,y)$, which are the terms in (6.38) and (6.39) in the lemma statement. □

Lemma 6.7.1.

$$G^{\text{MSJ}} \circ f^{\text{MSJ}}_{\Delta}(q,y) = \mathbb{1}_{q>0} G^{\text{Ak}} \circ f^{\text{Ak}}_{\Delta}(q,y) + \mathbb{1}_{q=0} O_\lambda(1) \tag{6.40}$$

Proof. Let us begin by using Lemmas 6.9.5 and 6.9.6 to give expressions for $G^{\text{MSJ}} \circ f^{\text{MSJ}}_{\Delta}(q,y)$ and $G^{\text{Ak}} \circ f^{\text{Ak}}_{\Delta}(q,y)$.

Note that whenever $q > 0$, $G^{\text{MSJ}} \circ f^{\text{MSJ}}_{\Delta}(q,y)$ is identical to $G^{\text{Ak}} \circ f^{\text{Ak}}_{\Delta}(q,y)$, because the two systems have the same transitions and because $f^{\text{MSJ}}_{\Delta}(q,y)$ and $f^{\text{Ak}}_{\Delta}(q,y)$ are identical.

Note also that whenever $q = 0$, both $G^{\text{MSJ}} f^{\text{MSJ}}_{\Delta}(q,y)$ and $G^{\text{Ak}} \circ f^{\text{Ak}}_{\Delta}(q,y)$ are $O_\lambda(1)$, because $\Delta(y)$ is bounded by a constant for all y, because $\mathbb{Y}^{\text{MSJ}}$ is finite.

As a result,

$$G^{\text{MSJ}} \circ f^{\text{MSJ}}_{\Delta}(q,y) = \mathbb{1}_{q>0} G^{\text{Ak}} \circ f^{\text{Ak}}_{\Delta}(q,y) + \mathbb{1}_{q=0} O_\lambda(1) \quad \square$$

Lemma 6.9.7. *In the MMSR-π system, the departure average distribution Y^{π}_d is given by*

$$\mathbb{P}\left\{Y^{\pi}_d = y'\right\} = \frac{1}{\lambda^*_\pi} \mathbb{E}[\mu^{\pi}_{Y^\pi, y', 1}] \tag{6.41}$$

Proof. We will show that

$$\mathbb{P}\left\{Y^{\pi}_d = y'\right\} = \frac{1}{\lambda^*_\pi} \sum_y \mathbb{P}\left\{Y^\pi = y\right\} \mu_{y,y',1}.$$

As an intermediate step, let Y^π_{DTMC} be the transition-average steady state of the Markov chain π. $\mathbb{P}\{Y^\pi_{DTMC} = y\}$ is the fraction of state-visits that are visits to y, in the embedded DTMC of π.

Let $\mu_{y,\cdot,\cdot}$ be the total transition rate out of state y:

$$\mu_{y,\cdot,\cdot} = \sum_{y',a} \mu_{y,y',a}$$

Note that the CTMC that controls Y^π and the DTMC that controls Y^π_{DTMC} visit the same states in the same order, but that Y^π stays in state y for $Exp(\mu_{y,\cdot,\cdot})$ time for each visit. As a result,

$$\mathbb{P}\{Y^\pi = y\} = a^\pi \mathbb{P}\{Y^\pi_{DTMC} = y\} \frac{1}{\mu_{y,\cdot,\cdot}}$$

where a^π is a normalization constant. Specifically, a^π is the long-term completion rate, which can be calculated as the reciprocal of the average time per visit to a state:

$$a^\pi = \left(\sum_y \mathbb{P}\{Y^\pi_{DTMC} = y\} \frac{1}{\mu_{y,\cdot,\cdot}} \right)^{-1}$$

From Y^π_{DTMC}, we can calculate the fraction of transitions that move from a generic state y to another generic state y' via a completion. Call this fraction $p_{y\to y',1}$.

$$p_{y\to y',1} = \mathbb{P}\{Y^\pi_{DTMC} = y\} \frac{\mu_{y,y',1}}{\mu_{y,\cdot,\cdot}}$$

Summing over all initial states y, we can find the fraction of transitions that are completions which result in the state y':

$$p_{\cdot\to y',1} = \sum_y \mathbb{P}\{Y^\pi_{DTMC} = y\} \frac{\mu_{y,y',1}}{\mu_{y,\cdot,\cdot}}$$

Let b^π be the overall fraction of transitions that are completions. Conditioning on the transition into state y' being a completion, we find that the probability that a generic completion results in state y' is

$$\mathbb{P}\{Y^\pi_d = y'\} = \frac{p_{\cdot\to y',1}}{b^\pi}$$

Combining all of the above equations, we find that

$$\begin{aligned}
\mathbb{P}\{Y^\pi_d = y'\} &= \frac{1}{b^\pi} \sum_y \mathbb{P}\{Y^\pi_{DTMC} = y\} \frac{\mu_{y,y',1}}{\mu_{y,\cdot,\cdot}} \\
&= \frac{1}{b^\pi} \sum_y \frac{1}{a^\pi} \mathbb{P}\{Y^\pi = y\} \mu_{y,\cdot,\cdot} \frac{\mu_{y,y',1}}{\mu_{y,\cdot,\cdot}} \\
&= \frac{1}{a^\pi b^\pi} \sum_y \mathbb{P}\{Y^\pi = y\} \mu_{y,y',1}
\end{aligned}$$

Recall that a^π is the long-term transition rate, and that b^π is the fraction of transitions that are completions. Thus, $a^\pi b^\pi$ is the long-term completion rate $X^\pi = \lambda^*_\pi$. □

6.10 Extensions of RESET: Finite skip models

While our main MSJ result, Theorem 6.5.2, was stated for the MSJ FCFS model, our techniques do not depend on the details of that model. Our RESET technique can handle a wide variety of models, which we call "finite skip" models:

Definition 6.10.1. *A* finite skip *queueing model is one in which jobs are served in near-FCFS order. Only jobs among the n oldest jobs in arrival order are eligible for service, for some constant n. Service is only dependent on the states of the n oldest jobs in arrival order, plus an optional environmental state from a finite-state Markov chain. Furthermore, jobs must have finite state spaces, and arrivals must be Poisson with i.i.d. initial job states.*

Definition 6.10.1 generalizes the work-conserving finite-skip (WCFS) class [86]. The MARC and RESET techniques can characterize the asymptotic mean response time of *any* finite skip model, via the procedure in Fig. 6.3. Additional finite skip MSJ models include nontrivial scheduling policies, including some backfilling policies; changing server need during service; multidimensional resource constraints; heterogeneous servers; turning off idle servers; and preemption overheads.

6.10.1 Nontrival scheduling policies: Backfilling

A common family of MSJ scheduling policies in practice are *backfilling* policies [33, 210, 225]. Under a backfilling policy, the scheduler begins by placing jobs into service in arrival order, as in the FCFS policy. However, once a job is encountered which does not fit in the available servers, additional jobs are considered for service. Some backfilling policies give rise to finite skip models, and can thus be handled by the RESET technique.

As an example, consider the "First Fit-k" policy: The scheduler iterates through the first k jobs in arrival order, checking for each job whether it can be served in the available servers. Each job that fits is served. This policy only serves jobs among the k oldest in arrival order, so it can be handled by the RESET technique.

Beyond backfilling policies, more advanced packing policies can also be considered. For instance, for small k the scheduler could simply search over all subsets of the k oldest jobs and serve the subset with maximal total server need $\leq k$. This policy is also finite skip, and the RESET technique also applies.

6.10.2 Changing server need during service

The standard MSJ model assumes that jobs require a fixed service need throughout their time in service. However, in some settings, jobs may require a varying number of servers. For example, consider a fork-join model with simultaneous start. Suppose that each job is made of some number of tasks, each with independent duration, and each requiring 1 server. As the tasks complete, the server need of the job as a whole diminishes, freeing up space for other jobs to run. This setting still gives rise to a finite-skip model, and poses no difficulty to our RESET technique.

Another natural setting in which server needs change over time is the directed acyclic graph (DAG) setting, in which jobs are broken up into small segments of work, with potentially complex dependencies between segments. The DAG scheduling literature often focuses on scheduling the segments of an individual DAG job. It is natural to consider a scheduling setting where many DAG jobs arrive over time. Holding the DAG scheduling policy constant, this model effectively gives rise to a MSJ model where server needs can vary over time, and potentially vary dynamically in response to the service conditions. As long as the high-level scheduling policy deciding which DAG jobs to run is finite-skip, the model as a whole is finite-skip, and our RESET technique can characterize its asymptotic mean response time.

6.10.3 Multidimensional resource constraints

The standard MSJ model considers a single constrained resource. However, computing jobs are often constrained by a variety of resources, such as CPU, GPU, other accelerators, memory bandwidth, cache capacity, network bandwidth, etc. Such multidimensional resource constraints are often considered in the VM scheduling literature. In that literature, only stability results are known. Our RESET technique thus gives the first characterization of asymptotic mean response time in that setting.

6.10.4 Heterogeneous servers

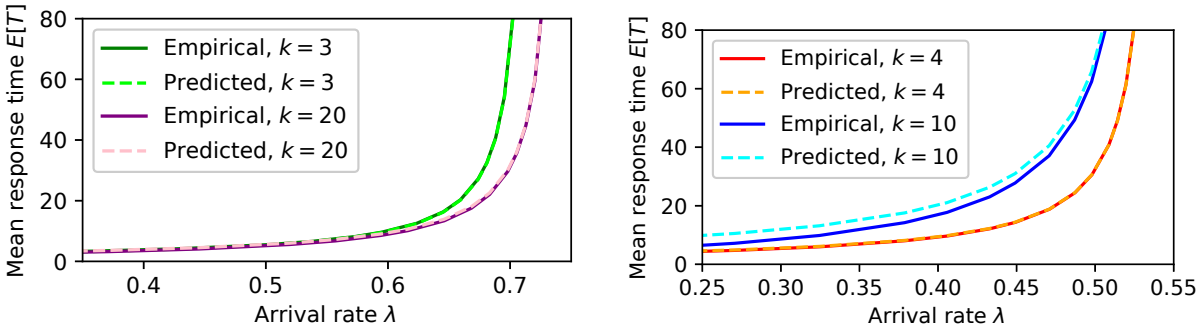
In the standard MSJ model, all servers are identical. However, it is also important to consider settings where different kinds of servers are available, which can provide different amounts of resources. One can also consider jobs that need to be served at a particular server or set of servers, such as a job that processes data stored at that server. In a multidimensional resource setting, some servers may also provide different resources, such as a GPU-heavy or CPU-heavy server. All of these extensions are compatible with the RESET technique.

6.10.5 Turning off idle servers

To improve energy efficiency, it may be preferable to turn off idle servers. Idle servers consume nearly as much energy as active servers. However, turned-off servers take some time to restart. It is important to characterize the impact of this start-up delay on mean response time to understand the tradeoff inherent in turning off idle servers. The process of turning off and on servers can be incorporated into a finite-skip model, because there are a finite number of possible states that the servers can be in. As a result, our RESET technique can provide a characterization of mean response time.

6.10.6 Preemption overheads

The FCFS policy never preempts any jobs. Prior work has studied settings with unlimited preemption. However, practical settings often allow only a limited subset of jobs to be preempted, and jobs may incur an overhead when preemption occurs. This overhead corresponds to the time necessary to snapshot a job in service, and for the new job to be transferred onto the freed servers.



(a) (1) $k = 3$, server need sampled uniformly from $\{1, 2, 3\}$, durations $Exp(1/3)$, $Exp(2/3)$, and $Exp(1)$, respectively. (2) $k = 20$, server need sampled uniformly from $\{1, 20\}$, durations $Exp(1)$ and $Exp(1/2)$, respectively.

(b) (1) $k = 4$, two classes of jobs: Server need 1, duration $Exp(1/4)$ w.p. 42%. Server need 4, duration $Exp(1)$ w.p. 58%. (2) $k = 10$, two classes of jobs: Server need 1, duration $Exp(1/10)$ w.p. 10%. Server need 10, duration $Exp(1)$ w.p. 90%.

Figure 6.4: Empirical and predicted mean response time $\mathbb{E}[T]$ for two MSJ settings in each of figures (a) and (b). Simulated 10^8 arrivals at arrival rates up to $\lambda/\lambda^* = 0.99$.

Models with preemption overheads have only recently begun to be analyzed in the M/G/1 setting [177], with no mean response time analysis known in the one-server-per-job multiserver model, much less the multiserver-job model. Preemption overheads can be modeled with a finite-number of additional states, marking the corresponding servers as undergoing preemption. As a result, our RESET technique can provide a characterization of mean response time.

6.11 Empirical Validation

We have characterized the asymptotic mean response time behavior of the FCFS multiserver-job system. To illustrate and empirically validate our theoretical results, we simulate the mean response time of the MSJ model to compare it to our predictions. Recall (6.3) from Theorem 6.5.2, in which we proved mean response time can be characterized as a dominant term plus a $O_\lambda(1)$ term:

$$\mathbb{E}[T^{\text{MSJ}}] = \frac{1}{\lambda^*} \frac{1 + \Delta(Y_d^{\text{Sat}}, Y^{\text{Sat}})}{1 - \lambda/\lambda^*} + O_\lambda(1) \tag{6.42}$$

In this section, we simulate mean response time $\mathbb{E}[T^{\text{MSJ}}]$, and compare it against the dominant term of (6.42), which we compute explicitly.

6.11.1 Accuracy of formula

In Fig. 6.4a, we show that our predictions are an excellent match for the empirical behavior of the MSJ system in two different settings. In the first, $k = 3$ servers and jobs have server needs of 1, 2, and 3. In the second, $k = 20$ servers, and jobs have server needs 1 and 20. We thereby cover a spectrum from few-server-systems to many-server-systems, demonstrating

extremely high accuracy in both regimes. The $O_\lambda(1)$ term in (6.42) is negligible in both of these examples.

6.11.2 Understanding the importance of Δ

Our results show that the relative completions function Δ is key to understanding the response time behavior of non-work-conserving systems such as the MSJ FCFS system. This is in contrast to work-conserving systems, in which response time is determined by the size distribution and load [86]. To illustrate this contrast, in Fig. 6.4b, we compare mean response time in two settings with the same size distribution and stability region, but which have very different Δ.

The first setting has $k = 4$, and 42% of jobs have server need 1, while 58% of jobs have server need 4. The second setting has $k = 10$, and 10% of jobs have server need 1, while 90% of jobs have server need 10. The settings' stability regions are near-identical: $\lambda_4^* \approx 0.5413, \lambda_{10}^* \approx 0.5411$, and their *size* distributions, defined as duration times server need over k, are both $Exp(1)$. However, our predictions for mean response time are very different in the two settings: $\Delta(Y_d^{\text{Sat}})_4 \approx 0.3271, \Delta(Y_d^{\text{Sat}})_{10} \approx 2.850$. The $k = 10$ setting considered here, with its relatively large value of $\Delta(Y_d^{\text{Sat}})$, is an especially difficult test-case. Nonetheless, our predictions are validated by the simulation results in Fig. 6.4b.

The differing mean response time behavior in these two settings is caused by the difference in *waste correlation*. In the $k = 10$ case, wasteful states persist for long periods of time: If a 1-server job is the only job in service, it takes more time for it to complete than in the $k = 4$ system. Thus, in the $k = 4$ case, wasteful states are more short-lasting. This difference in waste correlation produces the differences in $\Delta(Y_d^{\text{Sat}})$ and in mean response time.

This example highlights a crucial feature of MSJ FCFS: The failure of work conservation injects idiosyncratic idleness patterns in to the system. To characterize $\mathbb{E}[T]$, we need to characterize these patterns, which the RESET and MARC techniques enable us to do for the first time.

6.12 Technical Conclusion

We introduce the RESET and MARC techniques. The RESET technique allows us to reduce the problem of characterizing mean response time in the MSJ FCFS system, up to an additive constant, to the problem of characterizing the M/M/1 with Markovian service rate (MMSR), where the service process is controlled by the saturated system. The MARC technique gives the first explicit characterization of mean response time in the MMSR, up to an additive constant. Together, our techniques reduce $\mathbb{E}[T^{\text{MSJ}}]$ to two properties of the saturated system: the departure-average steady state Y_d^{Sat}, and the relative completions function $\Delta(y_1, y_2)$. Our RESET and MARC techniques apply to any finite skip model, including many MSJ generalizations.

We also introduce the simplified saturated system, a yet-simper variant of the saturated system with identical behavior. We empirically validate our theoretical result, showing that it closely tracks simulation at all arrival rates λ.

An important direction for future work is to analytically characterize the relative completions

$\Delta(y_1, y_2)$ for specific MSJ FCFS settings, such as settings where Y_d^{Sat} is known to have a product-form distribution [83, 88, 187].

6.13 General Conclusion

6.13.1 Summary

We start by summarizing the results proven in this chapter, as well as the key techniques behind these results.

Results: Analyze MSJ FCFS mean response time We characterize the FCFS scheduling policy's mean response time, in the form of a clean mathematical formula (See Theorem 6.5.2). This formula differs from the system's exact mean response time by at most an additive constant which does not depend on the arrival rate. The formula becomes tight as the arrival rate grows and the queue becomes long (the *heavy-traffic* limit).

Simulation result: Strong approximation at all loads While our results are tightest in the heavy-traffic limit, we show via simulation in Fig. 6.4 that our formula closely approximates the real mean response time across all arrival rates. In particular, we see that the difference between our formula and the simulated value is almost negligible even at moderate arrival rates.

Results: Other settings and scheduling policies Our results generalize to prove similar mean response time characterizations for a wide family of MSJ settings and scheduling policies, as we discuss in Section 6.10. In particular, our analysis can handle any *finite skip* system, which we define in Definition 6.10.1. Intuitively, a finite-skip system is any system in which jobs complete in near-FCFS order, regardless of variations in the number of servers which may be active. In particular, we can analyze MSJ systems with multidimensional resources, jobs which vary in resource requirement over time, scheduling policies which deviate from FCFS service to a limited extent, and many other important models.

Results: M/M/1 with Markovian Service Rate (MMSR) As an important stepping stone to analyzing the MSJ FCFS system, we characterize the mean response time of the MMSR system, again up to an additive constant (See Theorem 6.5.1). The MMSR system is a queueing system in which all jobs are identical, but the rate at which jobs complete varies over time, according to an arbitrary finite-state Markov chain. The MMSR system is an important system in its own right, with extensive prior work on the subject [39, 47, 95, 131, 151, 166]. Our result is the first closed-form mean response time analysis of the general MMSR system.

Key techniques Our key route of analysis is to break the MSJ system into two simpler queueing systems: the MMSR system and the saturated system. The saturated system is a closed queueing system where completions trigger new arrivals (See Section 6.4.5). As a result, there are always exactly k jobs in the system. Note that the saturated system is a finite-state Markov chain. We consider an MMSR system where the service rate is controlled by the saturated system. This combination is equivalent to a queueing system which we call the "At-least-k" system (See Sections 6.4.4 and 6.4.6). The At-least-k system is very similar to the MSJ system, in that there is an external arrival process and MSJ service. However, in the At-least-k system, whenever there are exactly k jobs in the system, if a job completes, a fresh job is immediately added to the system.

Thus, our analysis has two steps:

1. **Reduction to Saturated for Expected Time (RESET):** We prove that the MSJ and At-least-k systems have almost identical mean response times (See Theorem 6.5.2).
2. **Markovian Relative Completions (MARC):** We analyze the mean response time of the MMSR system (See Theorem 6.5.1), of which the At-least-k system is a special case. A key step in this analysis is the *relative completions* function Δ_π (See Section 6.4.8).

We refer to our overall analysis process as "finite-skip analysis".

6.13.2 Future Directions

Optimizing Finite-skip Scheduling There are a wide variety of finite-skip scheduling policies in the MSJ system. Any policy which selects among the k oldest jobs in the system is a finite-skip policy. Some such policies will keep far more servers full than FCFS, achieving better response times.

A natural open question is:

> Which finite-skip policy achieves the lowest mean response time?

Using our analysis in Theorem 6.5.2, this question can be simplified into two steps, at least in heavy traffic: What policy achieves the highest completion rate in the saturated system, and what policy achieves the lowest $\Delta_\pi(Y_d^\pi)$ in the saturated system, given that completion rate? We could try to answer these questions analytically, or we could try to answer them algorithmically. Algorithmically, this is a relatively standard problem in Markov Decision Processes (MDPs). In particular, achieving the highest completion rate is a standard average-cost MDP. Achieving the lowest $\Delta_\pi(Y_d^\pi)$ is not a standard MDP problem, so some further research may be needed. We discuss this problem further in Section 8.3.4.

6.13.3 Potential Impact

We now explore potential directions in which our predictions of mean response times of finite-skip scheduling policies could be applied to real-world environments.

Using Finite-Skip Analysis to Predict Response Times in Modern Computing Systems

Our analysis of MSJ FCFS in particular, and of all finite-skip scheduling policies more generally, can provide insight into the response times of a wide variety of important computing systems. These predictions of response time can allow better-informed decision making in these systems. For instance, in the dispatching setting, if a dispatcher is sending jobs to one of several computing clusters, the dispatcher could predict the response time at a given cluster when deciding where to send the job. In the capacity provisioning setting, our analysis could be used when choosing how many servers to employ to handle a given workload.

However, the road from theoretical results to adoption requires overcoming several hurdles. These include:

Multidimensional resource requirements: Hard and soft resources In real systems, jobs often require a variety of resources, such as CPU cores, GPUs, memory, network bandwidth, disk IO, and more. As discussed in Section 6.10, our finite-skip analysis can accommodate multidimensional resources. An important distinction in modeling these resources is between *hard* and *soft* resource requirements. A hard requirement is one where the job cannot be served at all without all of the specified resources. A soft requirement is one where the job can be served with a fraction of its required resources, but it will suffer a performance penalty. Our finite-skip analysis can accommodate either hard or soft requirements, as well as a mixture of the two. Establishing accurate models of the resource requirements in a given system is vital to getting accurate performance predictions using our finite-skip analysis.

Computationally intensive analysis Our mean response time analysis in Theorem 6.5.2 reduces the problem of understanding MSJ FCFS mean response time to the problem of understanding the behavior of the corresponding saturated system (See Section 6.4.5). Because the saturated system is a finite-state Markov chain, understanding its behavior is possible in principle using computational methods. Unfortunately, the number of possible states of the saturated system grows exponentially with the number of servers. To overcome this issue, we devised the simplified saturated system (See Section 6.4.11), which has a dramatically smaller number of states. In addition, for some settings, theoretical analysis of the saturated system may allow us to bypass computational methods entirely [83, 88, 187].

Prioritization In real computing systems, it is typically the case that some jobs have higher priority than others. One might still use a policy like MSJ FCFS within a class of jobs with equal priority. Our finite-skip analysis does not immediately apply to systems with prioritization. However, in single-server systems, the "transformer glasses" technique is used to relate the mean response time of a system with prioritization to that of a FCFS system with a different workload [104, 201]. The same technique could likely be applied in the MSJ setting to relate the mean response time of a MSJ scheduling policy with prioritization to that of a MSJ FCFS system with a different workload, and then apply our finite-skip analysis to characterize that system's mean response time.

Predicting Mean Response Time when Scheduling People

Consider a real-world scenario where you running a contracting company, and you receive projects to work on. Each project requires some number of people to complete. Moreover, people are not all the same: A project might require one technical expert and four general workers, for instance. Different people might have different skills, such as expertise on different topics. When projects come in, they are handled in FCFS order. Given that different projects require different skill-sets and different numbers of employees, we are interested in figuring out the average time to complete projects. The results in this chapter allow us to predict and understand the average time to completion for projects in this scenario.

Part IV

Response Time Tails

Chapter 7

Better Response Time Tail than FCFS: Nudge

This chapter is based on the paper "Nudge: Stochastically Improving upon FCFS", published in SIGMETRICS in 2021, written with my coauthors Kunhe Yang, Ziv Scully, and Mor Harchol-Balter [85].

7.1 General Introduction

Tail performance matters People who design and operate computing systems overwhelmingly use *tail* performance metrics to quantify the performance of their systems. Common tail metrics include T^{99}, the 99th percentile of job response time, or $\mathbb{P}\{T > 1s\}$, the fraction of jobs whose response time exceeds one second. Here "response time" refers to the distribution of durations from when a job arrives to the system to when it completes. Service Level Objectives, the actual contracts which define whether or not a system is functioning adequately, are typically defined using these tail metrics.

Scheduling for tail performance The choice of scheduling policy has a dramatic effect on the response time distribution, and the system's tail performance in particular. To capture the effect of scheduling on the response time distribution, we use a single-server queueing model. While a multiserver queueing model would be more realistic, understanding tail scheduling in the single-server model already presents several major open problems, so we focus on the single-server model throughout this chapter.

Advantages of FCFS A natural choice of scheduling policy for optimizing the tail of response time is the First-Come First-Served (FCFS) policy. As an indication of its effectiveness, one can show that on a *finite* sequence of jobs, FCFS minimizes the *maximum* response time of those jobs. To understand why, consider deciding how to serve just a pair of jobs. Whichever order the two jobs are served, the second of the two jobs will complete at the same point in time. To minimize the maximum of the two jobs' response times, the second job to complete must be the job that arrived second. Among the pair, jobs must be served in arrival order. By applying this argument inductively, one can show that all jobs must be served in arrival order, giving rise to the FCFS scheduling policy.

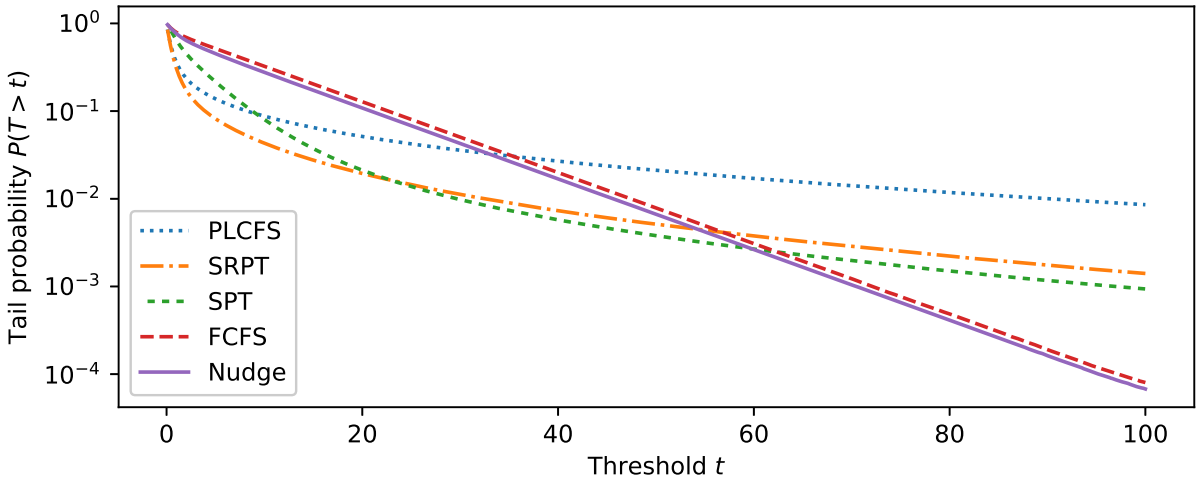


Figure 7.1: Comparison via simulation of tail probability under a variety of scheduling policies. The model is an M/G/1: Single server, Poisson arrival process, i.i.d. job sizes. The job size distribution is a hyperexponential distribution with two branches: $2/3$ of jobs are distributed as an exponential with mean $3/4$, and $1/3$ as an exponential with mean $3/2$. Server is occupied $\rho = 0.8$ fraction of time, 2 billion arrivals simulated.
Five scheduling policies are shown: First-Come First-Served (FCFS) serves jobs in arrival order. Preemptive Last-Come First-Served (PLCFS) preemptively serves jobs in reverse arrival order. Preemption refers to pausing a job in service, to be served again later. Shortest Processing Time (SPT) non-preemptively serves the job of smallest size. Shortest Remaining Processing Time (SRPT) preemptively serves the job of smallest remaining size. Nudge is our new policy.
Note that Nudge's tail probability lies strictly below FCFS's tail probability for all thresholds t.

In a stochastic setting, instead of the maximum response time, we study the asymptotic tail of response time. This is the behavior of the *tail probability* $\mathbb{P}\{T > t\}$, the probability that response time exceeds a threshold t, as the threshold t goes to infinity.

FCFS is known to achieve an asymptotic tail of response time which is at least *close* to optimal [27]. Moreover, FCFS is conjectured to achieve the *best possible* asymptotic tail of response time [232]. This optimality is conjectured even when comparing against scheduling policies which make use of size information: i.e. have prior knowledge of the duration of each job.

Our policy: Nudge We devise a new scheduling policy, Nudge, which we prove achieves better asymptotic tail of response time than FCFS. Nudge is the first policy to improve upon FCFS's asymptotic tail, overturning prior conjecture [232]. Moreover, we prove that Nudge achieves a much stronger property: For every threshold t, Nudge achieves a smaller tail probability $\mathbb{P}\{T > t\}$ than FCFS.

Nudge intuition The intuition behind Nudge is that we should be able to make use of size information to improve the tail performance of FCFS. However, standard size-based scheduling policies such as Shortest Remaining Processing Time (SRPT) give far too much priority to small jobs. While SRPT achieves optimal mean response time, its tail performance can be quite poor,

as we show in Fig. 7.1. This poor tail performance is caused by large jobs suffering very long response times. To improve upon FCFS's tail performance, we need to be very gentle in our use of size information, so that large jobs do not suffer too much. Nudge is designed to use size information in this gentle fashion.

Nudge definition Nudge classifies all jobs into three categories, based on the jobs' sizes: *small*, *large*, and *other*. Jobs are initially placed into the queue in arrival order. If a small job arrives to the system and finds a large job at the back of the queue, the small job moves ahead of the large job. However, to minimize the effect of this reordering on the tail of response time, Nudge only allows each small job to move past at most one large job, and only allows each large job to be moved past by at most one small job. Nudge then serves the job at the front of the queue.

Effect of scheduling on tail To visualize the improvement of Nudge over FCFS, in Fig. 7.1, we depict the tail probabilities of several common scheduling policies, as well as our new policy Nudge. For small thresholds t, PLCFS, SPT, and SRPT have low tail probabilities, but as the threshold t grows large, all three have much worse tail probability. In contrast, FCFS and Nudge achieve very good tail probabilities for large thresholds t.

Nudge improves upon FCFS's tail In Fig. 7.1, notice that our new policy, Nudge, achieves smaller tail probabilities than FCFS for every tail threshold t. Our main result in this chapter is to prove that Nudge *always* achieves smaller tail probabilities than FCFS, for all job size distributions in a broad class known as "light-tailed" distributions.

Because Nudge improves upon FCFS for all tail probability thresholds t, Nudge also improves upon FCFS with regards to all common tail performance metrics, including all percentiles of response time such as T^{99}, and all moments of response time such as $\mathbb{E}[T^2]$.

7.2 Technical Introduction

7.2.1 The Case for FCFS

While advanced scheduling algorithms are a popular topic in theory papers, it is unequivocal that the most popular scheduling policy used in practice is still First-Come First-Served (FCFS). There are many reasons for the popularity of FCFS. From a practical perspective, FCFS is easy to implement. Additionally, FCFS has a feeling of being fair.

However, there are also theoretical arguments for why one should use FCFS. For one thing, FCFS minimizes the *maximum* response time across jobs for *any* finite arrival sequence of jobs. By *response time* we mean the time from when a job arrives until it completes service.

For another thing, in an M/G/1 with a light-tailed job size distribution, FCFS is known to have a *weakly optimal* asymptotic tail of response time [27, 213]. Specifically, using T to denote response time, the asymptotic tail under FCFS is of the form:

$$\mathbb{P}\left\{T^{\mathsf{FCFS}} > t\right\} \sim C_{\mathsf{FCFS}} e^{-\theta^* t}, \tag{7.1}$$

where "$\sim$" indicates that the ratio of the two quantities converges to 1 in the $t \to \infty$ limit.

The exponent θ^* in (7.1) is known to be optimal, while the optimality of C_{FCFS} is an open problem [27]. The asymptotic tail growth under FCFS has been compared with more sophis-

ticated policies [27]. It has been shown that, for light-tailed job size distributions, the tail of response time under Processor-Sharing (PS), Preemptive Last-Come-First-Served (PLCFS), and Shortest-Remaining-Processing-Time (SRPT) each take the asymptotic form of

$$\mathbb{P}\{T > t\} \sim C' e^{-\theta' t},$$

where θ' is the *worst possible* exponential decay rate [164] over all work-conserving scheduling policies. Roughly, FCFS's tail exponent θ^* arises from the tail of the workload distribution, while the other policies' tail exponent θ' arises from the tail of the busy period distribution, which is much larger under light-tailed job size distributions.

7.2.2 The Case For Light-Tailed Job Size Distributions

In this chapter, we choose to focus on the case of light-tailed job size distributions. Light-tailed job size distributions show up naturally in workloads where all the transactions are of the same type (say shopping); while there is some variability in the time it takes to purchase an item, even high-variability distributions that arise in such settings are often light-tailed. Also, many natural distributions, like the Normal distribution, Exponential distribution, and all Phase-type distributions, are light-tailed. Finally, while heavy-tailed job size distributions are certainly prevalent in empirical workloads (see for example [42, 101, 218]), in practice, these heavy-tailed workloads are often *truncated*, which immediately makes them light-tailed. Such truncation can happen because there is a limit imposed on how long jobs are allowed to run. Alternatively, truncation can occur when a heavy-tailed job size distribution is divided into a few size classes as in [103, 108] where the smaller size classes end up being truncated distributions.

7.2.3 The Case for Non-Asymptotic Tails

Within the world of light-tailed job size distributions, FCFS is viewed as the best policy. However, while FCFS has a weakly optimal *asymptotic* tail, it is not best at minimizing $\mathbb{P}\{T > t\}$ for *all* t. In practice, one cares less about the asymptotic case than about particular t [105]. For example, one might want to minimize the fraction of response times that exceed $t = 0.5$ seconds, because such response times are noticeable by users. One might also want to meet several additional Service Level Objectives (SLOs) where one is charged for exceeding particular response time values, such as $t = 1$ minute, or $t = 1$ hour. SLOs are very common in the computing literature [35, 105, 158], in service industries [45, 209, 219], and in healthcare [22, 116]. Unfortunately, different applications have different SLOs. This leads us to ask:

> *When considering $\mathbb{P}\{T > t\}$, is it possible to strictly improve upon FCFS for <u>all</u> values of t?*

We are motivated by the fact that, for lower values of t, Shortest-Remaining-Processing-Time (SRPT) is better than FCFS, although FCFS clearly beats SRPT for higher values of t, as FCFS is weakly asymptotically optimal while SRPT is asymptotically pessimal [164, 172]. SRPT also minimizes mean response time [194], which is closely related to lower values of t. This motivates us to consider whether prioritizing small jobs might have some benefit, even in the world of light-tailed job size distributions.

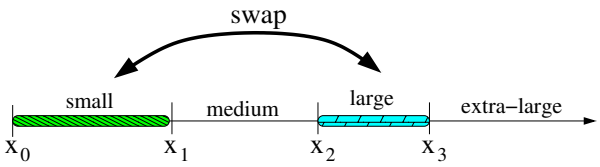


Figure 7.2: The Nudge algorithm.

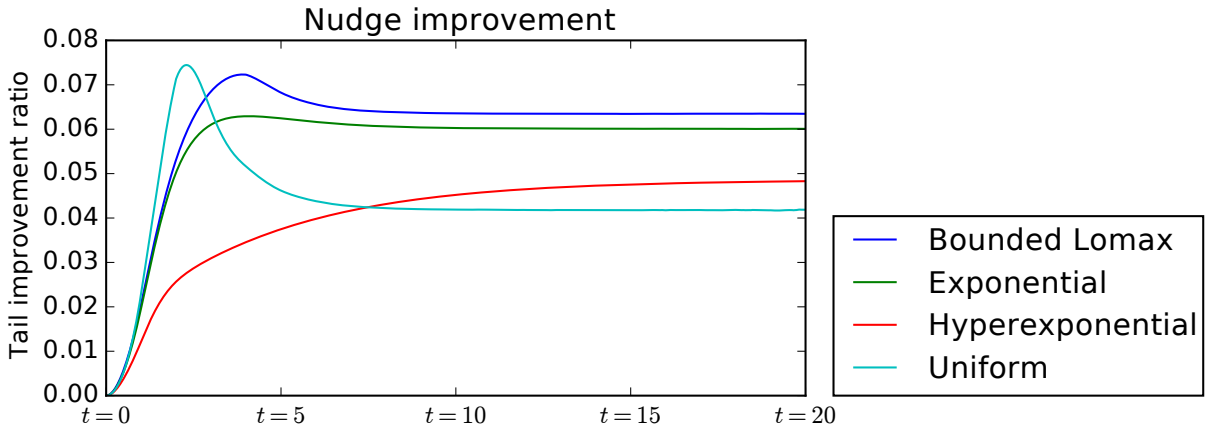


Figure 7.3: Empirical tail improvement of Nudge over FCFS in an M/G/1. The tail improvement ratio (TIR) is defined as $1 - \mathbb{P}\left\{T^{\text{Nudge}} > t\right\} / \mathbb{P}\left\{T^{\text{FCFS}} > t\right\}$. Specific job size distributions, each with mean 1: Uniform(0,2); Exponential with mean 1; Hyperexponential with branches drawn from Exp(2) and Exp(1/3), where the first branch has probability 0.8 and where $C^2 = 3$; BoundedLomax($\lambda = 2$, max$= 4$, $\alpha = 2$). Distributions in legend ordered by asymptotic improvement. Simulations run with 10 billion arrivals. Load $\rho = 0.8$. Nudge parameters: $x_1 = 1, x_2 = 1, x_3 = \infty$. Note that $x_1 = x_2$, so there are no medium-sized jobs. Empirically, Nudge often achieves its best performance with $x_1 = x_2$, though our proofs involve setting $x_1 < x_2$. See Section 7.10.

We ask more specifically:

> *Can partial prioritization of small jobs lead to a strict improvement over FCFS? Specifically, is there a scheduling policy which strictly improves upon FCFS with respect to* $\mathbb{P}\{T > t\}$*, for* every *possible* t *including large* t*?*

7.2.4 Our Answer: Nudge

This chapter answers the above question in the affirmative. We will define a policy, which we call *Nudge*, whose response time tail is *provably* better than that of FCFS for *every* value of t, assuming a light-tailed job size distribution[1] (see Theorem 7.5.2). We say that Nudge's response time *stochastically improves upon* that of FCFS, in the sense of stochastic dominance. Moreover,

[1]Technically, a Class I job size distribution. See Definition 7.4.3.

we prove that the asymptotic tail of response time of Nudge is of the form

$$P\{T^{\mathsf{Nudge}} > t\} \sim C_{\mathsf{Nudge}} e^{-\theta^* t},$$

with optimal decay rate θ^* and a superior leading constant $C_{\mathsf{Nudge}} < C_{\mathsf{FCFS}}$ (see Corollary 7.5.1). Thus, we demonstrate that FCFS is *not* strongly optimal, answering an open problem posed by Boxma and Zwart [27]. In particular, this is contrary to a conjecture of Wierman and Zwart [232] (see Section 7.3.1).

The intuition behind the Nudge algorithm is that we'd like to basically stick to FCFS, which we know is great for handling the extreme tail (high t), while at the same time incorporating a little bit of prioritization of small jobs, which we know can be helpful for the mean and lower t. We need to be careful, however, not to make too much use of size, because Nudge still needs to beat FCFS for high t; hence we want just a little "nudge" towards prioritizing small jobs.

We now describe the Nudge algorithm. Imagine that the job size distribution is divided into size regions, as shown in Fig. 7.2, consisting of small, medium, large, and extra large jobs. Most of the time, Nudge defaults to FCFS. However, when a "small" job arrives and finds a "large" job immediately ahead of it in the queue, we swap the positions of the small and large job in the queue. The one caveat is that a job which has already swapped is ineligible for further swaps. The size cutoffs defining small and large jobs will be defined later in this chapter.

The degree of the tail improvement of Nudge over FCFS is non-trivial. In Fig. 7.3, we see that for many common light-tailed job size distributions, Nudge results in a multiplicative improvement of 4-7% throughout the tail. In Section 7.10.2, we show that with low load and a high-variability job size distribution, Nudge's improvement can be as much as 10-15% throughout the tail. The magnitude of these improvements highlights the importance of scheduling, even in the light-tailed setting.

We additionally present an exact analysis of the performance of Nudge. Nudge does not fit into any existing framework for M/G/1 transform analysis, including the recently developed SOAP framework [201] (see Section 7.3.3). Nonetheless, we derive a tagged-job analysis of Nudge in Theorem 7.5.4, deriving the Laplace-Stieltjes transform of response time of Nudge.

7.2.5 Contributions and Roadmap

This chapter makes the following contributions.

- In Section 7.4.5, we introduce the Nudge policy.
- In Sections 7.5 to 7.7 we prove that with appropriately chosen parameters, Nudge stochastically improves upon FCFS for light-tailed[2] job size distributions; we also give a simple expression for such parameters. Moreover, in Section 7.9, we prove that Nudge achieves a multiplicative asymptotic improvement over FCFS.
- In Section 7.8, we derive the Laplace-Stieltjes transform of response time under Nudge.
- In Section 7.10, we empirically demonstrate the magnitude of Nudge's stochastic improvement over FCFS. We also discuss how to tune Nudge's parameters for best performance.
- In Section 7.11, we discuss practical considerations for using Nudge.
- In Section 7.12, we discuss several notable variants of Nudge.

[2]Technically, any Class I job size distribution. See Definition 7.4.3.

7.3 Prior Work

Most prior work on scheduling to optimize the tail of response time focuses on the *asymptotic* case, characterizing $\mathbb{P}\{T > t\}$ in the $t \to \infty$ limit. We review these results in Section 7.3.1.

Our main result, Theorem 7.5.2, is a *non-asymptotic* statement, characterizing the behavior of $\mathbb{P}\{T > t\}$ for *all* t, not just the $t \to \infty$ limit. There is much less prior work on the tail of response time outside of the asymptotic regime. We review the few results in this area in Section 7.3.2.

In addition to characterizing Nudge's tail of response time, we also give an exact transform analysis of Nudge's response time. Our analysis requires a novel approach that significantly differs from traditional analyses, as we discuss in Section 7.3.3.

This chapter's focus is the M/G/1 queue. All of the results cited in this section apply to the M/G/1, with some also applying to more general models, such as the GI/GI/1.

7.3.1 Asymptotic Tails: Extensive Theory, but Open Problems Remain

When optimizing the asymptotic tail, the goal is to find a policy π^* such that for all scheduling policies π,

$$\limsup_{t\to\infty} \frac{\mathbb{P}\left\{T^{\pi^*} > t\right\}}{\mathbb{P}\left\{T^{\pi} > t\right\}} \leq c$$

for some constant $c \in [1, \infty)$. Such a policy π^* is called *weakly optimal*; if $c = 1$, then π^* is called *strongly optimal* [27]. While weak optimality has been well studied, proving or disproving strong optimality is much harder.

One major theme of the prior work is that optimizing the asymptotic tail looks very different depending on the job size distribution.

- For light-tailed job sizes, FCFS is weakly optimal [27]. Specifically, the tail of response time has a form given in (7.1). Moreover, many popular preemptive policies such as PS, SRPT, and Foreground-Background (FB)[3] are "weakly pessimal": they have the *maximum* possible asymptotic tail, up to a constant factor, of any work-conserving scheduling policy [164, 172].
- For heavy-tailed job sizes, the reverse is true: PS, SRPT, FB, and similar policies are all weakly optimal [27, 172, 203], while FCFS is weakly pessimal.

This state of affairs prompts a question: is any policy weakly optimal in both the light-tailed and heavy-tailed cases? Nair et al. [164] show that a variant of PS achieves this, but their variant requires knowledge of the system's load. Wierman and Zwart [232] show that *any* policy that is weakly optimal in both the light- and heavy-tailed cases requires knowing some information about the system parameters, such as the load.

The above results mostly characterize weakly optimal scheduling policies, but the problem of strongly optimizing the tail remains open. Boxma and Zwart [27] pose the strong optimality of FCFS as an open problem. Wierman and Zwart [232] go further and conjecture that FCFS is in fact strongly optimal for light-tailed job size distributions. Despite a large of body of work

[3] FB at all times serves the jobs that have received the least service so far.

analyzing the tail asymptotics of FCFS [2–4, 191], the problem has remained open. We solve the problem by showing that FCFS is *not* strongly optimal. Specifically, our Corollary 7.5.1 implies

$$\lim_{t\to\infty} \frac{\mathbb{P}\left\{T^{\mathsf{Nudge}} > t\right\}}{\mathbb{P}\left\{T^{\mathsf{FCFS}} > t\right\}} = \frac{C_{\mathsf{Nudge}}}{C_{\mathsf{FCFS}}} < 1.$$

7.3.2 Non-asymptotic Tails: Few Optimality Results

Characterizing $\mathbb{P}\{T > t\}$ outside the asymptotic regime is a much harder problem than characterizing the asymptotic tail. As such, the strongest results in this area are for relatively simple scheduling policies. For FCFS under light-tailed job size distributions, it is known that $\mathbb{P}\{T > t\} < e^{-\theta^* t}$ for the same constant θ^* as appears in FCFS's asymptotic tail formula [126, 127]. As a result, this bound is tight up to a constant factor [127], subject to subtleties discussed in Section 7.4.3. Beyond FCFS, one of the few known results gives an improved characterization of response time under preemptive-priority scheduling policies [2, Section 2].

Very little is known about more complicated scheduling policies. While the Laplace-Steiltjes transform of T is known for a wide variety of scheduling policies [201, 211], these transforms do not readily yield useful bounds on $\mathbb{P}\{T > t\}$ for general job size distributions.

Given that characterizing $\mathbb{P}\{T > t\}$ is difficult, it comes as no surprise that optimizing $\mathbb{P}\{T > t\}$ is harder still. As such, rather than trying to crown a single optimal policy, we focus on a *relative* measure. Specifically, as we define in Definition 7.4.1, we say that policy π_1 *stochastically improves upon* another policy π_2 if $\mathbb{P}\{T^{\pi_1} > t\} \leq \mathbb{P}\{T^{\pi_2} > t\}$ for all t.

There are two stochastic improvement results in the literature, but both are much simpler than our Nudge result. Both results start with a well-known policy that does not use job sizes and show that a variation that does use job sizes stochastically improves response time.

- Nuyens et al. [172] show that SRPT and similar policies stochastically improve upon FB.
- Friedman and Henderson [66] and Friedman and Hurley [67] show that one can stochastically improve upon PS by using job sizes. Their policy, Fair Sojourn Protocol (FSP), guarantees in a sample-path sense that no job departs later than it would if the server were using PS.

The results above fit a common theme. Both FB and PS often share the server between multiple jobs. Sharing the server is fundamentally suboptimal. For example, when sharing the server between jobs 1 and 2, if we knew that we would finish job 1 first, then it would be better to devote the entire server to job 1 at first. Doing so improves the response time of job 1 without harming the response time of job 2. Roughly speaking, when FB and PS would share the server between jobs, SRPT and FSP serve the jobs one at a time, using job size information to choose the ordering.

FCFS is more difficult to stochastically improve upon than FB and PS. For one thing, FCFS never shares the server, removing this easy opportunity for stochastic improvement. Moreover, there is a sense in which FCFS is unimprovable: on any specific finite arrival sequence, FCFS minimizes the sorted vector of response times, where we order vectors lexicographically. For example, FCFS minimizes the maximum response time. As a result, the sample path arguments that work for improving FB and PS do not apply to improving FCFS.

In spite of these obstacles, we show in Theorem 7.5.2 that Nudge stochastically improves upon FCFS. Rather than reasoning in terms of sample paths, we take a fundamentally stochastic approach from the beginning. See our proofs in Section 7.6.

7.3.3 Transform of Response Time: Nudge Needs a Novel Approach

In Theorem 7.5.4, we give a closed-form expression for the Laplace-Stieltjes transform of Nudge's response time. There has been much prior work on analyzing the transform of response time of the M/G/1 under various scheduling policies. Some analysis techniques cover a wide variety of scenarios [68, 201]. However, as we explain below, none of these prior techniques can analyze Nudge.

SOAP Policies. Policies in the SOAP class, introduced by Scully et al. [201], schedule jobs based on an index calculated from each job's size and attained service[4], and their response time can be analyzed via the SOAP framework [201]. These include SRPT [196], FB [195], some multi-level processor sharing policies [130], and certain cases of the Gittins policy [173]. Unfortunately, Nudge is not a SOAP policy, so we cannot leverage this analysis method. This is because whether Nudge will swap a small job s with a large job ℓ depends in part on whether any other jobs arrive between ℓ and s. In contrast, a SOAP policy would make such a decision based on properties of ℓ and s alone.

Variations on FCFS. Nudge serves jobs in FCFS order by default and only ever swaps adjacent arrivals. One might therefore hope that Nudge could be analyzed as a variation on FCFS. There are many papers analyzing a variety of M/G/1 variants under FCFS scheduling. These include systems with generalized vacations [68] and exceptional first service [228]. Unfortunately, to the best of our knowledge, no prior analysis of a variation of FCFS applies to Nudge.

Other Analysis Techniques. There are a number of scheduling policies whose transform analyses do not fit into either of the previous categories, such as random order of service [129] and systems with accumulating priority [211]. However, these policies do not resemble Nudge, and the techniques used in their analyses do not readily apply to Nudge.

7.4 Model

7.4.1 Notation

We consider the M/G/1 queue in which job sizes are known. Let λ be the arrival rate, S be the job size distribution, and $s_{\min}$ be the minimum possible job size. Specifically, let $s_{\min}$ be the infimum of the support of S. We denote the load by $\rho = \lambda \mathbb{E}[S]$ and assume $0 < \rho < 1$.

The queueing time, T_Q, is the time from when a job arrives until it first receives service. The response time, T, is the time from when a job arrives until it completes. We write T_Q^{Alg} and T^{Alg} for the queueing time and response time under scheduling algorithm Alg, respectively.

For any continuous random variable V, we will use $f_V(\cdot)$ to denote the probability density function (p.d.f.) of V. We write the $\widetilde{V}(s)$ for the Laplace-Stieltjes transform of V.

[4]The index can also depend on certain other characteristics of the job, e.g. its class if there are multiple classes of jobs. Size and attained service are the attributes relevant to Nudge.

7.4.2 Stochastic Improvement

In this chapter, our goal is to prove that the Nudge policy stochastically improves upon the FCFS policy. We now define stochastic improvement, along with the related notion of tail improvement ratio.

Definition 7.4.1 (Stochastic Improvement)**.** *For two scheduling algorithms* Alg_1 *and* Alg_2*, we say that* Alg_1 *(strictly)* stochastically improves *upon* Alg_2 *if, for any response time cutoff* $t > s_{\min}$*, the probability that response time of* Alg_1 *exceeds* t *is smaller than the probability that* Alg_2*'s response time exceeds* t*, i.e.,*

$$\forall t > s_{\min}, \quad \mathbb{P}\left\{T^{\mathsf{Alg}_1} > t\right\} < \mathbb{P}\left\{T^{\mathsf{Alg}_2} > t\right\}.$$

Definition 7.4.2 (Tail improvement ratio)**.** *For any response time cutoff* t*, the* tail improvement ratio *of* Alg_1 *versus* Alg_2 *at* t*, denoted* $\mathsf{TIR}(t)$*, is defined as*

$$\mathsf{TIR}(t) \triangleq 1 - \frac{\mathbb{P}\left\{T^{\mathsf{Alg}_1} > t\right\}}{\mathbb{P}\left\{T^{\mathsf{Alg}_2} > t\right\}}.$$

The asymptotic tail improvement ratio*, denoted* $\mathsf{AsymTIR}$*, is defined as*

$$\mathsf{AsymTIR} \triangleq \liminf_{t\to\infty} \mathsf{TIR}(t) = 1 - \limsup_{t\to\infty} \frac{\mathbb{P}\left\{T^{\mathsf{Alg}_1} > t\right\}}{\mathbb{P}\left\{T^{\mathsf{Alg}_2} > t\right\}}.$$

7.4.3 Class I "Light-Tailed" Distributions

In this chapter, we focus on job size distributions for which the FCFS policy has an asymptotically exponential waiting time distribution. This property of FCFS will be crucial for our analysis. Prior work has exactly characterized the job size distributions for which FCFS has this property. These distributions are known as "class I" distributions [2, 3, 191].

Definition 7.4.3 (Class I Distribution)**.** *For a distribution* S*, let* $-s^*$ *be the rightmost singularity of* $\widetilde{S}(s)$*, with* $-s^* = -\infty$ *if* $\widetilde{S}(s)$ *is analytic everywhere.* S *is a* class I distribution *if and only if* $s^* > 0$ *and* $\widetilde{S}(-s^*) = \infty$*.*

Class I distributions can roughly be thought of as "well-behaved" light-tailed distributions. In contrast, class II distributions, the other class of light-tailed distributions, are very unusual and "paradoxical", and rarely occur as job size distributions.

For this chapter, the key property of class I job size distributions is that they cause FCFS to have an asymptotically exponential waiting time distribution for all loads [3, 4]. However, as shown by [3, 4], the waiting time also exhibits an exponential tail for light load if the job size is class II. For this reason, while we focus only on class I distributions, we believe that our results also hold for class II under light load. In Section 7.4.4, we characterize the exponential waiting time in more detail.

7.4.4 Characterizing the FCFS Waiting Time Distribution

In this chapter, we care about the exponential tail of the FCFS response time distribution. It turns out to be simpler to focus on the FCFS waiting time distribution, which is closely related. We will make use of two key concepts regarding the waiting time distribution. The first concept is the asymptotic exponential decay rate, as investigated in [4, 27]. We refer to this quantity as θ^* and formally define it to be the negative of the rightmost singularity of $\widetilde{T_Q^{\mathsf{FCFS}}}$. Based on the Cramer-Lundberg theory, the waiting time distribution T_Q^{FCFS} takes an asymptotic exponential tail:

$$\mathbb{P}\left\{T_Q^{\mathsf{FCFS}} > t\right\} \sim Ce^{-\theta^* t}. \tag{7.2}$$

The quantity θ^* is the least positive real solution to the equation

$$\widetilde{S}(-\theta^*) = \frac{\lambda + \theta^*}{\lambda}.$$

We also define the *normalized p.d.f.* to be

$$g(t) \triangleq f_{T_Q^{\mathsf{FCFS}}}(t) \cdot e^{\theta^* t}. \tag{7.3}$$

Note that (7.2) relates to the c.d.f. of waiting time, while (7.3) relates to the p.d.f. of waiting time.

We characterize three important properties of the normalized p.d.f., namely its maximum, minimum, and asymptotic limit. Let $g_{\max},\ g_{\min},\ g^*$ denote respectively the maximum, minimum and asymptotically limiting values of $g(\cdot)$ over $[0, \infty)$:

$$g_{\max} \triangleq \sup_{t\in[0,\infty)} g(t); \qquad g_{\min} \triangleq \inf_{t\in[0,\infty)} g(t); \qquad g^* \triangleq \lim_{t\to\infty} g(t).$$

The following lemma, implies these quantities are well defined.

Lemma 7.4.1. *Suppose S is a continuous class I job size distribution. For any load ρ, the normalized p.d.f. $g(t)$ is bounded above and below by positive constants, and $\lim_{t\to\infty} g(t)$ exists.*

Proof. First we show (following prior work [2, 3, 191]) that $\widetilde{T_Q^{\mathsf{FCFS}}}$ has a simple pole $-\theta^*$ as its rightmost singularity.

We let $-\theta^*$ be the root of the denominator of $\widetilde{T_Q^{\mathsf{FCFS}}}(s)$, which is

$$\lambda\widetilde{S}(s) - \lambda + s = 0 \iff \widetilde{S}(s) = \frac{\lambda - s}{\lambda}.$$

Since the left hand $\widetilde{S}(s)$ is convex in s[5], and the right hand $\frac{\lambda-s}{\lambda}$ is only linear in s, their intersection $s = -\theta^*$ must be a simple root. Moreover, such an intersection exists for $s < 0$ because

- $\widetilde{S}(0) = \frac{\lambda-0}{\lambda} = 1$;
- $\widetilde{S}'(0) = -\frac{1}{\mu} > -\frac{1}{\lambda}$;

[5]We have $\widetilde{S}''(s) = \int_{t=0}^{\infty} t^2 e^{-st} f_S(t)\,\mathrm{d}t > 0$ for every s in the convergence region of $\widetilde{S}(\cdot)$.

- $\widetilde{S}(s) \to \infty$ when s approaches the rightmost singularity of $\widetilde{S}$ (since S is a class I distribution).

Now we use final value theorem to establish the limit of the ratio between the p.d.f. $f_{T_Q^{\mathsf{FCFS}}}$ and the exponential function $e^{-\theta^* t}$. Recall the function $g(t) = f_{T_Q^{\mathsf{FCFS}}}(e)e^{\theta^* t}$ and consider its Laplace transform $\widetilde{G}(s) = \widetilde{T_Q^{\mathsf{FCFS}}}(s-\theta^*)$. Since the poles of $\widetilde{G}(s)$ map one-to-one to the poles of $\widetilde{T_Q^{\mathsf{FCFS}}}(s-\theta^*)$, the above arguments show that every pole of $\widetilde{G}(s)$ is either in the open left half plane or at the origin, and the origin is a simple pole. Therefore, the Final Value Theorem for $g(t)$ tells us

$$\lim_{t\to\infty} f_{T_Q}^{\mathsf{FCFS}}(t)e^{\theta^* t} = \lim_{t\to\infty} g(t) = \lim_{s\to 0} s\widetilde{G}(s) = \frac{(1-\rho)\theta^*}{-\lambda\widetilde{S}'(-\theta^*)-1} \triangleq g^* > 0. \tag{7.4}$$

Since the limit g^* exists, for $\epsilon = \frac{g^*}{2}$, there exists $N_\epsilon < \infty$ such that $\forall t \geq N_\epsilon$,

$$|g(t) - g^*| \leq \frac{g^*}{2} \;\Rightarrow\; \frac{g^*}{2}e^{-\theta^* t} \leq f_{T_Q}^{\mathsf{FCFS}}(t) \leq \frac{3g^*}{2}e^{-\theta^* t}.$$

Next, we want to show that

$$\min_{0\leq t<N_\epsilon} g(t) > 0 \quad \text{and} \quad \max_{0\leq t<N_\epsilon} g(t) < \infty. \tag{7.5}$$

First, note that $f_{T_Q}^{\mathsf{FCFS}}$ satisfies the following level-crossing differential equations [30] (we abbreviate $f_{T_Q}^{\mathsf{FCFS}}$ to f):

$$\begin{cases} f(0) = (1-\rho)\lambda; \\ f'(t) = \lambda f(t) - (1-\rho)\lambda f_S(t) - \lambda\int_{j=0}^{t} f(t-j)f_S(j)\,\mathrm{d}j. \end{cases}$$

To begin with, $f(t)$ is continuous because $f_S(t)$ is continuous. If $f(t) = 0$ for some $t < N_\epsilon$, we let $t_0 = \inf\{t < N_\epsilon : f(t) = 0\}$. Clearly $t_0 > 0$ because $f(0) = (1-\rho)\lambda > 0$. Note also that $f(t_0) = 0$, because f is continuous. Since $f(t_0) < f(0)$, $\exists 0 < s_0 < t_0$ s.t. $f_S(s_0) > 0$. Then $\exists a, b$ where $0 \leq a < s_0 < b \leq t_0$, s.t. $f(t) > 0$ for all $t \in [a, b]$. Now we have

$$f'(t_0) = -(1-\rho)\lambda f_S(t_0) - \lambda\int_{j=0}^{t_0} f(t_0-j)f_S(j)\,\mathrm{d}j < 0$$

because the first term $-(1-\rho)\lambda f_S(t_0) \leq 0$ and the second term

$$-\lambda\int_{j=0}^{t_0} f(t_0-j)f_S(j)\,\mathrm{d}j \leq -\lambda\int_{j=a}^{b} f(t_0-j)f_S(j)\,\mathrm{d}j < 0.$$

But $f'(t_0) < 0$ is impossible, because we assumed that $f(t_0) = 0$. The implication that $f'(t_0) < 0$ contradicts the fact that f is a non-negative probability density function. Therefore,

$$\min_{0\leq t<N_\epsilon} g(t) \geq \min_{0\leq t<N_\epsilon} g(t) > 0.$$

On the other hand, since $f'(t) \leq \lambda f(t)$ everywhere, we have $f(t) \leq (1-\rho)\lambda e^{\lambda t}$. Note that this bound holds even if S has infinite density at one or more points. As a result,

$$\max_{0\leq t<N_\epsilon} g(t) \leq \max_{0\leq t<N_\epsilon} \left[f_{T_Q}^{\mathsf{FCFS}}(t) e^{\theta^* N_\epsilon} \right] \leq (1-\rho)\lambda e^{\lambda N_\epsilon} e^{\theta^* N_\epsilon} < \infty.$$

Finally, note that

$$\inf_{t\in[0,\infty)} g(t) \triangleq g_{\min} \geq \min\left\{ \inf_{0\leq t<N_\epsilon} \left[f_{T_Q}^{\mathsf{FCFS}}(t) e^{\theta^* t} \right], \frac{g^*}{2} \right\} > 0$$
$$\sup_{t\in[0,\infty)} g(t) \triangleq g_{\max} \leq \max\left\{ \sup_{0\leq t<N_\epsilon} \left[f_{T_Q}^{\mathsf{FCFS}}(t) e^{\theta^* t} \right], \frac{3g^*}{2} \right\} < \infty$$

which indicates both $g_{\min}$ and $g_{\max}$ are well-defined and nonzero. This completes the proof. □

The ratio $g_{\max}/g_{\min}$ will be particularly important in our analysis. Intuitively, we can think of the ratio as measuring the deviation of the queueing time T_Q^{FCFS} from a perfect exponential distribution. The queueing time distribution is exactly an exponential distribution in an $M/M/1$, and diverges from an exponential to greater or lesser degree under any class I job size distribution. The degree of divergence will show up in our later results.

7.4.5 Scheduling Algorithm: Nudge

We now formally define the Nudge algorithm. $\mathsf{Nudge}(x_1, x_2, x_3)$ first divides jobs into four regions based on their sizes:

- "small": $0 = x_0 \leq S < x_1$.
- "medium": $x_1 \leq S < x_2$.
- "large": $x_2 \leq S < x_3$.
- "very large": $x_3 \leq S < x_4 = \infty$.

Throughout this chapter, we concentrate mostly on the "small" and the "large" jobs. For conciseness, we define $S_{\mathsf{small}}, S_{\mathsf{large}}, p_{\mathsf{small}}, p_{\mathsf{large}}$ as follows.

Definition 7.4.4. *We define S_{small} and S_{large} to be the distribution of small and large jobs, respectively. We also define p_{small} and p_{large} to be the fraction of small and large jobs, respectively.*

$$S_{\mathsf{small}} \sim [S|S < x_1], \qquad S_{\mathsf{large}} \sim [S|x_2 \leq S < x_3]$$
$$p_{\mathsf{small}} \triangleq \mathbb{P}\{S < x_1\}, \qquad p_{\mathsf{large}} \triangleq \mathbb{P}\{x_2 \leq S < x_3\}.$$

To determine which job to serve, Nudge maintains an ordering over jobs which have not yet entered service. We call this ordering the "queue". For each job, we also track whether or not it each has already been "swapped".

Whenever a job completes, Nudge serves the job at the front of the queue (if any), and serves it to completion. By default, newly arriving jobs are placed at the back of the queue, resulting in FCFS scheduling by default. However, if *three conditions* are satisfied, then a "swap" is performed. If

1. the arriving job is a small job, j_s,
2. the job at the back of queue is a large job, j_ℓ, and
3. the job j_ℓ at the back of queue has never been swapped,

then Nudge places the small job j_s just ahead of j_ℓ, in the second-to-last position in the queue. This is called a *swap*, and both j_ℓ and j_s are now marked as having been "swapped."

Because Nudge never swaps the same job twice, a job is only in the last position in the queue and eligible to be swapped immediately after it arrives. As a result, Nudge only ever swaps a job with the job that arrives immediately before or after it.

7.5 Main Results

7.5.1 Nudge Improves upon FCFS Non-Asymptotically

Our main goal is to show that Nudge stochastically improves upon FCFS. Nudge's performance crucially depends on the choice of parameters x_1, x_2, and x_3, i.e. which jobs are small and which jobs are large. We begin by asking: given job size distribution S and load ρ, for what choices of parameters x_1, x_2, and x_3 does Nudge stochastically improve upon FCFS? We answer this in Theorem 7.5.1, which gives sufficient conditions on the parameters for Nudge to stochastically improve upon FCFS. We prove Theorem 7.5.1 in Section 7.6.

Theorem 7.5.1 (Stochastic Improvement Regime)**.** *Suppose S is a continuous class I job size distribution. Then* $\mathsf{Nudge}(x_1, x_2, x_3)$ *stochastically improves upon* FCFS *for any $s_{\min} < x_1 \leq x_2 \leq x_3$ satisfying*[6]

$$\bullet \quad \frac{g_{\max}}{g_{\min}} \frac{\lambda + \theta^*}{\lambda} < \frac{1 - \widetilde{S_{\mathsf{large}}}(-\theta^*)^{-1}}{1 - \widetilde{S_{\mathsf{small}}}(-\theta^*)^{-1}}, \tag{7.6}$$

$$\bullet \quad x_1 + x_3 \leq 2x_2. \tag{7.7}$$

With Theorem 7.5.1 in hand, our goal reduces to the following question: given S and ρ, do there exist parameters satisfying the sufficient condition from Theorem 7.5.1? We answer this in Theorem 7.5.2, showing that as long as the minimum job size $s_{\min} = 0$, such parameters always exist. Our proof of Theorem 7.5.2 in Section 7.7 gives a simple construction of those parameters.

Theorem 7.5.2 (Existence of Stochastic Improvement)**.** *For any continuous class I job size distribution S with $s_{\min} = 0$ and any load $0 < \rho < 1$, there exist x_1, x_2, x_3 satisfying* (7.6) *and* (7.7)*, implying that* $\mathsf{Nudge}(x_1, x_2, x_3)$ *stochastically improves upon* FCFS.

7.5.2 Nudge Improves upon FCFS Asymptotically

Having shown that Nudge stochastically improves upon FCFS, we ask: is Nudge's improvement non-negligible in the asymptotic limit? We answer this in Theorem 7.5.3. Specifically, recall that in the $t \to \infty$ limit, $\mathbb{P}\left\{T^{\mathsf{FCFS}} > t\right\} \sim C_{\mathsf{FCFS}} e^{-\theta^* t}$. We show that $\mathbb{P}\left\{T^{\mathsf{Nudge}} > t\right\} \sim C_{\mathsf{Nudge}} e^{-\theta^* t}$

[6]Recall from Definition 7.4.4 that S_{small} and S_{large} depend on x_1, x_2, and x_3. This applies throughout this chapter.

and that, with appropriately set parameters, $C_{\mathsf{Nudge}} < C_{\mathsf{FCFS}}$. This implies that FCFS is *not* strongly optimal for asymptotic tail behavior (see Section 7.3.1), resolving a long-standing open problem [27, 232]. We also exactly derive the difference $C_{\mathsf{FCFS}} - C_{\mathsf{Nudge}}$. We prove Theorem 7.5.3 in Section 7.9, making use of Theorem 7.5.4.

Theorem 7.5.3 (Asymptotic Improvement Regime)**.** *Suppose S is a continuous class I job size distribution. For any $s_{\min} < x_1 \leq x_2 \leq x_3$, the asymptotic tail improvement ratio of* $\mathsf{Nudge}(x_1, x_2, x_3)$ *compared to* FCFS *is*

$$\mathsf{AsymTIR} = p_{\mathsf{small}} p_{\mathsf{large}} \frac{\lambda}{\lambda + \theta^*} \left(\widetilde{S_{\mathsf{large}}}(-\theta^*) - \frac{\lambda}{\lambda + \theta^*} \widetilde{S_{\mathsf{small}}}(-\theta^*) - \frac{\theta^*}{\lambda + \theta^*} \widetilde{S_{\mathsf{large}}}(-\theta^*) \widetilde{S_{\mathsf{small}}}(-\theta^*) \right).$$

Furthermore, AsymTIR *is positive, meaning $C_{\mathsf{Nudge}} < C_{\mathsf{FCFS}}$, if*

$$\frac{\lambda + \theta^*}{\lambda} < \frac{1 - \widetilde{S_{\mathsf{large}}}(-\theta^*)^{-1}}{1 - \widetilde{S_{\mathsf{small}}}(-\theta^*)^{-1}}.$$

Note that the asymptotic improvement regime in Theorem 7.5.3 is a superset of the non-asymptotic improvement regime in Theorem 7.5.1, because $g_{\max}/g_{\min} \geq 1$. Thus, whenever Theorem 7.5.1 guarantees a stochastic improvement, we also have $C_{\mathsf{FCFS}} > C_{\mathsf{Nudge}}$. Thus, by Theorem 7.5.2, there exists an asymptotic improvement whenever $s_{\min} = 0$.

Corollary 7.5.1 (Existence of Asymptotic Improvement)**.** *For any continuous class I job size distribution S with $s_{\min} = 0$ and any load $0 < \rho < 1$, there exist x_1, x_2, x_3 such that $C_{\mathsf{Nudge}} < C_{\mathsf{FCFS}}$.*

While Theorem 7.5.3 shows that there is a multiplicative improvement in the asymptotic tail, we find empirically that the same multiplicative improvement exists throughout nearly the entire tail. See Fig. 7.3 and Section 7.10.

7.5.3 Exact Analysis of Nudge

All of the above results compare Nudge's performance to that of FCFS. In particular, none of these results give an exact analysis of Nudge's response time. We give such an analysis in Theorem 7.5.4, in which we exactly derive $\widetilde{T^{\mathsf{Nudge}}}(s)$. This result is nontrivial, because Nudge does not fall into any class of policies with known analyses (see Section 7.3.3). We prove Theorem 7.5.4 in Section 7.8.

Theorem 7.5.4 (Transform of Response Time)**.** *The response time of Nudge has Laplace-Stieltjes transform*

$$\widetilde{T^{\mathsf{Nudge}}}(s) = \widetilde{T^{\mathsf{FCFS}}}(s) + p_{\mathsf{small}} p_{\mathsf{large}} \left(\widetilde{S_{\mathsf{large}}}(s)(1 - \widetilde{S_{\mathsf{small}}}(s)) \left(\widetilde{T_Q^{\mathsf{FCFS}}}(\lambda + s) - \widetilde{T_Q^{\mathsf{FCFS}}}(s) \right) \right.$$
$$\left. + \widetilde{S_{\mathsf{small}}}(s)(1 - \widetilde{S_{\mathsf{large}}}(s)) \left(\frac{\widetilde{T_Q^{\mathsf{FCFS}}}(s)}{\widetilde{S}(s)} - (1 - \rho) \frac{\lambda/\widetilde{S}(\lambda) - s/\widetilde{S}(s)}{\lambda - s} \right) \right).$$

7.6 Proof of Theorem 7.5.1: Stochastic Improvement Regime

Our goal in this section is to prove Theorem 7.5.1, which gives sufficient conditions on the parameters x_1, x_2, and x_3 for Nudge to stochastically improve upon FCFS. To do so, we employ a tagged job approach. In particular, we follow an arbitrary tagged job i making its way through a pair of coupled systems, one employing the FCFS policy and one employing the Nudge policy, both with the same arrival process and job sizes.

We focus on one particular response time threshold t, and in particular on the events $D_{i,t}$ and $I_{i,t}$, where the tagged job i has response time greater than t in one system and below in the other system. In (7.8), we write the difference in the response time tails of Nudge and FCFS in terms of the events $D_{i,t}$ and $I_{i,t}$. In Lemma 7.6.1, we derive formulas for the probabilities of these events.

Using these formulas, in Lemma 7.6.2, we derive a sufficient condition for Nudge to improve upon FCFS relative to a specific threshold t. This sufficient condition is dependent on the threshold t. In order to remove this dependence, we prove Lemma 7.6.3, a technical lemma regarding arbitrary random variables.

Finally, in Section 7.6.2, we prove Theorem 7.5.1, by demonstrating that the conditions given in Theorem 7.5.1 ensure that the sufficient condition in Lemma 7.6.2 holds relative to every response time threshold t, making use of Lemma 7.6.3 to do so.

7.6.1 Intermediate Lemmas

Consider a tagged job i that arrives into the steady state of the pair of coupled systems. We write T_i^{Nudge} and T_i^{FCFS} for job i's response time in the Nudge and FCFS systems, respectively. For any $t \geq 0$, define the events

$$I_{i,t} \triangleq \left\{T_i^{\mathsf{FCFS}} \leq t < T_i^{\mathsf{Nudge}}\right\}, \qquad D_{i,t} \triangleq \left\{T_i^{\mathsf{Nudge}} \leq t < T_i^{\mathsf{FCFS}}\right\}.$$

Intuitively, $D_{i,t}$ is the event in which Nudge *decreases* job i's response time relative to FCFS, specifically from above t to below t. Similarly, $I_{i,t}$ is the event in which Nudge *increases* job i's response time relative to FCFS. We can write

$$\mathbb{P}\left\{T_i^{\mathsf{Nudge}} > t\right\} = \mathbb{P}\left\{T_i^{\mathsf{FCFS}} > t\right\} + \mathbb{P}\left\{I_{i,t}\right\} - \mathbb{P}\left\{D_{i,t}\right\}. \tag{7.8}$$

The events $D_{i,t}$ and $I_{i,t}$ are defined using the Nudge and FCFS systems. Our next step is to express them in terms of only the FCFS system, which we understand well.

We begin by defining the relevant quantities in the FCFS system. Let i^- be the arrival immediately before job i, and let i^+ be the arrival immediately after, and let

$W_i \triangleq$ amount of work in the system (either Nudge or FCFS) when job i arrives,

$A_i \triangleq$ interarrival time between jobs i and i^+,

$S_i \triangleq$ size of job i.

We define analogous quantities for i^- and i^+. The work is the same in both systems because both Nudge and FCFS are work-conserving.

Note that Nudge will only ever swap job i with one of the adjacent arrivals, i^- or i^+ (see Section 7.4.5). Under what condition do we swap job i with job i^+? This happens if and only if the following events occur:

(a) Job i is large, which is when $x_2 \leq S_i < x_3$.
(b) Job i^+ is small, which is when $S_{i^+} < x_1$.
(c) Job i^+ arrives before job i enters service in the Nudge system.
(d) Job i has not swapped with any other job, namely job i^-.

Because job i cannot be both large and small, (a) implies (d). But (d) implies that (c) happens when $A_i \leq W_i$. This is because in the absence of swaps, job i would enter service after W_i time. Therefore, the event that job i swaps with job i^+ is

$$\mathsf{swap}_{i,i^+} \triangleq \{(x_2 \leq S_i < x_3) \ \wedge \ (S_{i^+} \leq x_1) \ \wedge \ (A_i \leq W_i)\}. \tag{7.9}$$

Crucially, this definition of swap_{i,i^+} depends only on quantities in the FCFS system. We define $\mathsf{swap}_{i^-,i}$ analogously.

Lemma 7.6.1 (Evaluating $\mathbb{P}\{I_{i,t}\}$ and $\mathbb{P}\{D_{i,t}\}$)**.** *We have*

$$\mathbb{P}\{I_{i,t}\} = \mathbb{P}\left\{\mathsf{swap}_{i,i^+} \ \wedge \ (W_i + S_i \leq t < W_i + S_i + S_{i^+})\right\}, \tag{7.10}$$
$$\mathbb{P}\{D_{i,t}\} = \mathbb{P}\left\{\mathsf{swap}_{i^-,i} \ \wedge \ (W_{i^-} - A_{i^-} + S_i \leq t < W_{i^-} - A_{i^-} + S_i + S_{i^-})\right\}. \tag{7.11}$$

Proof. We begin by computing $\mathbb{P}\{I_{i,t}\}$. The event $I_{i,t}$ occurs only if $T_i^{\mathsf{Nudge}} > T_i^{\mathsf{FCFS}}$, which in turn occurs only if job i swaps with the next arrival, namely job i^+. If this swap occurs, then $T_i^{\mathsf{Nudge}} = T_i^{\mathsf{FCFS}} + S_{i^+}$. We know that $T_i^{\mathsf{FCFS}} = W_i + S_i$, so (7.10) follows from

$$\begin{aligned} I_{i,t} &= \mathsf{swap}_{i,i^+} \ \wedge \ (T_i^{\mathsf{FCFS}} \leq t < T_i^{\mathsf{Nudge}}) \\ &= \mathsf{swap}_{i,i^+} \ \wedge \ (T_i^{\mathsf{FCFS}} \leq t < T_i^{\mathsf{FCFS}} + S_{i^+}) \\ &= \mathsf{swap}_{i,i^+} \ \wedge \ (W_i + S_i \leq t < W_i + S_i + S_{i^+}). \end{aligned}$$

We now compute $\mathbb{P}\{D_{i,t}\}$. By similar reasoning to the above, the event $D_{i,t}$ occurs only if job i swaps with job i^-. If this swap occurs, then $T_i^{\mathsf{Nudge}} = T_i^{\mathsf{FCFS}} - S_{i^-}$. We again have $T_i^{\mathsf{FCFS}} = W_i + S_i$, so

$$\begin{aligned} D_{i,t} &= \mathsf{swap}_{i^-,i} \ \wedge \ (T_i^{\mathsf{Nudge}} \leq t < T_i^{\mathsf{FCFS}}) \\ &= \mathsf{swap}_{i^-,i} \ \wedge \ (T_i^{\mathsf{FCFS}} - S_{i^-} \leq t < T_i^{\mathsf{FCFS}}) \\ &= \mathsf{swap}_{i^-,i} \ \wedge \ (W_i + S_i - S_{i^-} \leq t < W_i + S_i). \end{aligned}$$

To obtain (7.11), observe that conditioned on $\mathsf{swap}_{i^-,i}$, we have $W_i = W_{i-} + S_{i^-} - A_{i^-}$. □

Now, we give sufficient conditions for Nudge to improve upon FCFS relative to a particular threshold t.

Lemma 7.6.2 (Strict Improvement at a Given Threshold)**.** *Given any $t > s_{\min}$, where $s_{\min}$ is the smallest value of S,*

$$\mathbb{P}\left\{T^{\mathsf{Nudge}} > t\right\} < \mathbb{P}\left\{T^{\mathsf{FCFS}} > t\right\}$$

if the following inequality in terms of t holds:

$$\frac{g_{\max}}{g_{\min}}\frac{\lambda+\theta^*}{\lambda} < \frac{\mathbb{E}\left[e^{\theta^*\min(t,S_{\mathsf{small}}+S_{\mathsf{large}})} - e^{\theta^*\min(t,S_{\mathsf{small}})}\right]}{\mathbb{E}\left[e^{\theta^*\min(t,S_{\mathsf{small}}+S_{\mathsf{large}})} - e^{\theta^*\min(t,S_{\mathsf{large}})}\right]}. \tag{7.12}$$

Proof. Because the tagged job i is a random sample arriving to the steady state of the system, by (7.8), we have $\mathbb{P}\left\{T^{\mathsf{Nudge}} > t\right\} < \mathbb{P}\left\{T^{\mathsf{FCFS}} > t\right\}$ if and only if $\mathbb{P}\left\{I_{i,t}\right\} < \mathbb{P}\left\{D_{i,t}\right\}$. Our approach is to use Lemma 7.6.1 to bound each of $\mathbb{P}\left\{I_{i,t}\right\}$ and $\mathbb{P}\left\{D_{i,t}\right\}$, from which the desired sufficient condition follows.

We begin by computing $\mathbb{P}\left\{I_{i,t}\right\}$:

$$\begin{aligned}
\mathbb{P}\left\{I_{i,t}\right\} &= \mathbb{P}\left\{\mathsf{swap}_{i,i^+} \ \wedge \ (W_i + S_i \le t < W_i + S_i + S_{i^+})\right\} && \text{[by Lemma 7.6.1]}\\
&= \mathbb{P}\left\{(A_i \le W_i) \wedge (W_i + S_i \le t < W_i + S_i + S_{i^+}) \wedge (x_2 \le S_i < x_3) \wedge (S_{i^+} < x_1)\right\} && \text{[by (7.9)]}\\
&\le \mathbb{P}\left\{(W_i + S_i \le t < W_i + S_i + S_{i^+}) \wedge (x_2 \le S_i < x_3) \wedge (S_{i^+} < x_1)\right\} && \text{[discarding } A_i \le W_i\text{]}\\
&= \mathbb{P}\left\{\left((t - S_i - S_{i^+})^+ \le W_i \le (t - S_i)^+\right) \wedge (x_2 \le S_i < x_3) \wedge (S_{i^+} < x_1)\right\}\\
&= p_{\mathsf{large}}p_{\mathsf{small}} \cdot \mathbf{E}_{S_i\sim S_{\mathsf{large}}, S_{i^+}\sim S_{\mathsf{small}}}\left[\int_{w=(t-S_i-S_{i^+})^+}^{(t-S_i)^+} f_{W_i}(w)\,\mathrm{d}w\right] && \left[\begin{array}{l}\text{change of measure for } S_i\\ \text{and } S_i^+\text{, independence of}\\ S_i,\ S_{i^+}\text{, and } W_i\end{array}\right]\\
&\le p_{\mathsf{large}}p_{\mathsf{small}} \cdot \mathbf{E}_{S_i\sim S_{\mathsf{large}}, S_{i^+}\sim S_{\mathsf{small}}}\left[\int_{w=(t-S_i-S_{i^+})^+}^{(t-S_i)^+} g_{\max}e^{-\theta^* w}\,\mathrm{d}w\right] && \left[\begin{array}{l}\text{by Lemma 7.4.1 and}\\ \text{the fact that } W_i \sim\\ T_Q^{\mathsf{FCFS}}\end{array}\right]\\
&= p_{\mathsf{large}}p_{\mathsf{small}} \cdot \frac{e^{-\theta^* t}}{\theta^*} \cdot g_{\max}\mathbb{E}\left[e^{\theta^*\min(t,S_{\mathsf{small}}+S_{\mathsf{large}})} - e^{\theta^*\min(t,S_{\mathsf{large}})}\right]. && (7.13)
\end{aligned}$$

Similarly, we compute $\mathbb{P}\{D_{i,t}\}$:

$$\begin{aligned}
&\mathbb{P}\{D_{i,t}\} = \mathbb{P}\left\{\mathsf{swap}_{i^-,i} \ \wedge \ (W_{i^-} - A_{i^-} + S_i \le t < W_{i^-} - A_{i^-} + S_i + S_{i^-})\right\} \\
&= \mathbb{P}\left\{\begin{array}{l} (A_{i^-} \le W_{i^-}) \wedge (W_{i^-} - A_{i^-} + S_i \le t < W_{i^-} - A_{i^-} + S_i + S_{i^-}) \\ \wedge (S_i < x_1) \wedge (x_2 \le S_{i^-} < x_3) \end{array}\right\} \\
&= \mathbb{P}\left\{\left(A_{i^-} + (t - S_i - S_{i^-})^+ \le W_{i^-} \le A_{i^-} + (t - S_i)^+\right) \ \wedge \ (S_i < x_1) \ \wedge \ (x_2 \le S_{i^-} < x_3)\right\} \\
&= p_{\mathsf{small}} p_{\mathsf{large}} \cdot \mathbf{E}_{S_i \sim S_{\mathsf{small}}, S_{i^-} \sim S_{\mathsf{large}}, A_{i^-} \sim \mathrm{Exp}(\lambda)} \left[\int_{w=A_{i^-}+(t-S_i-S_{i^-})^+}^{A_{i^-}+(t-S_i)^+} f_{W_{i^-}}(w)\,\mathrm{d}w\right] \\
&\qquad\qquad \text{[mutual independence of } S_i,\ S_{i^+},\ A_{i^-}, \text{ and } W_i\text{]} \\
&\ge p_{\mathsf{small}} p_{\mathsf{large}} \cdot \mathbf{E}_{S_i \sim S_{\mathsf{small}}, S_{i^-} \sim S_{\mathsf{large}}, A_{i^-} \sim \mathrm{Exp}(\lambda)} \left[\int_{w=A_{i^-}+(t-S_i-S_{i^-})^+}^{A_{i^-}+(t-S_i)^+} g_{\min} e^{-\theta^* w}\,\mathrm{d}w\right] \\
&\qquad\qquad \text{[by Lemma 7.4.1 and the fact that } W_i \sim T_Q^{\mathsf{FCFS}}\text{]} \\
&\ge p_{\mathsf{small}} p_{\mathsf{large}} \cdot \mathbf{E}_{S_i \sim S_{\mathsf{small}}, S_{i^-} \sim S_{\mathsf{large}}} \left[\int_{a=0}^{\infty} \int_{w=a+(t-S_i-S_{i^-})^+}^{a+(t-S_i)^+} g_{\min} e^{-\theta^* w} \cdot \lambda e^{-\lambda a}\,\mathrm{d}w\,\mathrm{d}a\right] \\
&\qquad\qquad [A_{i^-} \sim \mathrm{Exp}(\lambda)] \\
&= p_{\mathsf{small}} p_{\mathsf{large}} \cdot \frac{e^{-\theta^* t}}{\theta^*} \cdot g_{\min} \frac{\lambda}{\lambda + \theta^*} \mathbb{E}\left[e^{\theta^* \min(t, S_{\mathsf{small}} + S_{\mathsf{large}})} - e^{\theta^* \min(t, S_{\mathsf{small}})}\right]. \qquad (7.14)
\end{aligned}$$

Combining the bounds (7.13) and (7.14), we find that $\mathbb{P}\{I_{i,t}\} < \mathbb{P}\{D_{i,t}\}$ holds if

$$g_{\max} \mathbb{E}\left[e^{\theta^* \min(t, S_{\mathsf{small}} + S_{\mathsf{large}})} - e^{\theta^* \min(t, S_{\mathsf{large}})}\right] < g_{\min} \frac{\lambda}{\lambda + \theta^*} \mathbb{E}\left[e^{\theta^* \min(t, S_{\mathsf{small}} + S_{\mathsf{large}})} - e^{\theta^* \min(t, S_{\mathsf{small}})}\right]. \qquad (7.15)$$

□

Having proven Lemma 7.6.2, we have a sufficient condition, namely (7.12), for Nudge to improve upon FCFS at a specific value of t. But our goal is to improve upon FCFS for *all* values of t. We therefore seek a condition which implies that (7.12) holds for all t.

We start by simplifying (7.12). Let $A = e^{\theta^* S_{\mathsf{small}}}$, $B = e^{\theta^* S_{\mathsf{large}}}$, and $c = e^{\theta^* t}$. Then (7.12) becomes

$$\frac{g_{\max}}{g_{\min}} \frac{\lambda + \theta^*}{\lambda} < \frac{\mathbb{E}\left[\min(AB, c) - \min(A, c)\right]}{\mathbb{E}\left[\min(AB, c) - \min(B, c)\right]}. \qquad (7.16)$$

Here the only appearance of the specific value of t is via c. Our strategy is to lower-bound the right-hand side of (7.16) by a quantity that does not include c. The following lemma helps accomplish this under an additional assumption.

Lemma 7.6.3. *Let A, B be two independent real-valued random variables and c be a fixed constant. Suppose $1 \le A \le c$ and $A < B$. Under these assumptions, if $\mathbb{P}\{B > c\} > 0$ and*

$$c\mathbb{E}[B] \ge \mathbb{E}[A]\,\mathbb{E}[B | B > c], \qquad (7.17)$$

then

$$\frac{\mathbb{E}\left[\min(AB, c) - \min(A, c)\right]}{\mathbb{E}\left[\min(AB, c) - \min(B, c)\right]} = \frac{\mathbb{E}\left[\min(AB, c) - A\right]}{\mathbb{E}\left[\min(AB, c) - \min(B, c)\right]} \ge \frac{\mathbb{E}[AB - A]}{\mathbb{E}[AB - B]}. \qquad (7.18)$$

Proof. First we observe $\mathbb{E}[\min(AB, c) - A] > \mathbb{E}[\min(AB, c) - \min(B, c)]$ because $A < \min(B, c)$. Based on this, we can shrink the left hand side of inequality (7.18) by adding the same positive term to both the denominator and numerator. We compute

$$\frac{\mathbb{E}[\min(AB, c) - A]}{\mathbb{E}[\min(AB, c) - \min(B, c)]} = \frac{\mathbb{E}[AB - A] - \mathbb{E}[(AB - c)\mathbb{1}AB > c]}{\mathbb{E}[AB - B] - \mathbb{E}[(AB - c)\mathbb{1}AB > c] + \mathbb{E}[(B - c)\mathbb{1}B > c]}. \tag{7.19}$$

Since $A \geq 1$ and $B > 0$, we have $AB \geq B$. Therefore,

$$0 \leq \mathbb{1}AB > c - \mathbb{1}B > c \leq \mathbb{1}AB > c = \mathbb{1}AB - c > 0.$$

We proceed by adding a positive term, $\mathbb{E}[(AB - c)(\mathbb{1}AB > c - \mathbb{1}B > c)]$, to both the denominator and numerator of the right hand side of (7.19) and obtain

$$\frac{\mathbb{E}[\min(AB, c) - A]}{\mathbb{E}[\min(AB, c) - \min(B, c)]} \geq \frac{\mathbb{E}[AB - A] - \mathbb{E}[(AB - c)\mathbb{1}B > c]}{\mathbb{E}[AB - B] - \mathbb{E}[(AB - B)\mathbb{1}B > c]}.$$

Hence, to establish inequality (7.18), it suffices to show

$$\begin{aligned}
&\frac{\mathbb{E}[(AB - c)\mathbb{1}B > c]}{\mathbb{E}[(AB - B)\mathbb{1}B > c]} \geq \frac{\mathbb{E}[AB - A]}{\mathbb{E}[AB - B]} \\
\iff &\mathbb{E}[(AB - c)\mathbb{1}B > c]\,\mathbb{E}[AB - B] \geq \mathbb{E}[(AB - B)\mathbb{1}B > c]\,\mathbb{E}[AB - A] \\
\iff &(\mathbb{E}[A]\,\mathbb{E}[B|B > c] - c)\,\mathbb{P}\{B > c\}\,\mathbb{E}[AB - B] \\
&\quad \geq (\mathbb{E}[A] - 1)\,\mathbb{E}[B|B > c]\,\mathbb{P}\{B > c\}\,\mathbb{E}[AB - A] \\
\iff &c\mathbb{E}[B](\mathbb{E}[A] - 1) \geq \mathbb{E}[A]\,\mathbb{E}[B|B > c](\mathbb{E}[A] - 1) \\
\iff &c\mathbb{E}[B] \geq \mathbb{E}[A]\,\mathbb{E}[B|B > c],
\end{aligned}$$

which is precisely the condition provided in (7.17). □

7.6.2 Main Proof

Theorem 7.5.1. Suppose S is a continuous class I job size distribution. Then $\mathsf{Nudge}(x_1, x_2, x_3)$ stochastically improves upon FCFS for any $s_{\min} < x_1 \leq x_2 \leq x_3$ satisfying

$$\bullet \quad \frac{g_{\max}}{g_{\min}} \frac{\lambda + \theta^*}{\lambda} < \frac{1 - \widetilde{S_{\mathsf{large}}}(-\theta^*)^{-1}}{1 - \widetilde{S_{\mathsf{small}}}(-\theta^*)^{-1}}, \tag{7.20}$$

$$\bullet \quad x_1 + x_3 \leq 2x_2. \tag{7.21}$$

Proof. We prove Theorem 7.5.1 by verifying the condition in Lemma 7.6.2. For every $t > s_{\min}$, we will show that Inequalities (7.20) and (7.21) together imply (7.12).

(i) When $s_{\min} < t \leq x_2$, the denominator of the right hand side of (7.12) becomes

$$\mathbb{E}\left[e^{\theta^* \min(t, S_{\mathsf{small}} + S_{\mathsf{large}})} - e^{\theta^* \min(t, S_{\mathsf{large}})}\right] = \mathbb{E}\left[e^t - e^t\right] = 0.$$

Thus (7.12) is not well defined, but (7.15) holds trivially, which is sufficient.

(ii) When $x_2 < t < x_3$, we let $A = e^{\theta^* S_{\mathsf{small}}}$, $B = e^{\theta^* S_{\mathsf{large}}}$ and $c = e^{\theta^* t}$. Then clearly $1 \le A \le c$ and $A < B$. By (7.21), we know that

$$\begin{aligned} c\mathbb{E}[B] &= e^{\theta^* t}\mathbb{E}\left[e^{\theta^* S_{\mathsf{large}}}\right] \ge e^{\theta^*(2x_2)} \ge e^{\theta^*(x_1+x_3)} \\ &\ge \mathbb{E}\left[e^{\theta^*(S_{\mathsf{small}}+S_{\mathsf{large}})}\right] \ge \mathbb{E}\left[e^{\theta^* \min(S_{\mathsf{small}}+S_{\mathsf{large}},t)}\right] \ge \mathbb{E}[A]\,\mathbb{E}[B|B>c]. \end{aligned}$$

We can therefore apply Lemma 7.6.3 to obtain

$$\frac{\mathbb{E}\left[e^{\theta^* \min(S_{\mathsf{small}}+S_{\mathsf{large}},t)} - e^{\theta^* S_{\mathsf{small}}}\right]}{\mathbb{E}\left[e^{\theta^* \min(S_{\mathsf{small}}+S_{\mathsf{large}},t)} - e^{\theta^* \min(S_{\mathsf{large}},t)}\right]} \ge \frac{\mathbb{E}\left[e^{\theta^*(S_{\mathsf{small}}+S_{\mathsf{large}})} - e^{\theta^* S_{\mathsf{small}}}\right]}{\mathbb{E}\left[e^{\theta^*(S_{\mathsf{small}}+S_{\mathsf{large}})} - e^{\theta^* S_{\mathsf{large}}}\right]}. \tag{7.22}$$

Moreover, condition (7.20) implies that

$$\frac{g_{\max}}{g_{\min}}\frac{\lambda+\theta^*}{\lambda} < \frac{1-\widetilde{S_{\mathsf{large}}}(-\theta^*)^{-1}}{1-\widetilde{S_{\mathsf{small}}}(-\theta^*)^{-1}} = \frac{\mathbb{E}\left[e^{\theta^*(S_{\mathsf{small}}+S_{\mathsf{large}})} - e^{\theta^* S_{\mathsf{small}}}\right]}{\mathbb{E}\left[e^{\theta^*(S_{\mathsf{small}}+S_{\mathsf{large}})} - e^{\theta^* S_{\mathsf{large}}}\right]} \tag{7.23}$$

Combining (7.22) with (7.23) establishes (7.12).

(iii) When $t \ge x_3$, we have $\min(t, S_{\mathsf{small}}) = S_{\mathsf{small}}$ and $\min(t, S_{\mathsf{large}}) = S_{\mathsf{large}}$. Therefore, condition (7.20) is equivalent to (7.12).

Therefore, for every $t > s_{\min}$, we have proven that $\mathbb{P}\left\{T^{\mathsf{Nudge}} > t\right\} < \mathbb{P}\left\{T^{\mathsf{FCFS}} > t\right\}$. □

7.7 Proof of Theorem 7.5.2: Existence of Stochastic Improvement

Theorem 7.5.2. For any continuous class I job size distribution S with $s_{\min} = 0$ and any load $0 < \rho < 1$, there exist x_1, x_2, x_3 satisfying (7.20) and (7.21), implying that $\mathsf{Nudge}(x_1, x_2, x_3)$ stochastically improves upon FCFS.

Proof. We start by constructing x_1, x_2, x_3 that satisfy both Inequality (7.20) and (7.21). For notational convenience, let $M = \frac{g_{\max}}{g_{\min}}\frac{\lambda+\theta^*}{\lambda}$. First fix an arbitrary $x_3 > 0$ and let $x_2 = \frac{3}{4}x_3$, then compute $\widetilde{S_{\mathsf{large}}}(-\theta^*)$ and choose a small enough x_1 such that

$$x_1 < \min\left\{-\frac{1}{\theta^*}\ln\left(1 - \frac{1-\widetilde{S_{\mathsf{large}}}(-\theta^*)^{-1}}{M}\right), \frac{1}{2}x_3\right\}. \tag{7.24}$$

Clearly, such $x_1 > s_{\min} = 0$ in (7.24) exists because $M > 1$, so we have

- $x_1 + x_3 < \dfrac{3}{2}x_3 = 2x_2,$ (7.25)
- $\dfrac{1-\widetilde{S_{\mathsf{large}}}(-\theta^*)^{-1}}{1-\widetilde{S_{\mathsf{small}}}(-\theta^*)^{-1}} = \dfrac{1-\widetilde{S_{\mathsf{large}}}(-\theta^*)^{-1}}{1-\mathbb{E}\left[e^{\theta^* S_{\mathsf{small}}}\right]^{-1}} \ge \dfrac{1-\widetilde{S_{\mathsf{large}}}(-\theta^*)^{-1}}{1-e^{-\theta^* x_1}} > M = \dfrac{g_{\max}}{g_{\min}}\dfrac{\lambda+\theta^*}{\lambda}.$ (7.26)

By Theorem 7.5.1, (7.25) and (7.26) together imply that $\mathbb{P}\left\{T^{\mathsf{Nudge}} \ge t\right\} < \mathbb{P}\left\{T^{\mathsf{FCFS}} \ge t\right\}$ for every $t > s_{\min} = 0$. Therefore,

$$\forall t > 0, \quad \mathbb{P}\left\{T^{\mathsf{Nudge}} > t\right\} < \mathbb{P}\left\{T^{\mathsf{FCFS}} > t\right\}. \qquad \square$$

7.8 Proof of Theorem 7.5.4: Transform of Response Time

In this section we compute an exact formula for $\widetilde{T^{\mathsf{Nudge}}}(s)$. The formula holds for arbitrary job size distributions, not just those of class I.

At a high level, our analysis works by considering two systems experiencing identical arrivals: one using Nudge, and one using FCFS. We consider a tagged job arriving to this pair of systems in equilibrium and determine how its Nudge queueing time relates to its FCFS queueing time.

- *Small jobs:* Nudge queueing time is FCFS queueing time, possibly minus a large job's size.
- *Large jobs:* Nudge queueing time is FCFS queueing time, possibly plus a small job's size.
- *Other jobs:* Nudge queueing time is identical to FCFS queueing time.

We will determine $\widetilde{T^{\mathsf{Nudge}}_{Q,\mathsf{small}}}(s)$ and $\widetilde{T^{\mathsf{Nudge}}_{Q,\mathsf{large}}}(s)$, from which $\widetilde{T^{\mathsf{Nudge}}}(s)$ easily follows.

7.8.1 Probabilistic Interpretation of the Laplace-Stieltjes Transform

Before jumping into the Nudge queueing time analysis, we recall a probabilistic interpretation of the Laplace-Stieltjes transform.

Let V be a nonnegative random variable. Consider a time interval of length V and a Poisson process of rate s that is independent of V. We call the increments of the Poisson process "interruptions". Let $\mathsf{NoPoisson}(V, s)$ be the event that there are no interruptions during the time interval. Then [104, Exercise 25.7]

$$\widetilde{V}(s) = \mathbb{P}\left\{\mathsf{NoPoisson}(V, s)\right\}. \tag{7.27}$$

The interpretation in (7.27) necessarily requires $s \geq 0$. Fortunately, formulas we derive using (7.27) are still valid for $s < 0$ because Laplace transforms are uniquely defined by their value on any bounded interval on the real line [34].

7.8.2 Transform for Large Jobs

Lemma 7.8.1. *The queueing time of large jobs under Nudge has Laplace-Stieltjes transform*

$$\widetilde{T^{\mathsf{Nudge}}_{Q,\mathsf{large}}}(s) = \big(1 - p_{\mathsf{small}}(1 - \widetilde{S_{\mathsf{small}}}(s))\big)\widetilde{T^{\mathsf{FCFS}}_{Q}}(s) + p_{\mathsf{small}}(1 - \widetilde{S_{\mathsf{small}}}(s))\widetilde{T^{\mathsf{FCFS}}_{Q}}(\lambda + s).$$

Proof. Consider a large tagged job arriving to the pair of systems, one using Nudge and the other using FCFS, in equilibrium. We can think of the job's Nudge queueing time as the time it takes to do the following two steps:

(a) We first wait for its FCFS queueing time, namely T^{FCFS}_{Q}.

(b) If during that T^{FCFS}_{Q} time there has been at least one arrival, and if the first such arrival is a small job, we then wait for that small job's service, which takes S_{small} time. Note that the small job's size is independent of the FCFS queueing time.

We will use (7.27) to compute $\widetilde{T^{\mathsf{Nudge}}_{Q,\mathsf{large}}}(s)$. To that end, we associate each of the Nudge and FCFS systems with a Poisson "interruption" process of rate s. The interruption processes are independent of the arrival times and job sizes of each system, but they are coupled to each other in the following way. At any moment in time when the systems are busy, some job j has been in service for some amount of time t. We couple the interruption processes such that interruptions occur at the same (j,t) pairs in both systems.

By (7.27), $\widetilde{T^{\mathsf{Nudge}}_{Q,\mathsf{large}}}(s)$ is the probability that no interruptions occur during steps (a) and (b). We compute this probability by conditioning on the following event:

$$E = \begin{Bmatrix} \text{the next arrival after the tagged job is small,} \\ \text{and an interruption occurs during its service} \end{Bmatrix}$$

Note that E does not consider whether the next arrival occurs before the tagged job exits the queue. Therefore, it is independent of the length T^{FCFS}_Q of step (a).

If E does not occur, then there are no interruptions during step (b). Therefore, there are no interruptions if and only if there are no interruptions during step (a). By (7.27), this happens with probability $\widetilde{T^{\mathsf{FCFS}}_Q}(s)$.

If E does occur, then an interruption will occur during step (b) if and only if a new job arrives during step (a). That is, by conditioning on E, we have predetermined that the next arrival will be small and, if it swaps with the tagged job, will cause an interruption. Therefore, to avoid interruptions, we need to avoid interruptions *and arrivals* during step (a). Merging the arrival and interruption processes yields a Poisson process of rate $\lambda+s$, so avoiding interruptions corresponds to the event $\mathsf{NoPoisson}(T^{\mathsf{FCFS}}_Q, \lambda+s)$. By (7.27), this happens with probability $\widetilde{T^{\mathsf{FCFS}}_Q}(\lambda+s)$.

Conditioning on whether E occurs and using (7.27) to compute $\mathbb{P}\{E\} = p_{\mathsf{small}}(1-\widetilde{S_{\mathsf{small}}}(s))$, we obtain the desired expression. □

7.8.3 Transform for Small Jobs

Lemma 7.8.2. *The queueing time of small jobs under Nudge has Laplace-Stieltjes transform*

$$\widetilde{T^{\mathsf{Nudge}}_{Q,\mathsf{small}}}(s) = \widetilde{T^{\mathsf{FCFS}}_Q}(s)\left(1+\frac{p_{\mathsf{large}}(1-\widetilde{S_{\mathsf{large}}}(s))}{\widetilde{S}(s)}\right) - p_{\mathsf{large}}(1-\widetilde{S_{\mathsf{large}}}(s))(1-\rho)\cdot\frac{\lambda/\widetilde{S}(\lambda)-s/\widetilde{S}(s)}{\lambda-s}.$$

The analysis of small jobs is more involved than the analysis of large jobs. We therefore state and prove several more intermediate results before proving Lemma 7.8.2.

Consider a small tagged job arriving to the pair of systems, one using Nudge and the other using FCFS, in equilibrium. The main question we need to answer is whether the tagged job will swap with a large job in the Nudge system. Our main insight is that we can tell whether the swap will occur by examining just the FCFS system. Because we understand FCFS well, this makes it relatively simple to tell whether a swap will occur.

Lemma 7.8.3. *The small tagged job swaps with a large job in the Nudge system if and only if, when it arrives, the FCFS system has a nonempty queue whose last job is large.*

The proof of Lemma 7.8.3 follows very similar reasoning to our analysis at the start of Section 7.6.1.

Proof. By definition of Nudge, the tagged job swaps if and only if, when it arrives, the Nudge system has a nonempty queue whose last job is a large job that has not been swapped. It therefore suffices to show that at any moment in time, the FCFS system has a nonempty queue whose last job is large if and only if the Nudge system has a nonempty queue whose last job is a large job that has not been swapped.

We first note that the total amount of work in both systems is the same at every moment in time, because both FCFS and Nudge are work conserving.

Suppose the FCFS system has a nonempty queue whose last job j is large. Because it is the last job in the FCFS queue, there have been no new arrivals after j. In the Nudge system, this means j has not been swapped, so either j is the last job in the Nudge queue or has entered service. By work conservation, both systems had the same amount of work when j arrived, so j must still be in the Nudge queue, as desired.

Suppose the Nudge system has a nonempty queue whose last job j is a large job that has not been swapped. We argue similarly to the previous direction: there have been no arrivals since j because it is at the end of the Nudge queue without being swapped, and j cannot have entered service in the FCFS system by work conservation, so j must be the last job in the FCFS queue, as desired. □

Thanks to Lemma 7.8.3, we can determine the queueing time of the small tagged job by looking at the state of the FCFS system when it arrives. We describe the equilibrium state of the FCFS system with the following quantities:

- W: the amount of work in the system.
- N_Q: the number of jobs in the queue.
- W_{most}: the amount of work in the system, excluding the last job in the queue if $N_Q \geq 1$.
- S_{last}: the size of the last job in the queue, or 0 if $N_Q = 0$.

Note that these quantities are not independent. In particular, $W = W_{\mathsf{most}} + S_{\mathsf{last}}$. However, W_{most} and S_{last} are *conditionally* independent given $N_Q \geq 1$.

Armed with Lemma 7.8.3 and the system state notation, we are ready to compute $T^{\mathsf{Nudge}}_{Q,\mathsf{small}}(s)$, thus proving Lemma 7.8.2. Our computation will make use of an additional lemma which we state after the proof.

Proof of Lemma 7.8.2. Consider the small tagged job arriving to the pair of systems in equilibrium. By Lemma 7.8.3, its Nudge queueing time is

$$T^{\mathsf{Nudge}}_{Q,\mathsf{small}} = \begin{cases} W_{\mathsf{most}} & \text{if } N_Q = 0 \\ W_{\mathsf{most}} + S_{\mathsf{last}}\mathbf{1}(\neg(x_2 \leq S_{\mathsf{last}} < x_3)) & \text{if } N_Q \geq 1. \end{cases}$$

Applying (7.27) and the conditional independence of W_{most} and S_{last} yields

$$\begin{aligned}\widetilde{T^{\mathsf{Nudge}}_{Q,\mathsf{small}}}(s) &= \mathbb{P}\left\{\mathsf{NoPoisson}(W_{\mathsf{most}}, s) \ \wedge \ N_Q = 0\right\} \\ &\quad + \mathbb{P}\left\{\mathsf{NoPoisson}(W_{\mathsf{most}}, s) \ \wedge \ N_Q \geq 1 \ \wedge \ (\mathsf{NoPoisson}(S_{\mathsf{last}}, s) \ \vee \ x_2 \leq S_{\mathsf{last}} < x_3)\right\} \\ &= \mathbb{P}\left\{\mathsf{NoPoisson}(W_{\mathsf{most}}, s) \ \wedge \ N_Q = 0\right\} \\ &\quad + \mathbb{P}\left\{\mathsf{NoPoisson}(W_{\mathsf{most}}, s) \ \wedge \ N_Q \geq 1\right\} \cdot \left(\widetilde{S}(s) + p_{\mathsf{large}}(1 - \widetilde{S_{\mathsf{large}}}(s))\right). \end{aligned} \tag{7.28}$$

It remains only to compute the two probabilities in (7.28). Let

$$q \triangleq \mathbb{P}\left\{\mathsf{NoPoisson}(W, s) \ \wedge \ N_Q = 0\right\} = \mathbb{P}\left\{\mathsf{NoPoisson}(W_{\mathsf{most}}, s) \ \wedge \ N_Q = 0\right\}, \tag{7.29}$$

making q the first probability in (7.28). We now compute the second probability in (7.28) in terms of q. First, note that T_Q^{FCFS} and W are identically distributed. Recalling the conditional independence of W_{most} and S_{last}, we have, using (7.27) throughout,

$$\begin{aligned}\widetilde{T_Q^{\mathsf{FCFS}}}(s) - q = \widetilde{W}(s) - q &= \mathbb{P}\left\{\mathsf{NoPoisson}(W, s) \ \wedge \ N_Q \geq 1\right\} \\ &= \mathbb{P}\left\{\mathsf{NoPoisson}(W_{\mathsf{most}}, s) \ \wedge \ N_Q \geq 1 \ \wedge \ \mathsf{NoPoisson}(S_{\mathsf{last}}, s)\right\} \\ &= \mathbb{P}\left\{\mathsf{NoPoisson}(W_{\mathsf{most}}, s) \ \wedge \ N_Q \geq 1\right\} \cdot \widetilde{S}(s). \end{aligned} \tag{7.30}$$

Plugging (7.29) and (7.30) into (7.28) yields

$$\widetilde{T^{\mathsf{Nudge}}_{Q,\mathsf{small}}}(s) = q + \left(\widetilde{T_Q^{\mathsf{FCFS}}}(s) - q\right)\left(1 + \frac{p_{\mathsf{large}}(1 - \widetilde{S_{\mathsf{large}}}(s))}{\widetilde{S}(s)}\right).$$

Lemma 7.8.4 below computes the value of q, yielding the desired result. □

Lemma 7.8.4. *Let* $q \triangleq \mathbb{P}\left\{\mathsf{NoPoisson}(W, s) \ \wedge \ N_Q = 0\right\}$. *We have*

$$q = \widetilde{T_Q^{\mathsf{FCFS}}}(\lambda) \cdot \frac{\lambda\widetilde{S}(s) - s\widetilde{S}(\lambda)}{\lambda - s} = \frac{1-\rho}{\widetilde{S}(\lambda)} \cdot \frac{\lambda\widetilde{S}(s) - s\widetilde{S}(\lambda)}{\lambda - s}.$$

To prove Lemma 7.8.4, we require an additional lemma.

Lemma 7.8.5. *Let* V *be a nonnegative random variable, and let* $\mathrm{Exp}(r)$ *and* $\mathrm{Exp}(s)$ *be exponentially distributed random variables of rates* r *and* s*, respectively. Suppose* V*,* $\mathrm{Exp}(r)$*, and* $\mathrm{Exp}(s)$ *are mutually independent. Then*

$$\mathbb{P}\left\{V < \mathrm{Exp}(r) + \mathrm{Exp}(s)\right\} = \frac{r\widetilde{V}(s) - s\widetilde{V}(r)}{r - s}.$$

Proof. We compute

$$\begin{aligned}
\mathbb{P}\left\{V < \mathrm{Exp}(r) + \mathrm{Exp}(s)\right\} &= \int_{v=0}^{\infty} \mathbb{P}\left\{v < \mathrm{Exp}(r) + \mathrm{Exp}(s)\right\} f_V(v)\,\mathrm{d}v \\
&= \int_{v=0}^{\infty} \left(\int_{u=0}^{\infty} \int_{t=0}^{\infty} \mathbf{1}(v < t+u) \cdot re^{-rt} \cdot se^{-su}\,\mathrm{d}t\,\mathrm{d}u \right) f_V(v)\,\mathrm{d}v \\
&= \int_{v=0}^{\infty} \frac{re^{-rv} - se^{-sv}}{r-s} f_V(v)\,\mathrm{d}v \\
&= \frac{r\widetilde{V}(s) - s\widetilde{V}(r)}{r-s}.
\end{aligned}$$

□

Proof of Lemma 7.8.4. Consider a FCFS system in equilibrium along with an independent Poisson "interruption" process of rate s. Call a job *lucky* if it enters the system while $N_Q = 0$ and experiences no interruptions during its queueing time. Because Poisson arrivals see time averages [235], q is probability an arriving job is lucky.

We compute q in an unusual way. Let a job's *departure period* be the time interval starting when the job enters service and ending when the next job enters service. Jobs enter service at average rate λ, so q is the average number of lucky jobs that arrive a departure period. More formally, by renewal-reward theorem,

$$\begin{aligned}
q &= \mathbb{P}\left\{\text{arrival is lucky}\right\} \\
&= \frac{\text{average rate of lucky arrivals}}{\lambda} \\
&= \frac{\text{average rate of lucky arrivals}}{\text{average rate of departure periods}} \\
&= \mathbb{E}\left[\text{number of lucky arrivals during a departure period}\right].
\end{aligned}$$

Moreover, because a job is lucky only if $N_Q = 0$, only the first arrival in a departure period can possibly be lucky, so

$$q = \mathbb{P}\left\{\text{first arrival in a departure period is lucky}\right\}.$$

Consider a job j. The first arrival in j's departure period is lucky if both of the following events occur:

$E_1 =$ there are no arrivals during j's queueing time

$E_2 = j$ completes before the first interruption after the first arrival of j's departure period.

By (7.27), $\mathbb{P}\left\{E_1\right\} = \widetilde{T_Q^{\mathsf{FCFS}}}(\lambda)$. We compute $\mathbb{P}\left\{E_2 \mid E_1\right\}$ below.

Conditioned on E_1, the queue is empty when j enters service, so the first arrival during j's departure period is simply the first arrival after j enters service. Let $\mathrm{Exp}(\lambda)$ be the amount of time between when j enters service and the next arrival, and let $\mathrm{Exp}(s)$ be the amount of time between that next arrival and the first interruption after it. Both $\mathrm{Exp}(\lambda)$ and $\mathrm{Exp}(s)$ are

exponentially distributed with rates λ and s, respectively, and they and j's size are mutually independent. Because j's size is distributed as S, we have

$$\mathbb{P}\{E_2 \mid E_1\} = \mathbb{P}\{S < \text{Exp}(\lambda) + \text{Exp}(s)\} = \frac{\lambda\widetilde{S}(s) - s\widetilde{S}(\lambda)}{\lambda - s},$$

where the latter equality follows from Lemma 7.8.5.

It remains only to show $\widetilde{T_Q^{\mathsf{FCFS}}}(\lambda) = (1-\rho)/\widetilde{S}(\lambda)$. Because response time T^{FCFS} is a sum of independent random variables with distributions T_Q^{FCFS} and S, we have $\widetilde{T_Q^{\mathsf{FCFS}}}(\lambda) = \widetilde{T^{\mathsf{FCFS}}}(\lambda)/\widetilde{S}(\lambda)$. By (7.27), $\widetilde{T^{\mathsf{FCFS}}}(\lambda)$ is the probability that no arrivals occur during a job's response time. But this is simply the probability that a job leaves an empty system when it departs, which is $1-\rho$. □

7.8.4 Overall Response Time Transform

Theorem 7.5.4. The response time of Nudge has Laplace-Stieltjes transform

$$\widetilde{T^{\mathsf{Nudge}}}(s) = \widetilde{T^{\mathsf{FCFS}}}(s) + p_{\mathsf{small}}p_{\mathsf{large}}\left(\widetilde{S_{\mathsf{large}}}(s)(1-\widetilde{S_{\mathsf{small}}}(s))\left(\widetilde{T_Q^{\mathsf{FCFS}}}(\lambda+s) - \widetilde{T_Q^{\mathsf{FCFS}}}(s)\right)\right.$$
$$\left. + \widetilde{S_{\mathsf{small}}}(s)(1-\widetilde{S_{\mathsf{large}}}(s))\left(\frac{\widetilde{T_Q^{\mathsf{FCFS}}}(s)}{\widetilde{S}(s)} - (1-\rho)\frac{\lambda/\widetilde{S}(\lambda) - s/\widetilde{S}(s)}{\lambda - s}\right)\right).$$

Proof. The expression follows by plugging the results of Lemmas 7.8.1 and 7.8.2 into

$$\widetilde{T^{\mathsf{Nudge}}}(s) = p_{\mathsf{small}} \cdot \widetilde{T^{\mathsf{Nudge}}_{Q,\mathsf{small}}}(s) \cdot \widetilde{S_{\mathsf{small}}}(s) + p_{\mathsf{large}} \cdot \widetilde{T^{\mathsf{Nudge}}_{Q,\mathsf{large}}}(s) \cdot \widetilde{S_{\mathsf{large}}}(s)$$
$$+ (1 - p_{\mathsf{small}} - p_{\mathsf{large}}) \cdot \widetilde{T_Q^{\mathsf{FCFS}}}(s) \cdot \frac{\widetilde{S}(s) - p_{\mathsf{small}}\widetilde{S_{\mathsf{small}}}(s) - p_{\mathsf{large}}\widetilde{S_{\mathsf{large}}}(s)}{1 - p_{\mathsf{small}} - p_{\mathsf{large}}}$$

and simplifying the resulting expression.

One key step is recognizing that $\widetilde{T^{\mathsf{FCFS}}}(s) = \widetilde{T_Q^{\mathsf{FCFS}}}(s) \cdot \widetilde{S}(s)$. □

7.9 Proof of Theorem 7.5.3: Asymptotic Improvement

Theorem 7.5.3. Suppose S is a continuous class I job size distribution. For any $s_{\min} < x_1 \leq x_2 \leq x_3$, the asymptotic tail improvement ratio of $\mathsf{Nudge}(x_1, x_2, x_3)$ compared to FCFS is

$$\mathsf{AsymTIR} = p_{\mathsf{small}}p_{\mathsf{large}}\frac{\lambda}{\lambda+\theta^*}\left(\widetilde{S_{\mathsf{large}}}(-\theta^*) - \frac{\lambda}{\lambda+\theta^*}\widetilde{S_{\mathsf{small}}}(-\theta^*) - \frac{\theta^*}{\lambda+\theta^*}\widetilde{S_{\mathsf{large}}}(-\theta^*)\widetilde{S_{\mathsf{small}}}(-\theta^*)\right).$$

Furthermore, $\mathsf{AsymTIR}$ is positive, meaning $C_{\mathsf{Nudge}} < C_{\mathsf{FCFS}}$, if

$$\frac{\lambda+\theta^*}{\lambda} < \frac{1 - \widetilde{S_{\mathsf{large}}}(-\theta^*)^{-1}}{1 - \widetilde{S_{\mathsf{small}}}(-\theta^*)^{-1}}.$$

Below we give a high-level overview of the proof, followed by the full proof.

Proof sketch. Using the Final Value Theorem, one can show that for $\mathsf{Alg} \in \{\mathsf{Nudge}, \mathsf{FCFS}\}$,

$$C_{\mathsf{Alg}} = \frac{1}{\theta^*} \lim_{s \to 0} s\widetilde{T^{\mathsf{Alg}}}(s - \theta^*).$$

Combining this with Theorem 7.5.4, which relates $\widetilde{T^{\mathsf{Nudge}}}(s)$ to $\widetilde{T^{\mathsf{FCFS}}}(s)$, will relate C_{Nudge} to C_{FCFS}.

To obtain $\mathsf{AsymTIR} = 1 - C_{\mathsf{Nudge}}/C_{\mathsf{FCFS}}$, we compute $\lim_{s\to 0}(s\widetilde{T^{\mathsf{FCFS}}}(s - \theta^*) - s\widetilde{T^{\mathsf{Nudge}}}(s - \theta^*))$ we compute $\lim_{s\to 0}(s\widetilde{T^{\mathsf{Nudge}}}(s - \theta^*))$ in terms of C_{FCFS} via Theorem 7.5.4. Each non-vanishing term has an $s\widetilde{T_Q^{\mathsf{FCFS}}}(s - \theta^*)$ factor. Because $\widetilde{T_Q^{\mathsf{FCFS}}}(s) = \widetilde{T^{\mathsf{FCFS}}}(s)/\widetilde{S}(s)$, we can express $C_{\mathsf{FCFS}} - C_{\mathsf{Nudge}}$ as a constant times C_{FCFS}; this constant is $\mathsf{AsymTIR}$. □

Proof. We prove this theorem using the Laplace-Stieltjes transform derived in Theorem 7.5.4.

The transform of the tail of T^{Nudge} can be calculated as

$$\begin{aligned}
\int_{t=0}^{\infty} e^{-st}\mathbb{P}\left\{T^{\mathsf{Nudge}} > t\right\} \mathrm{d}t &= -\int_{t=0}^{\infty} \mathbb{P}\left\{T^{\mathsf{Nudge}} > t\right\} \mathrm{d}\left(\frac{e^{-st}}{s}\right) \\
&= -\mathbb{P}\left\{T^{\mathsf{Nudge}} > t\right\}\left(\frac{e^{-st}}{s}\right)\Bigg|_0^{\infty} + \int_{t=0}^{\infty} \frac{e^{-st}}{s} \mathrm{d}\mathbb{P}\left\{T^{\mathsf{Nudge}} > t\right\} \\
&= \frac{1}{s}\left(1 - \int_{t=0}^{\infty} e^{-st} f_{T^{\mathsf{Nudge}}}(t)\,\mathrm{d}t\right) \\
&= \frac{1 - \widetilde{T^{\mathsf{Nudge}}}(s)}{s}. \qquad (7.31)
\end{aligned}$$

Then the transform of $e^{\theta^* t}\mathbb{P}\left\{T^{\mathsf{Nudge}} > t\right\}$ is obtained by translating (7.31) horizontally through θ^* units:

$$\int_{t=0}^{\infty} e^{-st}\left(e^{\theta^* t}\mathbb{P}\left\{T^{\mathsf{Nudge}} > t\right\}\right)\mathrm{d}t = \frac{1 - \widetilde{T^{\mathsf{Nudge}}}(s - \theta^*)}{s - \theta^*}.$$

Now, we are ready to calculate C_{Nudge} using this transform. From Final Value Theorem,

$$\begin{aligned}
C_{\mathsf{Nudge}} &= \lim_{t\to\infty} e^{\theta^* t}\mathbb{P}\left\{T^{\mathsf{Nudge}} > t\right\} \\
&= \lim_{s\to 0} s\frac{1 - \widetilde{T^{\mathsf{Nudge}}}(s - \theta^*)}{s - \theta^*} \\
&= \frac{1}{\theta^*}\lim_{s\to 0} s\widetilde{T^{\mathsf{Nudge}}}(s - \theta^*)
\end{aligned}$$

Now, we substitute the expression from Theorem 7.5.4, and drop terms that are negligible in $s \to 0$ limit.

$$\begin{aligned}
&\frac{1}{\theta^*}\lim_{s\to 0} s\widetilde{T^{\mathsf{Nudge}}}(s - \theta^*) = C_{\mathsf{FCFS}} \\
&+ \frac{1}{\theta^*} p_{\mathsf{small}} p_{\mathsf{large}} \left(\widetilde{S_{\mathsf{large}}}(-\theta^*)(\widetilde{S_{\mathsf{small}}}(-\theta^*) - 1)C_{Q,\mathsf{FCFS}} + \widetilde{S_{\mathsf{small}}}(-\theta^*)(1 - \widetilde{S_{\mathsf{large}}}(-\theta^*))\frac{C_{Q,\mathsf{FCFS}}}{\widetilde{S}(-\theta^*)}\right), \qquad (7.32)
\end{aligned}$$

where

$$C_{Q,\mathsf{FCFS}} = \lim_{s\to 0} s\widetilde{T_Q^{\mathsf{FCFS}}}(s-\theta^*) = \lim_{s\to 0} s\frac{\widetilde{T^{\mathsf{FCFS}}}(s-\theta^*)}{\widetilde{S}(s-\theta^*)} = \frac{\theta^* C_{\mathsf{FCFS}}}{\widetilde{S}(-\theta^*)}. \tag{7.33}$$

We recall that $-\theta^*$ is the rightmost singularity of $\widetilde{T_Q^{\mathsf{FCFS}}}(s) = \frac{(1-\rho)s}{\lambda\widetilde{S}(s)-\lambda+s}$, which indicates that θ^* is the smallest positive value that satisfies

$$\lambda\widetilde{S}(-\theta^*) - \lambda - \theta^* = 0 \quad \text{and} \quad \widetilde{S}(-\theta^*) = \frac{\lambda+\theta^*}{\lambda}. \tag{7.34}$$

Using (7.33) and (7.34) to simplify (7.32), we obtain

$$\begin{aligned} &C_{\mathsf{Nudge}} \\ &= C_{\mathsf{FCFS}}\left(1 - p_{\mathsf{small}}p_{\mathsf{large}}\frac{\lambda}{\lambda+\theta^*}\left(\widetilde{S_{\mathsf{large}}}(-\theta^*) - \frac{\lambda}{\lambda+\theta^*}\widetilde{S_{\mathsf{small}}}(-\theta^*) - \frac{\theta^*}{\lambda+\theta^*}\widetilde{S_{\mathsf{large}}}(-\theta^*)\widetilde{S_{\mathsf{small}}}(-\theta^*)\right)\right). \end{aligned}$$

This gives us

$$\begin{aligned} \mathsf{AsymTIR} &= 1 - \frac{C_{\mathsf{Nudge}}}{C_{\mathsf{FCFS}}} \\ &= p_{\mathsf{small}}p_{\mathsf{large}}\frac{\lambda}{\lambda+\theta^*}\left(\widetilde{S_{\mathsf{large}}}(-\theta^*) - \frac{\lambda}{\lambda+\theta^*}\widetilde{S_{\mathsf{small}}}(-\theta^*) - \frac{\theta^*}{\lambda+\theta^*}\widetilde{S_{\mathsf{large}}}(-\theta^*)\widetilde{S_{\mathsf{small}}}(-\theta^*)\right). \end{aligned}$$

By assumption,

$$\frac{\lambda+\theta^*}{\lambda} < \frac{1-\widetilde{S_{\mathsf{large}}}(-\theta^*)^{-1}}{1-\widetilde{S_{\mathsf{small}}}(-\theta^*)^{-1}},$$

so we have

$$\begin{aligned} \mathsf{AsymTIR} &= p_{\mathsf{small}}p_{\mathsf{large}}\frac{\lambda}{\lambda+\theta^*}\widetilde{S_{\mathsf{large}}}(-\theta^*)\widetilde{S_{\mathsf{small}}}(-\theta^*)\left(\widetilde{S_{\mathsf{small}}}(-\theta^*)^{-1} - \frac{\lambda}{\lambda+\theta^*}\widetilde{S_{\mathsf{large}}}(-\theta^*)^{-1} - \frac{\theta^*}{\lambda+\theta^*}\right) \\ &= p_{\mathsf{small}}p_{\mathsf{large}}\frac{\lambda}{\lambda+\theta^*}\widetilde{S_{\mathsf{large}}}(-\theta^*)\widetilde{S_{\mathsf{small}}}(-\theta^*)\left(\frac{\lambda}{\lambda+\theta^*}\left(1-\widetilde{S_{\mathsf{large}}}(-\theta^*)^{-1}\right) - \left(1-\widetilde{S_{\mathsf{small}}}(-\theta^*)^{-1}\right)\right) \\ &> 0. \end{aligned}$$

Hence $C_{\mathsf{Nudge}} < C_{\mathsf{FCFS}}$. □

7.10 Empirical Lessons

This chapter proves that Nudge stochastically improves upon FCFS under the correct choice of parameters, and achieves multiplicative improvement in the asymptotic tail. However, there are a few practical questions remaining. These questions center around finding Nudge parameters in practice. In this section, we demonstrate several practical lessons on choosing Nudge parameters.

1. (Section 7.10.1) We find that Nudge typically achieves its greatest improvement over FCFS when the Nudge parameters specify that all jobs are either large or small (i.e. $x_1 = x_2$, $x_3 = \infty$).

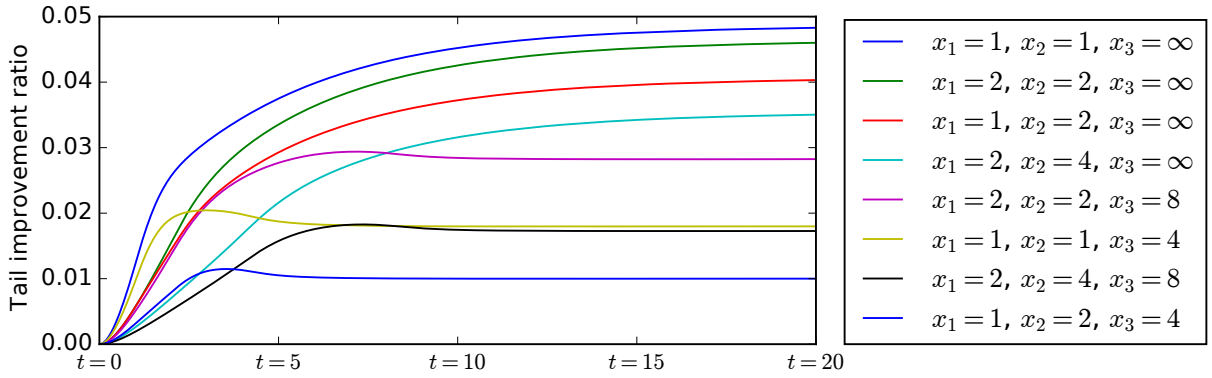


Figure 7.4: Empirical tail improvement of Nudge over FCFS under a variety of Nudge parameter choices. Highest improvement occurs when $x_1 = x_2$, $x_3 = \infty$. Job size distribution is hyperexponential with branches drawn from $\mathrm{Exp}(2)$ and $\mathrm{Exp}(1/3)$, where the first branch has probability 0.8. $\mathbb{E}[S] = 1$, $C^2 = 3$. Simulations run with 10 billion arrivals. Load $\rho = 0.8$. Parameter choices are listed in order of asymptotic improvement.

2. (Section 7.10.2) We find that when load is low, Nudge can dramatically improve upon FCFS (10-20%) in the common case where job size variability is relatively high (i.e. $C^2 > 1$). When job size variability is lower and load is low, improvement is smaller.
3. (Section 7.10.3) We find that the space of parameters that lead Nudge to *asymptotically* improve upon FCFS typically also cause Nudge to *stochastically* improve upon FCFS. This is serendipitous, because Theorem 7.5.3 provides a simple, exact method to check whether given Nudge parameters will achieve asymptotic improvement over FCFS.

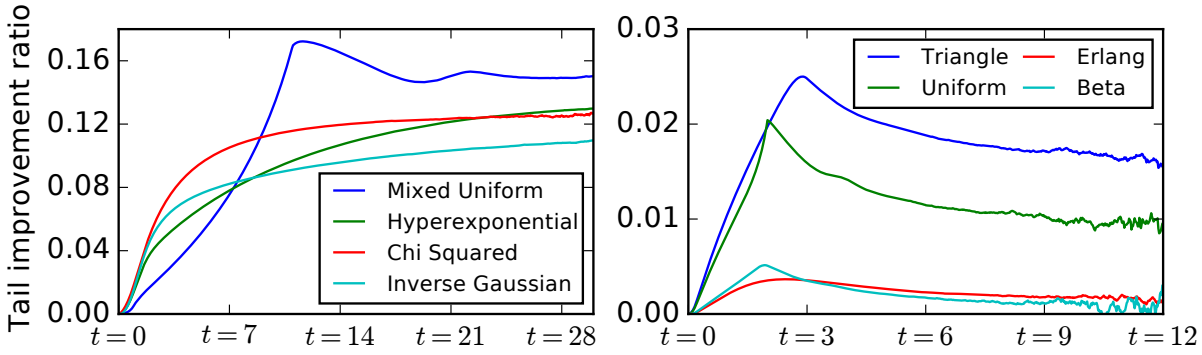
7.10.1 All Jobs Should Be Either Large or Small

When evaluating Nudge on common job size distributions, we have found that the greatest improvement over FCFS is achieved by setting the Nudge parameters such that all jobs are either large or small (i.e. $x_1 = x_2$, $x_3 = \infty$), with no medium or very large jobs. This is a pattern we have seen with great consistency across a variety of job size distributions.

In Fig. 7.4, we show one instance of this pattern, for the case of a particular hyperexponential distribution. We see that the two choices of Nudge parameters that display the *least* improvement over FCFS are those where both medium and very large jobs exist, i.e. $x_1 \neq x_2$ and $x_3 \neq \infty$.

To explain this phenomenon, note that when we remove medium and very large jobs, we end up with more swaps. Empirically, we have found that the *quantity* of swaps is more important than the *quality* of those swaps, and thus maximizing the number of swaps leads to the largest improvement. While empirically removing medium and very large jobs improves performance, our analytical result in Theorem 7.5.2 requires medium and very large jobs.

Setting the Nudge parameters so that all jobs are either large or small dramatically simplifies the problem of choosing Nudge parameters, in addition to achieving consistently strong performance. Now, only *one free parameter* remains: the cutoff between small and large jobs.



(a) High variance: $C^2 > 1$; $x_1 = x_2 = 1, x_3 = \infty$ (b) Low variance: $C^2 < 1$; $x_1 = x_2 = 0.2, x_3 = \infty$

Figure 7.5: Empirical tail improvement of Nudge over FCFS at low load ($\rho = 0.4$) under a variety of job size distributions with $\mathbb{E}[S] = 1$. (a) Higher variance distributions show dramatic improvement. (b) Lower variance distributions show modest improvement. Specific distributions: In (a), Mixed Uniform: Uniform$(0, 1)$ w/prob. 0.9, else Uniform$(0, 11)$, $C^2 = 3.33$; Hyperexponential: Exp(2) w/ prob. 0.8, else Exp$(1/3)$, $C^2 = 3$; ChiSquared(1), $C^2 = 2$; InverseGaussian($\mu = 1, \lambda = 1/2$), $C^2 = 2$. In (b), Triangle w/ min $= 0$, mode $= 0$, max $= 3$, $C^2 = 1/2$; Uniform$(0, 2)$, $C^2 = 1/3$; Erlang($k = 3$, $\lambda = 1/3$, $C^2 = 1/3$; Beta($\alpha = 2, \beta = 2$) scaled by a factor of 2, $C^2 = 1/5$. Distributions listed in order of asymptotic improvement. Simulations run with 10 billion arrivals.

7.10.2 Low Load: Dramatic Improvement when Variability is High

At low load, Nudge has the potential for dramatic improvement over FCFS ($> 10\%$ throughout the tail), in the common case where the job size distribution is more variable than an exponential distribution, i.e. $C^2 > 1$. On the other hand, under low-variability job size distributions ($C^2 < 1$), we find that Nudge's improvement shrinks at lower loads; here it helps to set the x_1 cutoff close to 0.

In Fig. 7.5 we show these patterns for a wide variety of distributions at relatively low load $\rho = 0.4$. In Fig. 7.5(a), we have four high-variance job size distributions, each with $C^2 \in [2, 3.33]$. In every case, Nudge dramatically improves upon FCFS, with TIR in the range of 10-15%. In Fig. 7.5(b), we have four low-variance job size distributions, each with $C^2 \in [1/5, 1/2]$. In these cases, we reduce the cutoff x_1 to 0.2 for best performance, and Nudge's improvement over FCFS is under 3%.

Intuitively, when load is low, each job waits behind fewer other jobs on average, so Nudge's one swap per job has a greater relative impact. When those swaps are broadly beneficial for the tail, as occurs when job size variance is high, Nudge achieves the most dramatic improvement over FCFS. When job size variance is low, swaps involving small jobs that are near the mean job size cause the response time of the large jobs to suffer too much. To alleviate this, we reduce the small job cutoff x_1 to maintain stochastic improvement over FCFS.

Job size dist.	x_1	Asym.	Stoc.	x_1	Asym.	Stoc.	x_1	Asym.	Stoc.	x_1	Asym.	Stoc.	x_1	Asym.	Stoc.
Exponential	0.5	✓	✓	1	✓	✓	2	×	×	4	×	×	8	×	×
Hyperexponential	0.5	✓	✓	1	✓	✓	2	✓	✓	4	✓	✓	8	✓	✓
Bounded Lomax	0.5	✓	✓	1	✓	✓	1.5	×	×	2	×	×	3	×	×
Uniform	0.1	✓	✓	0.2	✓	✓	0.5	×	×	0.75	×	×	1	×	×
Beta	0.1	✓	✓	0.2	✓	✓	0.3	×	×	0.4	×	×	0.5	×	×

Table 7.1: Presence or absence of asymptotic and stochastic improvement of Nudge over FCFS under a variety of job size distributions and Nudge parameter choices. Stochastic improvement occurs whenever asymptotic improvement occurs. Each row gives a distinct job size distribution, and each cell gives a distinct Nudge parameter setting. In every case, $x_2 = x_1$ and $x_3 = \infty$, so only x_1 is specified. Load $\rho = 0.4$. Specific job size distributions, each with mean 1: Exponential with mean 1, Uniform(0, 2), Hyperexponential drawn from $\mathrm{Exp}(2)$ w/ prob. 0.8 and $\mathrm{Exp}(1/3)$ with prob. 0.2, BoundedLomax($\lambda = 2, \max = 4, \alpha = 2$), Beta($\alpha = 2, \beta = 2$) scaled by a factor of 2.

7.10.3 Asymptotic Improvement Means Stochastic Improvement

After extensively simulating Nudge under different loads and job size distributions, we have found that the space of parameters under which Nudge asymptotically improves upon FCFS typically matches the space under which Nudge stochastically improves upon FCFS.

In Table 7.1, we show the consistency of this relationship across a wide variety of job size distributions and choices of Nudge parameters. The distributions range from a low-variance Beta distribution with $C^2 = 1/5$ to a hyperexponential distribution with $C^2 = 3$. Across the spectrum, Nudge stochastically improves over FCFS whenever it asymptotically improves over FCFS.

This connection between asymptotic and stochastic improvement is surprising given that the conditions that we need to prove stochastic improvement (Theorem 7.5.1) are much more stringent than what we need to prove asymptotic improvement (Theorem 7.5.3). Nonetheless, the connection is highly useful because we have provided a simple analytical formula for determining when Nudge asymptotically improves upon FCFS (Theorem 7.5.3).

7.11 Nudge in practice

Nudge can be used in practice even if some of the assumptions made in this chapter are not perfectly satisfied.

In this chapter, we assume that exact job size information is known to the scheduler. However, such information is only used to determine which size class (small, large, etc.) a job should be placed in. In practice, only estimates of job size may be known. In such a setting, the scheduler could assign jobs that are clearly above or below a size threshold to the large and small classes, while placing ambiguous jobs in the medium class. If the estimates are reasonably accurate, we would expect such a Nudge policy to stochastically improve upon FCFS.

We also assume that the exact job size distribution is known to the scheduler. This assumption is needed to choose the Nudge parameters for our proofs in Section 7.5. However, our empirical

results in Section 7.10 show that much less information is needed in practice to choose good Nudge parameters. For instance, as we saw in Section 7.10.2, the following choice of parameters works well empirically: By default, set $x_1 = x_2 = \mathbb{E}[S], x_3 = \infty$. However, if load is low and job size variability (C^2) is low, set $x_1 = x_2 = \mathbb{E}[S]/5, x_3 = \infty$.

7.12 Variants on Nudge

As Nudge is such a simple policy, there are many interesting variants of Nudge that one could investigate. We now discuss the advisability of several such variants.

Recall that Nudge only ever swaps a job at most once. One might consider allowing a job to swap a second or third time with new arrivals, or even an unlimited number of times. Unfortunately, this change could ruin Nudge's stochastic improvement over FCFS, if implemented poorly. In particular, under a Nudge variant where large jobs can be swapped with an unlimited number of small arrivals, such highly-swapped large jobs will typically dominate the response time tail, dramatically worsening the variant's tail performance. A wiser variant might be to allow large jobs to be swapped with a bounded number of small jobs, or to allow only the small jobs to be swapped an unlimited number of times.

Another interesting variant of Nudge would only swap in a probabilistic fashion, such as with an i.i.d. coin flip. We believe such a policy could achieve stochastic improvement over FCFS. However, proving such a result would be no simpler than for Nudge, because probabilistic swapping does not change the shape of the distribution of swaps. Moreover, the variant's tail improvement ratios would likely be smaller than those of Nudge, because a smaller fraction of jobs are involved in swaps.

Finally, one could design a more complicated variant of Nudge which would consider a job's exact size when deciding whether to swap, rather than simply comparing the job's size to a threshold. For instance, one might decide to swap all pairs of jobs whose sizes differ by a factor of 2, as long as neither job has yet been swapped. These more complicated Nudge variants might achieve even larger stochastic improvements over FCFS than Nudge. Beyond FCFS, such Nudge variants might be able to stochastically improve upon some or even all Nudge policies. We leave this possibility as an open question.

7.13 Technical Conclusion

We introduce Nudge, the first scheduling policy whose response time distribution stochastically improves upon that of FCFS. Specifically, we prove that with appropriately chosen parameters, Nudge stochastically improves upon FCFS for light-tailed job size distributions[7]. From an asymptotic viewpoint, we prove that Nudge achieves a multiplicative improvement over FCFS, disproving the strong asymptotic optimality conjecture for FCFS. Finally, we derive the Laplace-Stieltjes transform of response time under Nudge, using a novel technique. Nudge is simple to implement and is a practical drop-in replacement for FCFS when job sizes are known.

[7]More specifically, continuous class I job size distributions with positive density at 0.

One of the major insights of this chapter is that improving the tail does not follow the same intuitions that we use in improving the mean. While improving mean response time is often a matter of helping small jobs jump ahead of large ones, when it comes to the tail, this has to be done in a very measured way. Too much help to the small jobs causes the tail to get a lot worse. Nudge finds the exactly appropriate way to do this.

One direction for future work is further exploring the stochastic improvement frontier. Can we stochastically improve upon other commonly used scheduling policies? Can we improve upon Nudge itself, such as with a more complicated variant of Nudge (see Section 7.12)? One policy which cannot be stochastically improved upon is SRPT, due to its optimal mean response time. Can we prove that other policies are unimprovable?

Another direction is simplifying the definition of Nudge. Our empirical results in Section 7.10 indicate that in practice, Nudge can always stochastically improve upon FCFS with only two classes of jobs: small and large. Thus Nudge appears to only need a *single* cutoff. It would be of practical importance to figure out how to extend the theorems in this chapter to hold for this simplified definition of Nudge.

7.14 General Conclusion

7.14.1 Summary

We start by summarizing the results proven in this chapter, as well as the key techniques behind these results.

Results: Nudge achieves better asymptotic tail response time than FCFS We prove that our novel Nudge policy, which we define in Section 7.4.5, achieves a smaller asymptotic tail of response time than FCFS (See Theorem 7.5.3 and Corollary 7.5.1). In particular, in the limit as the response time threshold t goes to infinity, the ratio of tail probabilities $\mathbb{P}\left\{T^{\mathsf{Nudge}} > t\right\} / \mathbb{P}\left\{T^{\mathsf{FCFS}} > t\right\}$ is strictly below 1. This result overturns the prior conjecture of FCFS's strong tail optimality [232]. This result holds for a broad class of job size distributions known as "light-tailed" job size distributions (See Section 7.4.3), which is the class of job size distributions for which FCFS achieves good tail performance [27].

Results: Nudge stochastically dominates FCFS Moreover, we prove that Nudge *stochastically dominates* FCFS with respect to the tail of response time (See Theorems 7.5.1 and 7.5.2). By "stochastically dominates", we mean that for every response time threshold t, Nudge's tail probability $\mathbb{P}\left\{T^{\mathsf{Nudge}} > t\right\}$ is smaller than FCFS's tail probability $\mathbb{P}\left\{T^{\mathsf{FCFS}} > t\right\}$. As a result, Nudge improves upon FCFS for *every* common tail performance metric, including all percentiles of response time such as T^{99}, and all moments of response time such as $\mathbb{E}[T^2]$.

Simulation result: Nudge achieves significant improvement While our results only prove that some improvement over FCFS is achieved, we show via simulation in Fig. 7.5 that Nudge achieves a significant margin of improvement over FCFS. For instance, in Fig. 7.5(a), we see that under a variety of relatively high-variance job size distributions, Nudge achieves 10-15% better tail performance than FCFS across all response time thresholds.

Key technique: Analyze Nudge relative to FCFS Our analysis relies on the wealth of results about the distribution of response time under FCFS. Because Nudge is defined to be a

slight modification of FCFS, we are able to quantify the scenarios where Nudge outperforms FCFS and vice versa. Every time Nudge interchanges the service order of a pair of jobs relative to FCFS, that decision is beneficial relative to some response time thresholds t, and detrimental relative to other thresholds. We use prior results on FCFS's response time distribution to prove that in total, the beneficial interchanges outweigh the detrimental interchanges. In particular, for every threshold t, beneficial interchanges occur at a higher rate than detrimental interchanges. This relative analysis is the key to proving Lemma 7.6.2, which is the key step towards proving our main results.

7.14.2 Subsequent Results: Nudge-K

The Nudge policy and the results discussed in Section 7.14.1 have been generalized to an entire family of scheduling policies in subsequent work. Van Houdt [222] defines the Nudge-K family of scheduling policies. While Nudge only moves a job by at most one position relative to the arrival ordering, the Nudge-K policy allows a small job to move past up to K large jobs. Van Houdt characterizes the asymptotic tail of response time of Nudge-K, and theoretically characterizes the value of K for which Nudge-K achieves the best asymptotic tail of response time. Note that the Nudge policy considered in this paper is equivalent to that paper's Nudge-1. Van Houdt also empirically finds workloads for which Nudge-2 stochastically improves upon Nudge-1, answering one of the open problems we pose in Section 7.12. Finally, Van Houdt finds complex and interesting behavior when the "small" and "large" jobs, rather than being strictly ordered by size, can have overlapping size distributions.

7.14.3 Future Direction: Intermediate Tail

This chapter invents the Nudge policy, and proves that it stochastically improves upon FCFS. FCFS is primarily of interest for its excellent tail performance for very large tail thresholds t. Nudge is therefore a better choice in this regime. At the opposite extreme, when scheduling for mean response time, SRPT is known to achieve optimal mean response time.

The *intermediate* tail performance regime, neither focusing on the asymptotic tail nor the mean, is underexplored. One way of quantifying this intermediate tail is to look at the *moments* of response time, namely $\mathbb{E}[T^\alpha]$ for various constants α. If $\alpha = 1$, we have the mean. In the $\alpha \to \infty$ limit, the asymptotic tail dominates. A natural intermediate choice is $\alpha = 2$, giving rise to $\mathbb{E}[T^2]$, the second moment of response time.

As a measure of the intermediate tail, $\mathbb{E}[T^2]$ has the advantage that it has already been analyzed for many scheduling policies. For many scheduling policies, the Laplace-Stieltjes transform of mean response time $\widetilde{T}(s)$ is known [104, 201], and the second moment $\mathbb{E}[T^2]$ can easily be extracted from $\widetilde{T}(s)$. However, the policies that would achieve the best second-moment performance have not been analyzed.

For instance, as we discuss in Section 8.3.5, a natural scheduling policy to try to optimize $\mathbb{E}[T^2]$ is the t/s policy, which prioritizes jobs by their ratio of time in system to job size. The t/s policy has not been analyzed, though the Accumulating Priority Queue framework [21, 59, 159, 211] may be a useful tool for analyzing it. Analyzing the t/s policy's second moment $\mathbb{E}[T^2]$ would be a major step forward in our understanding of the intermediate tail performance regime.

7.14.4 Future Direction: Multiserver Nudge

This chapter studies Nudge in the single-server setting. However, multiserver settings, as studied in the rest of this thesis, better model the behavior of large-scale computing systems. Therefore, a natural open question is:

> In a multiserver setting, does Nudge stochastically improve upon FCFS? Does Nudge achieve a better asymptotic tail of response time than FCFS?

To prove that Nudge does achieve such an improvement, the work-conserving finite-skip analysis from Chapter 4 and the general finite-skip analysis from Chapter 6 may be useful, as Nudge is a finite-skip scheduling policy. While the results in those chapters focus on mean response time, the techniques could likely be generalized to analyze the tail of response time.

7.14.5 Potential Impact

We now explore potential directions in which the Nudge scheduling policy could be applied.

Adopting Nudge into Modern Computing Systems

Our analysis of the Nudge scheduling policy shows that Nudge can significantly improve tail performance compared to FCFS scheduling, which is the default scheduling policy in many computing systems. Nudge stochastically improves upon FCFS scheduling, making it a better choice for all tail performance metrics.

Moreover, Nudge is an extremely simple policy to implement: It does not perform preemption, and a job's position in the service ordering is typically determined long before the job reaches service.

However, the road from theoretical results to adoption requires overcoming several hurdles. These include:

Size estimates. Nudge uses exact size information, but in real systems, only size estimates may be available. Fortunately, Nudge only uses size information to classify jobs as either "small" or "large", for the purpose of moving a small job past a large job. If size estimates are instead used for this classification, Nudge will still improve upon FCFS, as long as those estimates are usually accurate. If estimates are relatively low quality, it may be helpful to classify some jobs as "uncertain", and avoid interchanging those jobs. If some jobs can be reliably estimated to be smaller than other jobs, then Nudge can improve upon FCFS.

General arrival processes. In this chapter, we assumed a Poisson arrival process, where jobs arrive at a static rate. In real systems, arrivals may be burstier, temporarily arriving faster and slower than the long-term rate. In these bursty scenarios, it is important not to interchange the order of a pair of small and large jobs where the small job arrives much later than the large job. Such interchanges can have an outsized negative impact on tail performance. In the Poisson arrival case, this scenario is rare enough that it can safely be ignored. In the bursty arrival setting, it would be useful to place an explicit limit: An

interchange should only be performed if the difference in arrival times is not much larger than the mean interarrival time.

Scheduling People Using Nudge

Given how useful Nudge can be when scheduling jobs in a computing system, it is natural to consider it for scheduling people. For instance, one could use Nudge to schedule calls at a call center. One could estimate jobs as either shorter or longer based on the call description, and then interchange the order of one short call and one long call. By doing so, one would achieve stochastic improvement over the default FCFS ordering.

Moreover, Nudge has advantageous properties specific to the setting of scheduling people. People care deeply about fairness in scheduling. In this regard, Nudge is an appealing choice. People generally agree that FCFS is a fair way to schedule. Under Nudge, everyone achieves a response time similar to their response time under FCFS: A long job is delayed by at most one small job's duration. As a result, no one is inconvenienced too much, while improving the overall tail performance of the system.

Nudge also produces advantageous incentives when compared to more dramatic scheduling policies such as Shortest Remaining Processing Time (SRPT). Under SRPT, the job with the shortest duration receives immediate priority over all other jobs. As a result, people may be sorely tempted to misreport the features of their job, to try to receive service much faster. Under Nudge, while smaller jobs still receive service somewhat sooner, the possible effect of such misreporting is much smaller, reducing the incentive to lie.

Part V

Conclusion

Chapter 8

Conclusion

8.1 Summary

In this thesis, we study three themes within the broad heading of optimal scheduling in multiserver queues. We now summarize the results from each theme and the key techniques we use to prove those results.

Theme 1: One-server-per-job multiserver scheduling In Chapters 2 and 3, we study scheduling in standard multiserver models where each job occupies a single server. In both cases, we study how to optimally make use of *size information*, where a job's size is the inherent amount of work in the job. We summarize our results and key techniques in Section 8.1.1.

Theme 2: Multiserver-job scheduling In Chapters 4 to 6, we study scheduling in *multiserver-job* (MSJ) models, where different jobs require different numbers of servers. We invent novel policies, analyze the performance of novel and existing policies, and prove optimality results in the MSJ setting. We summarize our results and key techniques in Section 8.1.2.

Theme 3: Tail scheduling In Chapter 7, we study scheduling for the purpose of improving the *tail* of response time. In contrast, the other two themes focus on *mean* response time. Our work in this theme focuses on single-server models, but we hope that the resulting policy and analysis will be useful in future multiserver analysis. We summarize our results and key techniques in Section 8.1.3.

8.1.1 Theme 1: One-server-per-job Multiserver Scheduling

Multiserver systems are the default in large-scale computing. A computing cluster will have hundreds or thousands of machines, each with dozens of cores. Thousands of jobs might be in progress at the same time. Scheduling decisions, deciding in what order to serve the jobs, have the potential to yield major performance improvements with no additional resources. However, scheduling theory has overwhelmingly focused on singe-server models. In this thesis, we prove the first results on optimal multiserver scheduling.

We study two models of multiserver scheduling: a *central queue* model, in Chapter 2, and a *dispatching* model, in Chapter 3. We depict these two models in Fig. 8.1.

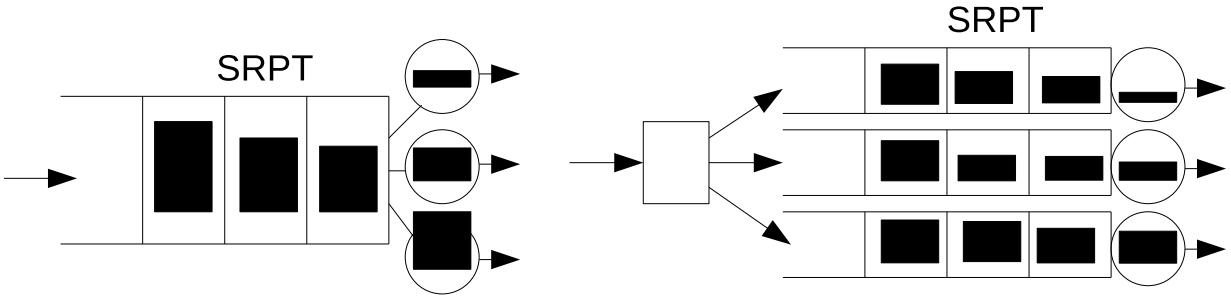


Figure 8.1: The central-queue multiserver model (Chapter 2) and the dispatching multiserver model (Chapter 3). In both cases, the scheduling policy is Shortest-Remaining-Processing-Time (SRPT), which serves the job(s) of least remaining size. In the dispatching model, a dispatching policy must also be chosen. The dispatching policy determines which jobs to send to which servers.

In Chapter 2, we primarily study an existing scheduling policy, the Shortest-Remaining-Processing-Time (SRPT) policy, which serves the k jobs of least remaining *size*. Size is the inherent work of a job, which in this setting is proportional to the amount of time in service the job requires. We show the central-queue SRPT policy in Fig. 8.1.

In the dispatching setting we consider in Chapter 3, there are two policy decisions that must be made: The *dispatching* policy, which determines which jobs to send to which servers, and the scheduling policy, which determines the order in which to serve the jobs at a given server. In Chapter 3, we study the SRPT scheduling policy, which is the optimal scheduling policy to combine with any dispatching policy. We devise the novel *guardrails* class of dispatching policies for use in combination with SRPT scheduling (See Section 3.3). The key property of guardrails dispatching policies is to ensure that each job has a mixture of small, medium, and large jobs. We depict dispatching to servers which use SRPT scheduling in Fig. 8.1.

Results: Optimality In Chapter 2, we prove that the multiserver SRPT scheduling policy achieves optimal mean response time in the *heavy-traffic* limit, the limit in which the arrival rate approaches the capacity of the system (See Corollary 2.7.1). In Chapter 3, we prove that any dispatching policy in our novel guardrails class of policies, when combined with the SRPT scheduling policy, achieves optimal mean response time in the heavy-traffic limit (See Corollary 3.4.1). These proofs are the first optimality results in their respective multiserver settings. In Chapter 2, we prove that several additional scheduling policies also achieve optimal mean response time in the heavy-traffic limit (See Section 2.8).

Results: Bounds on mean response time In Chapter 2, we prove an upper bound on mean response time under multiserver SRPT (See Theorem 2.6.2). This upper bound proves that multiserver SRPT's mean response time is not much larger than that of *resource-pooled SRPT*. In a resource-pooled system, all of the k original servers are combined into one giant server which runs k times faster. Using prior single-server analysis, we know the mean response time in the resource-pooled SRPT system [196]. Our bound relative to the resource-pooled SRPT system becomes tight in the heavy-traffic limit (See Theorem 2.7.1). The resource-pooled SRPT system is a lower bound on the achievable mean response time in the multiserver system, so this tightness implies our optimality result. In Chapter 3, we likewise prove an upper bound on mean

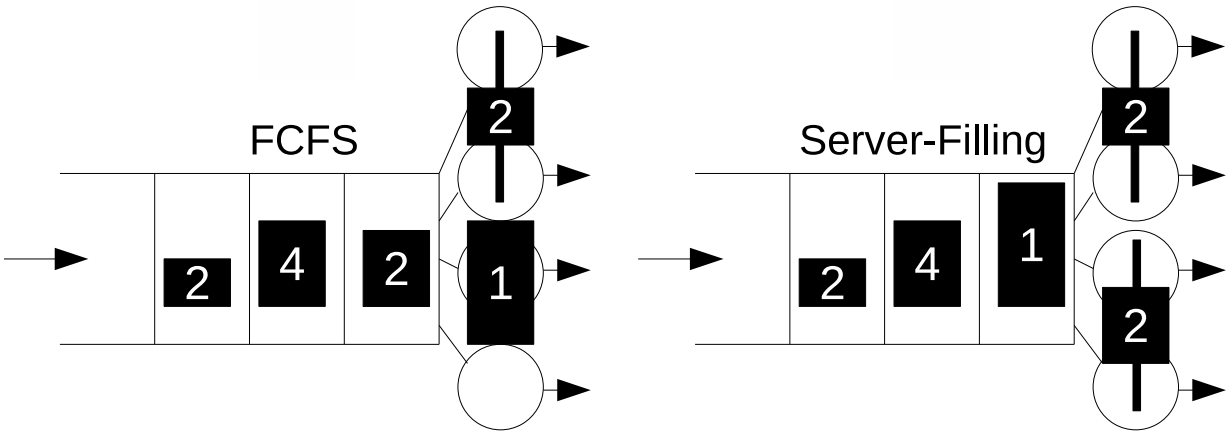


Figure 8.2: The multiserver-job (MSJ) model, under two scheduling policies: MSJ FCFS (Chapter 6) and ServerFilling (Chapter 4). In the MSJ model, a job has two characteristics: A *server need*, the number of servers it requires in order to run, and a *service duration*, the amount of time in service it requires in order to complete.

response time under the combination of a guardrails dispatching policy and SRPT scheduling (See Theorem 3.4.1). This bound likewise becomes tight in the heavy traffic limit (See Theorem 3.4.2).

Simulation results: Low mean response time at moderate load In Chapter 2, we conduct simulations which show that multiserver SRPT's mean response time converges to that of resource-pooled SRPT even at moderate arrival rates, not just in the heavy traffic limit (See Fig. 2.6). In Chapter 3, we conduct simulations which show that the combination of guardrails dispatching and SRPT scheduling achieves either the best or close-to-best mean response time at moderate loads, under a variety of workloads (See Section 3.6).

Key technique: Coupling with resource-pooled SRPT Our key idea in both Chapters 2 and 3 is to compare each of our multiserver systems to a resource-pooled SRPT system by coupling their arrival processes. Specifically, we feed the multiserver system and the resource-pooled system the same arrivals, and prove that both systems have nearly identical amounts of *relevant work* present at any given time. "Relevant work" is the amount of work present in the system that would be prioritized ahead of a job with a given size. We prove this similarity in relevant work between the coupled multiserver and resource-pooled systems in Lemmas 2.6.2 and 3.5.2. We then build off the similarity in relevant work to prove that the systems have similar mean response time, which is our main result.

8.1.2 Theme 2: Multiserver-job Scheduling

In large-scale computing systems, it is not only the case that there are many servers in the system, and hence many jobs in service at a time. In addition, jobs often require dramatically different amounts of resources, ranging from a small fraction of a machine to most of an entire cluster. To capture the behavior of such systems, we study the multiserver-job model.

In the multiserver job (MSJ) model, each job has two requirements: Its *server need*, the

number of servers it requires in order to run, and its *service duration*. We depict this model in Fig. 8.2.

Within the MSJ model, we consider several different settings which differ on several axes:

Preemption: When a job in service is preempted, it is paused and returned to the queue, and it can be resumed later. One could either allow or disallow preemption.

Server need distribution: One could consider a general server need distribution, where jobs of any server need are possible, or a restricted server need distribution, where only certain server needs are possible. One might restrict the set of possible server needs to ensure better performance.

Duration information: One could consider scheduling policies which know the duration of the jobs in advance, or which do not have such knowledge.

We study three distinct settings:

Chapter 4: We consider preemption to be available, a restricted server need distribution, and *unknown* durations.

Chapter 5: We consider preemption to be available, a restricted server need distribution, and *known* durations.

Chapter 6: We consider preemption to be *unavailable*, a *general* server need distribution, and unknown durations.

These settings present a tradeoff between flexibility and performance. Chapter 6 is the most flexible, followed by Chapter 4 and then Chapter 5, while we prove that the best performance is achieved in Chapter 5, then Chapter 4, then Chapter 6.

Results: Mean response time analysis and optimality In each chapter, we characterize the mean response time of either novel or existing scheduling policies.

In Chapter 4, we start by examining prior MSJ scheduling policies. Unfortunately, all prior MSJ scheduling policies either are very complicated, or have poor mean response time. Moreover, no closed-form mean response time analysis is known for any prior scheduling policies.

We invent the ServerFilling scheduling policy (See Section 4.4.4), which is a relatively simple scheduling policy with good mean response time, and for which we can analyze mean response time.

ServerFilling is designed for the MSJ setting where all server needs are powers of 2. Furthermore, the total number of servers, k, is also a power of 2. To analyze ServerFilling, we prove that its mean response time is nearly identical to the mean response time of the *resource-pooled* First-Come First-Served (FCFS) scheduling policy (See Theorem 4.6.2). By the "resource-pooled" system, we refer to a system in which all k servers are combined into one giant server which runs any job at a speed k times faster than each individual server. Because single-server mean response time is well understood, this allows us to analyze ServerFilling's mean response time. This approximation becomes tight in the *heavy-traffic* limit, the limit in which the arrival rate approaches the capacity of the system (See Theorem 4.6.1). Our results also generalize to a large class of scheduling policies known as work-conserving finite-skip (WCFS) policies (See Section 4.3). This class includes the DivisorFilling policy, which handles the MSJ settings where all server needs are divisors of k (See Section 4.7).

In Chapter 5, we invent the ServerFilling-SRPT scheduling policy (See Section 5.4.2). We prove that the ServerFilling-SRPT scheduling policy achieves optimal mean response time for the MSJ setting, in the heavy-traffic limit (See Theorem 5.5.2). Specifically, we prove that ServerFilling-SRPT's mean response time is nearly identical to that of resource-pooled SRPT (See Theorem 5.5.1). The mean response times of ServerFilling-SRPT and resource-pooled SRPT converge in the limit as the arrival rate approaches the capacity of the system. Our results also generalize to DivisorFilling-SRPT (See Section 5.7), as well as the ServerFilling-Gittins policy for the scenario where job sizes are unknown (See Section 5.6). The Gittins index policy is the analogue of SRPT for the setting of unknown job sizes, as it achieves optimal single-server mean response time [72, 73, 199].

In Chapter 6, we analyze the MSJ FCFS scheduling policy's mean response time (See Theorem 6.5.2). Note that the FCFS policy leaves servers empty when jobs cannot fit into the available servers, as shown in Fig. 8.2, making it considerably more difficult to analyze. Nonetheless, we give a mathematical formula closely approximating MSJ FCFS's mean response time. This approximation becomes tight in the heavy-traffic limit. Our result relates the behavior of the MSJ system to that of the *saturated system*, a closed system where completions trigger new arrivals and the total number of jobs in the system is always constant. Our results also generalize to handle a large class of scheduling policies called *finite skip* scheduling policies (See Section 6.10).

Simulation results: Accurate analysis and low mean response time at moderate load In Chapter 4, we conduct simulations which show that our approximation for ServerFilling's mean response time is highly accurate, even more accurate than our bounds imply (See Fig. 4.7). In Chapter 5, we conduct simulations which show that ServerFilling-SRPT achieves excellent mean response time even at moderate load, always remaining close to the resource-pooled SRPT lower bound (See Fig. 5.5). In Chapter 6, we conduct simulations which show that our approximation for MSJ FCFS's mean response time is highly accurate, across a range of arrival rates (See Fig. 6.4.

Key technique: W^2 method One of our key ideas in Chapters 4 to 6 is the W^2 *method*. This technique takes its simplest form in Chapter 4. First, we define a job's size, its inherent work, to be the product of the fraction of service capacity it requires and its service duration. Note that the fraction of service capacity a job requires is its server need divided by k, the total number of servers. We then define W to be the total amount of work in the system, the total remaining size of all jobs in the system. With this definition of work, whenever all k servers are occupied, work is completed at rate 1. The ServerFilling policy keeps all k servers occupied whenever at least k jobs are present (see Lemma 4.4.1), ensuring that it completes work at rate 1.

We then examine the random variable W^2, and examine its rate of change, both due to arrivals and work completion. We know that the expected rates of change due to these two causes must be equal in equilibrium. In Lemma 4.6.2, we use this equality of rates to derive bounds on $E[W]$, the expected work in the system. These bounds consist of a term matching the work in a resource-pooled M/G/1, and a *waste* term, which is only nonzero when *fewer* than k servers are occupied. Using the properties of ServerFilling, we are able to bound the waste term. This expression is the key to our analysis of ServerFilling's mean work, and by extension ServerFilling's mean response time.

Extension: W_x^2 method In Chapter 5, rather than quantifying mean *total* work $E[W]$, we need to quantify mean *relevant* work $E[W_x]$. Relevant work is work consisting of jobs with

remaining size smaller than some threshold x. Analyzing relevant work is key to understanding SRPT and SRPT-like scheduling policies, such as ServerFilling-SRPT. We now examine the rate of change of W_x^2 due to arrival, work completion, and *recyclings*, jobs diminishing in remaining size down to remaining size x. As we show in Lemma 5.5.2, the equality of rates allows to give an expression for mean relevant work $E[W_x]$. This technique is also known as the "work decomposition law", and was introduced in our prior work on multiserver Gittins [202]. This expression is the starting point for our analysis of ServerFilling-SRPT's mean response time.

Extension: Relative completions In Chapter 6, the MSJ FCFS scheduling policy does not generally keep all of the servers occupied. As a result, work W does not typically complete at rate 1. To overcome this, we jettison the use of work W altogether. Instead, we take the number of jobs in the queue Q, and subtract off the *relative completions* $\Delta(Y)$ (See Section 6.4.8). Y represents the state of the k oldest jobs in the system, which includes all of the jobs which could be in service. We design the relative completions Δ to ensure that the difference $Q - \Delta(Y)$ always decreases due to completion at a *constant expected rate*. In this way, $Q - \Delta(Y)$ plays the same role in our analysis that W plays in Chapters 4 and 5. In particular, we examine the rate of change of the function $(Q - \Delta(Y))^2$ as a key step in our analysis (See Definitions 6.6.1 and 6.7.1).

8.1.3 Theme 3: Tail scheduling

In computing systems, system operators typically focus on performance metrics which highlight jobs with longer response times. Such metrics are known as *tail* performance metrics. These long-delayed jobs represent the most annoyed users and the most severe problems, and making them the system operator's most pressing concern. With that in mind, our scheduling policies should seek to improve the *tail* of response time.

We study scheduling for the goal of improving the tail of response time, in contrast to the focus on *mean* response time in the rest of this system. We study the tail of response time in a single-server setting, as even in this simpler setting, the effect of scheduling on tail performance already presents several important open problems.

In Chapter 7, we devise the novel *Nudge* scheduling policy (See Section 7.4.5). We compare Nudge against the FCFS scheduling policy, which was previously the best known scheduling policy for optimizing the *asymptotic tail* of response time [27]. The asymptotic tail refers to the asymptotic behavior of the *tail probability* $P(T > t)$.

Results: Better asymptotic tail than FCFS We prove that Nudge achieves a better *asymptotic* tail of response time than FCFS (See Corollary 7.5.1). Previously, FCFS was the best known scheduling policy for the asymptotic tail of response time, and was conjectured to be the optimal scheduling policy for the asymptotic tail [232]. We overturn that conjecture with our result on Nudge.

Results: Stochastic improvement on FCFS We prove that Nudge *stochastically improves* upon FCFS (See Theorem 7.5.2). Specifically, we prove that for all thresholds t, Nudge achieves a better tail probability $P(T > t)$ than FCFS. This result implies that Nudge improves upon FCFS for essentially all tail metrics, including all response time percentiles such as T^{99} and all moments of response time such as $E[T^2]$.

Simulation result: Significant improvement We conduct simulations which show that Nudge's improvement over FCFS is sizable in practice, achieving a 10-15% improvement in tail probability across a wide range of thresholds t and a wide variety of job size distributions (See Fig. 7.5(a)).

Key technique: Relative analysis Nudge is defined to be a slight modification of FCFS – reordering jobs by at most one position relative to the arrival ordering. As a result, we are able to characterize the scenarios where Nudge outperforms FCFS, and FCFS outperforms Nudge. Every reordering of jobs is beneficial relative to some response time thresholds t, and detrimental relative to other thresholds. In Lemma 7.6.2, we use prior results on FCFS's response time distribution to bound the relative probabilities of benefit and detriment relative to a given threshold t. We show that for every threshold, beneficial reorderings occur at a faster rate than than detrimental reorderings.

8.2 Potential impact

We now explore potential directions in which our scheduling policies could be applied. We discuss stochastic multiserver scheduling (Chapters 2 and 3) in Section 8.2.1, multiserver-job scheduling (Chapters 4 to 6) in Section 8.2.2, and tail scheduling (Chapter 7) in Section 8.2.3). For each theme, we discuss potential impact in two areas: scheduling for large-scale computing, and scheduling involving people.

8.2.1 Stochastic Multiserver Scheduling

We now explore potential directions in which the central-queue SRPT (Chapter 2) scheduling policy could be applied to real-world environments, as well as the guardrails class of dispatching policies for use with SRPT scheduling (Chapter 3).

Adopting SRPT and Guardrails into Multiserver Computing Systems

Our analysis shows that SRPT scheduling can dramatically lower response times compared to other scheduling policies, and that when SRPT scheduling is used, our guardrails class of dispatching policies can also dramatically lower response times. Both central-queue SRPT and the combination of guardrails dispatching and SRPT scheduling are optimal for mean response time under sufficiently heavy load. Our hope is that our results will lead computing system operators to adopt these policies in their systems.

However, the road from theoretical results to adoption requires overcoming several hurdles, including:

Unknown or estimated sizes: Often only approximate size information is known, rather than the exact size information utilized by SRPT. If the job size distribution is known or the estimate quality distribution is known, the *Gittins index* scheduling policy can be applied. The Gittins policy has long been known to achieve optimal single-server mean response time [72, 73, 199], making it the appropriate analogue of SRPT scheduling in this setting. In the central-queue setting, we have shown in subsequent work that multiserver Gittins

scheduling achieves asymptotically optimal mean response time [202]. In the dispatching setting, the problem of dispatching in a system where the servers use Gittins scheduling is an open problem (See Section 8.3.3). If estimates of relatively good quality are available, one can simply use the guardrails approach, using a job's size estimate to select a priority class.

In subsequent work, we invent the SRPT-B scheduling policy for which situation where estimates are available but their quality is unknown [205]. We prove that in the single-server setting, SRPT-B achieves *consistent* and *gracefully degrading* performance. Specifically, SRPT-B matches the optimal mean response time of SRPT when estimates are highly accurate, and degrades smoothly as estimates become less accurate. SRPT-B would likely perform well in the multiserver setting, but it has not been analyzed.

Preemption overhead: The SRPT scheduling policy often preempts jobs, putting partially completed jobs back into the queue. In real systems, frequent preemption may be undesirable, or there may be overhead associated with having many incomplete jobs in progress at once. To adapt to these real-world concerns, one should only preempt jobs at the highest-impact times. Preemption is most important to ensure that small jobs that can finish quickly are not stuck behind large jobs that must finish slowly. One should therefore only preempt jobs that are larger than a typical job in the system, and only preempt infrequently: If jobs typically take about 1 second to complete, we might preempt a large job after it has been in service for 1 second. Finally, we should only perform preemption when the queue is long, meaning that many short jobs that are being held up by this one long job.

Multilevel dispatch: In this theme, we focused on the cases of central-queue multiserver scheduling, and multiserver dispatch to individual servers. However, in many systems, there are more levels of queues, and more dispatching decisions. For instance, a job might be dispatched to one of many computing clusters, then to one of many machines within that cluster, and then wait in a central queue within that machine for one of many cores in the machine. In any such multilevel hierarchy, the combination of SRPT scheduling and guardrails dispatching will achieve low mean response times. Our optimality results will likely compose, proving similar optimality results for such a hierarchical setting.

Differing importance: In the real world, some jobs may be far more latency-sensitive than others. This is naturally modeled by introducing a holding cost per second waited for each job. The natural generalization of SRPT to this setting is to prioritize jobs according to the ratio of holding cost to remaining size. This policy is known as the "$c\mu$-rule". The results of this theme generalize to prove optimality in that setting as well. Specifically, if the ratio of largest to smallest holding cost is bounded, and does not change over time, the techniques of this theme suffice. Gittins-based techniques may be helpful to handle the setting of general holding costs [202].

Adopting SRPT and Guardrails for Scheduling People at Multiple Servers

Given that SRPT is so good at reducing mean response time for computing systems, it is natural to also try to apply it to scheduling people. For instance, if two people show up to a post office, and one needs to mail a letter while the other needs to fill out some complicated shipping paperwork, we should first serve the "short job", the person who won't take as long.

However, challenges arrive when serving people. Some of these challenges, such as differing time-sensitivity and overheads associated with preemption, have already been discussed early in this chapter in the context of scheduling for computing.

Challenge: Fairness Some challenges are specific to the context of serving people. In particular, people have strong opinions about fairness: If everyone is put in the same line, and certain people are pulled out for favored treatment, people will get upset. However, if one makes the scheduling structure transparent to the people involved, it will help quell that anger. For instance, when checking people in to be seen at a bank office, one could have several sign-in lists, associated with different durations of interactions and/or different time-sensitivities. Then people know that the order they're being seen isn't due to favoritism or other inappropriate reasons. People can also watch themselves approaching the front of their specific list, which makes people more confident that they will be helped soon.

8.2.2 Multiserver-job Scheduling

We now explore potential directions in which the ServerFilling (Chapter 4) amd ServerFilling-SRPT (Chapter 5) scheduling policies could be applied in real-world multiserver-job (MSJ) scheduling environments, as well as our analysis of the MSJ FCFS scheduling policy (Chapter 6).

From Theory to Practice in Multiserver Computing Systems:

- **Adoption of ServerFilling and ServerFilling SRPT scheduling policies**
- **Utilizing the mean response time prediction from our MSJ FCFS analysis**

Our hope is that our results will lead computing system operators to adopt our scheduling policies in their systems, and to adopt our predictions of response time into their decision-making process. However, the road from theoretical results to adoption requires overcoming several hurdles, including:

Selecting server needs: Our results show that the best MSJ performance can be achieved when the job's server needs can consistently be packed onto the k servers, filling all k. Perfect packing can be guaranteed when all server needs are powers of 2, and k is a power of 2. This is key to our analysis of ServerFilling and ServerFilling-SRPT. However, in practice, one might instead choose to be more flexible with server needs, and use a heuristic packing strategy. If one used a heuristic packing strategy, it would be advisable to avoid having jobs with server needs that are just *above* a divisor of k. For instance if $k = 1000$ servers, one should avoid having jobs with server need 501 or 334, as a couple of such jobs could lead to lots of servers being left idle. If server needs are chosen well, the heuristic packing policy would likely only leave a small number of servers idle on average, which would only degrade performance slightly. The magnitude of the mean response time impact can be quantified using the approach from Chapter 6.

Multidimensional resource requirements: In real systems, jobs often require a variety of resources, such as CPU cores, GPUs, memory, network bandwidth, disk IO, and more.

These can be modeled as multidimensional resource requirements, in contrast to the single-dimensional server need model considered in this chapter. In a general multidimensional setting, using all of the resources at all times is typically not attainable. As a result, the results of Chapters 4 and 5, which focus on this "full utilization" regime, will not be applicable. However, if a single bottleneck resource exists, then we can focus on fully utilizing that one resource, and the results of Chapters 4 and 5 may be applicable.

By contrast, the techniques of Chapter 6 can directly handle the multidimensional setting. The multidimensional setting falls within the "finite skip" class of policies discussed in Section 6.10.

Jobs with changing server need: In real systems, a job which requires concurrent service on many servers often consists of many separate tasks sent to individual servers, which then run mostly separately. Typically, some tasks will finish before others, until a few straggling tasks are still running.

We can model this as a job whose server need diminishes as it receives service. Our results in this theme can handle this variant of the MSJ model. In particular, if the job's server need is always a divisor of k, the DivisorFilling and DivisorFilling-SRPT policies from Chapters 4 and 5 can be applied. For instance, in a $k = 12$ server system, a job could have server need 4, then 3, then 2, then 1.

More generally, for arbitrary server needs, the MSJ FCFS analysis in Chapter 6 also applies to this model of changing server needs.

ServerFilling, ServerFilling-SRPT, and MSJ FCFS for Scheduling People in Groups

Given our results for ServerFilling, ServerFilling-SRPT, and MSJ FCFS, it is natural to apply our policies and our analysis for scheduling jobs that involve people. For example, in a contracting-based business, different contracts might require different sizes of teams, and a ServerFilling or ServerFilling-SRPT approach could be employed to schedule those contracts. A landscaping company might have many lawns to tend to, with different sized lawns requiring different numbers of people.

Challenge: Preemption However, challenges arise when scheduling people. Preemption must be handled carefully, and performed rarely. To reduce the preemption rate of ServerFilling, it makes sense to expand the set of jobs that are eligible for service. If people complete a job, the first choice should be to assign those people to another job that requires the same number of people. Only if no such jobs are available, or if such an assignment would prioritize a newly arrived job over a job that has been waiting for a long time, should preemption be performed instead.

Challenge: Differing importance When scheduling jobs involving people, it is typically for different jobs to have different time sensitivities. A contract might be a rush-order, for instance. To incorporate this into a performance objective, it is common to introduce a holding cost, representing the cost per day until completion. To incorporate this information into a scheduling decision, one should make use of the "$c\mu$-rule", which prioritizes jobs according to the ratio of the job's holding cost to its remaining size. In particular, one should use the ServerFilling-$c\mu$ policy, analogous to ServerFilling-SRPT. Our optimality results on ServerFilling-Gittins actually cover ServerFilling-$c\mu$ policy, proving that it minimizes mean cost-weighted response time

in the limit as arrival rate approaches capacity. If size information is unknown, one could just use ServerFilling-c, just prioritizing by holding cost.

8.2.3 Tail scheduling

We now explore potential directions in which the Nudge scheduling policy (Chapter 7) could be applied in real-world scheduling environments.

Adopting Nudge into Modern Computing Systems

Our analysis of Nudge shows that it can significantly improve tail performance compared to FCFS scheduling, which is both the default scheduling policy in many fields, as well as the previous best known scheduling policies. Nudge stochastically improves upon FCFS scheduling, making it a better choice for all tail performance metrics. Moreover, Nudge is an extremely simple policy to implement: It merely reorders jobs by at most one position relative to FCFS scheduling.

However, our analysis of Nudge focused only on the single-server scheduling setting. As we have discussed throughout this thesis, the multiserver scheduling model is more realistic. The insights that inspired Nudge also apply in the multiserver setting, so its strong tail performance is likely to transfer to that setting. Future work will be needed to verify and prove that Nudge works well in the multiserver setting, however.

Beyond moving to multiserver settings, the road to practical adoption also requires overcoming several hurdles, including:

Size estimates: Nudge uses exact size information, but in real systems, only size estimates may be available. Fortunately, Nudge only uses size information to classify jobs as either large or small, for the purpose of moving a small job ahead of a large job. If size estimates are instead used for this classification, Nudge will still improve upon FCFS as long as the estimates as sufficiently accurate.

Unknown sizes: In other settings, no size information may be known at all. We can design a no-information variant of Nudge for this setting. Key to this policy is the insight that in many practical scheduling settings, the job size distribution is high-variance. In high-variance settings, a job that has received a significant amount of service typically has a larger remaining size than a fresh job. We can think of this not-yet-complete job as a "large" job for the purpose of Nudge, and think of the next fresh job in the queue as a "small" job. We would preempt the incomplete large job, and run the fresh, hopefully smaller job for some period of time. When the fresh job completes, or runs for long enough that it doesn't seem so small any more, we would then return to the paused large job.

Scheduling People Using Nudge

Given how useful Nudge can be when scheduling jobs in a computing system, it is natural to consider it for scheduling people. People often do something a bit like Nudge in an ad-hoc fashion. Suppose two people are waiting in a checkout line, and the one farther forward in line has lots to buy and the one behind has very little. In this case, the person in front will often let

the other person go by, as a matter of courtesy. But if someone's already been passed in this way, they might refuse to do it again. Nudge can be seen a formalization of this social convention, and an explanation of why it's a helpful idea.

Nudge specifically achieves a good sense of individual fairness, while also improving overall performance. At an individual level, FCFS feels fair: No matter who you are or what the characteristics of your job are, you'll wait for the people who arrived before you, and people who arrived after you will wait for you. Nudge has a similar feeling of individual fairness: Only at most one person who arrived after you will run before you, and only if that person has a small job. This is in contrast to a scheduling policy such as SRPT, where some jobs wait for almost no one, and others wait for almost everyone.

8.3 Open problems

We end this thesis by presenting five open problems in optimal multiserver scheduling, each of which build upon the work in this thesis. The open problems are:

Section 8.3.1: Finding a scheduling policy for the central-queue multiserver model (M/G/k) with better mean response time than multiserver SRPT.

Section 8.3.2: Proving tight lower bounds on M/G/k scheduling.

Section 8.3.3: Optimal dispatching under unknown sizes or estimate sizes, by dispatching to queues using Gittins scheduling.

Section 8.3.4: Optimal scheduling in the general multiserver-job model.

Section 8.3.5: Optimal scheduling for the *intermediate* tail of response time, focusing on metrics between the extremes of mean response time and the asymptotic tail of response time.

8.3.1 Improving Upon SRPT-k at Moderate Load

In the context of multiserver scheduling (in the M/G/k), SRPT has multiple optimality properties. As we showed in Chapter 2, under heavy traffic, namely as load $\rho \to 1$, multiserver SRPT (SRPT-k) achieves asymptotically optimal mean response time. At the opposite extreme, if there were no arrivals at all, SRPT-k has long been known to achieve optimal mean response time [77, 152].

However, SRPT-k makes scheduling decisions which seem suboptimal in specific circumstances. For example, consider a system that has $k+1$ jobs, where k is the number of servers. SRPT-k will serve the k smallest jobs, quickly completing 2 of them. At this point, the system will have $k-1$ jobs, so a server will be idle. A different policy could have reordered service (prioritizing the largest job) to delay the point in time when a server first becomes idle. The server idleness wouldn't be a problem if no jobs would arrive in the future. However, when future jobs arrive, the system will have more work present than was necessary, resulting in poor response times.

Note that this scenario can only occur frequently under *moderate* load: If load is very high, there will rarely be only $k+1$ jobs in the system. If load is very low, it is unlikely that enough arrivals will occur in the near future for the extra work in the system to matter. Only at moderate load can both events occur in sequence.

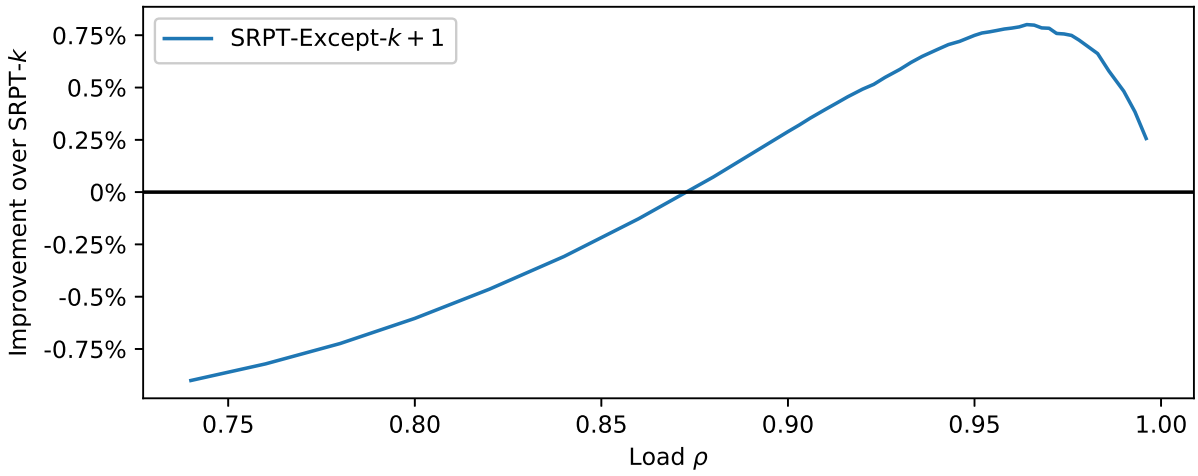


Figure 8.3: The relative mean response time improvement of SRPT-Except-$k+1$ (SEK) over multiserver SRPT. We define the improvement to be $1 - E[T^{SEK}]/E[T^{SRPT\text{-}k}]$, meaning that positive values represent a mean response time improvement over SRPT-k. Setting is $k = 2$ servers and a hyperexponential job size distribution in which 95% of jobs take $Exp(1.9)$ time, and 5% take $Exp(0.1)$ time. The squared coefficient of variation $C^2 = 10$, representing a relatively high-variance distribution. The largest improvement is by a margin of 0.8% at load $\rho = 0.964$. Simulation used 10^8 arrivals, loads up to 0.996.

We therefore ask:

> Does there exist a multiserver scheduling policy with better mean response time than SRPT-k, under moderate load?

In preliminary simulation-based work, we have found a case where a policy can achieve better mean response time than SRPT-k.

Let us define the "SRPT-Except-$k+1$" (SEK) policy as follows, with a switching parameter c:

- If there are k or fewer jobs in the system, serve all of them.
- If there are $k+2$ or more jobs in the system, serve the k jobs of least remaining size.
- If there are exactly $k+1$ jobs in the system, and k jobs have remaining size $\leq c$, and one job has remaining size $\geq c$, serve the $k-1$ jobs with least remaining size, and the job with largest remaining size.
- Otherwise, serve the k jobs of least remaining size.

SEK is the same as SRPT-k except for a carefully crafted exception in the case where $k+1$ jobs are present.

In Fig. 8.3, we simulate the relative mean response time improvement of SEK, as compared to SRPT-k. The setting is a $k = 2$ server system with a relatively high-variance job size distribution with squared coefficient of variation $C^2 \sim 10$. We choose switching parameter $c = 4$.

As Fig. 8.3 shows, SEK achieves better mean response times than SRPT-k for load $\rho > 0.87$, achieving the largest improvement ratio at load $\rho = 0.964$.

This raises several open questions: Can we prove that SEK always achieves better mean response time than SRPT-k, for all workloads, under some load ρ and some switching parameter c? How large of an improvement ratio can it achieve? Can it achieve arbitrarily large improvement ratios?

8.3.2 Tight Lower bounds on M/G/k Scheduling

To prove optimality for multiserver scheduling, our strategy has been to prove upper bounds on the mean response time of a specific policy, and compare that upper bound to a lower bound on the achievable mean response time for any scheduling policy. The lower bound that we have used is the mean response time of the resource-pooled SRPT policy, which combines all k servers into one giant server that runs k times as fast. In the heavy traffic limit, as the load ρ approaches the capacity of the system, this lower bound is tight. This tight lower bound allows us to prove our optimality results in Chapters 2, 3 and 5 and in subsequent papers [202].

However, outside of the heavy traffic limit, the resource-pooled SRPT system is too powerful, and the resulting bound is not tight. This prevents us from proving optimality results outside of the heavy-traffic limit.

One other important lower bound is the light-traffic bound. Even if all jobs received service immediately on arrival, the mean response time would be $kE[S]$, where k is the number of servers and S is the random variable representing a job's size. We have observed in simulation that for a wide range of lower loads, this bound is close to tight [78]. The challenge arises in lower bounding M/G/k scheduling in the intermediate regime, where neither of these lower bounds are tight.

One way to derive improved lower bounds is to use the WINE formula [202], which we made use of in Chapter 5. The WINE formula quantifies mean response time in terms of the relevant work W_x in the system:

$$E[T] = \frac{1}{\lambda} \int_0^\infty \frac{E[W_x]}{r^2(1-\rho_x)} dx$$

Relevant work W_x is defined to be the total remaining size of all jobs with remaining size less than x. The WINE formula is exact, so a lower bound on relevant work translates directly to a lower bound on mean response time.

Both the resource-pooled SRPT lower bound and the immediate-service lower bound can also be used to give lower bounds on mean relevant work $E[W_x]$. Using the WINE formula, we can combine these two bounds, selecting different bounds at different size thresholds x to give a better bound than with either formula individually.

We show these three lower bounds, as well as the mean response time for multiserver SRPT, in Fig. 8.4. There remains a significant gap between the best lower bound and the mean response time of SRPT. To close this gap, we must both find better scheduling policies than SRPT, as we discuss in Section 8.3.1, and prove tighter lower bounds on multiserver scheduling.

To improve our lower bounds, the WINE formula offers a promising path forward: We can prove stronger lower bounds on relevant work, in order to prove stronger lower bounds on response time.

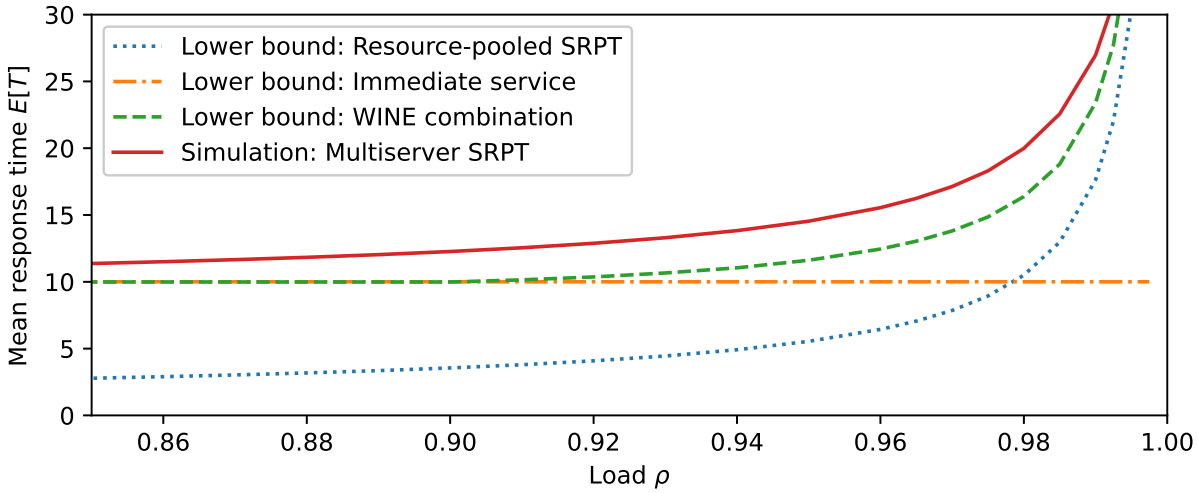


Figure 8.4: Three lower bounds on multiserver mean response time, compared to multiserver SRPT. The three lower bounds are the resource-pooled SRPT lower bound, the immediate-service lower bound, and the combination of the two using WINE. The setting is an M/M/10: an exponential job size distribution and $k = 10$ servers.

8.3.3 Optimal Dispatching to Gittins Queues

In Chapter 3, we studied the problem of dispatching and scheduling in an immediate-dispatch system in which exact job sizes were known. We invented the guardrails class of dispatching policies, which achieve asymptotically optimal mean response time when combined with SRPT scheduling at the servers.

The optimal single-server scheduling policy for mean response time when exact sizes are not known is the Gittins policy [72, 73, 199]. In this way, Gittins is the analogue of SRPT for the more general setting where exact sizes are not known. It is therefore natural to consider dispatching to queues using Gittins scheduling.

However, Gittins' single-server optimality is weaker than SRPT's single-server optimality. SRPT achieves optimal mean response time under an arbitrary arrival process. In contrast, the Gittins optimality result only holds under a Poisson arrival process (in an M/G/1). This becomes an issue when we dispatch to these queues, as the resulting arrival process to an individual queue at a specific server will be far from Poisson, even though the overall arrival process is Poisson.

We therefore ask:

> What is the optimal scheduling policy for dispatching to queues using Gittins scheduling, when exact size information is not available?

Within this question, there are two important scenarios to consider: When size estimates are available, and when no size information is available.

Dispatching with size estimates If size estimates are known, and are relatively accurate, it would make sense to combine a guardrails-like policy with a Preemptive-Shortest-Estimated-Job-First (PSJF-E) scheduling policy, which preemptively prioritizes jobs according to their esti-

mates. With relatively accurate estimates, the Gittins policy resembles the PSJF-E policy. Moreover, the PSJF-E policy is much easier to specify, and potentially more amenable to analysis.

In contrast to PSJF-E, SRPT-based policies can be fragile when faced with occasional inaccuracies [205]. Our results from Chapters 2 and 3 prove the same asymptotic optimality results for PSJF scheduling, as for SRPT scheduling. It therefore is promising to pursue combining PSJF-E scheduling with an estimate-based variant of guardrails.

Dispatching with unknown sizes With unknown sizes, all jobs are indistinguishable at the point of dispatching, so the dispatching policy should be based purely on the queue states, to balance the queues. In addition to knowing the number of jobs at each queue, the dispatcher would also know the amount of service already received (the *age*) of the jobs at each queue.

As a starting point, one could calculate the excepted amount of time that a given job would have to wait behind the jobs at each queue before it would complete service. A sensible greedy dispatching policy would then dispatch to the queue where that expected waiting time is the smallest. One could upgrade the policy by also considering the expected delay that the dispatched job would inflict on the other jobs at that queue.

8.3.4 Scheduling in the General MSJ Model

In Chapter 5, we considered the problem of optimal MSJ scheduling. We explored the power of two server-need setting, where ServerFilling-SRPT is asymptotically optimal, as well as the divisible server-need setting, where DivisorFilling-SRPT is optimal.

General server needs However, this leaves open the problem of optimal scheduling in the *general* server-need MSJ setting, where the server need distribution can be arbitrary. In this setting, there is a fundamental tension between serving the smallest jobs, and ensuring that servers are never left idle unnecessarily. For instance, consider a system where jobs have server needs 1 and 2, and $k = 3$ servers. Suppose that the 1-server jobs typically have smaller sizes than the 2-server jobs. We want to prioritize the 1-server jobs, because they have smaller sizes. However, if we complete all of the 1-server jobs, but still have many 2-server jobs left, we will be forced to leave a server idle. To achieve optimal mean response time, we must balance this tradeoff. The power-of-two and divisible settings avoid this tradeoff by ensuring that all k servers can be filled from among an arbitrary set of jobs, which does not hold in general.

Strategy: Minimum amount of 1-server jobs One strategy for handling the conflicting goals of prioritizing small jobs without wasting servers in the above $k = 3$ server scenario would be to maintain a minimum target amount of work of 1-server jobs in the system. If 1-server jobs are running low, then the policy would serve both a 1-server and a 2-server job, giving time for 1-server jobs to replenish. If there are plenty of 1-server jobs, then we could freely serve the jobs of least size.

Challenge: Lower bound One challenge for a policy that does not strictly prioritize the jobs of smallest size is that it would not achieve similar response time to resource-pooled SRPT. It would likely achieve similar response time to a different resource-pooled single-server scheduling policy. Proving the resulting response time to be optimal would be challenging, as there is no immediate lower bound ruling out better response times.

Alternative direction: Optimal finite-skip policy Another avenue of interest would be to study optimal *finite-skip* policies, the policies analyzed in Chapter 6. Finite-skip policies serve

jobs in near-FCFS order, which can be attractive for fairness and the tail of response time. The finite-skip approach is also better-suited to finding optimal non-preemptive policies, which can be advantageous in some settings.

Based on our results from Chapter 6, minimizing mean response time among finite-skip policies amounts to optimizing the throughput and relative completions of the saturated system. As a result, this can be considered a Markov Decision Process (MDP) optimization problem over the saturated system, which has a finite state space. One could therefore computationally derive the optimal policy. One could also hope to analytically determine the optimal solution to the MDP problem.

8.3.5 Scheduling for Intermediate Tail

In this thesis, we primarily considered two kinds of performance metrics for our scheduling policies: *mean* response time, in Chapters 2 to 6, and the *asymptotic tail* of response time, in Chapter 7. These two metrics can be thought of as representing two extremes: The metric of mean response time give equal weigh to every second of delay, while the metric of asymptotic tail overwhelmingly focuses on the most delayed jobs in the system.

Intermediate tail However, when computing system operators measure the performance of their systems, they typically use metrics that lie somewhere in between these two extremes. Their metrics consider more delayed jobs somewhat more heavily than less delayed jobs, but not overwhelmingly more heavily. Common metrics include the *tail probability* $P(T > t)$ and percentiles of response time such as T^{99}. We refer to these less-extreme metrics as measuring the *intermediate* tail.

Unfortunately, metrics such as tail probability and percentiles of response time are poorly suited to theoretical analysis. Their values are not even explicitly known for simple single-server scheduling policies such as FCFS and SRPT.

Moments of response time A better set of intermediate-tail metrics for theoretical analysis are the *moments of response time*, $E[T^\alpha]$ for $\alpha > 1$. The mean is the $\alpha = 1$ moment, and the asymptotic tail can be thought of as equivalent to the $\alpha \to \infty$ limit. As a result, moments $\alpha \in (1, \infty)$ cover the intermediate tail spectrum.

Moreover, for integer α, the moments of response time are explicitly known for many common single-server scheduling policies, via the Laplace-Steltjes transform of response time [104, 201]. Of particular importance is $\alpha = 2$, the expected square of mean response time, which is the natural first case after mean response time. We therefore ask:

What scheduling policy minimizes $E[T^2]$?

A natural scheduling policy to minimize $E[T^2]$ is to prioritize the job with the largest ratio of time in system to job size. We call this the t/s policy. The t/s policy can be seen as a variant of the $c\mu$ rule for minimizing weighted mean response time. In this case, a job's weight is it time in system. More generally, one could define the t^β/s policy, with an eye towards minimizing $E[T^{\beta+1}]$.

Very few scheduling policies that make use of time-in-system information have been analyzed. However, one of the few such classes of policies are the "Accumulating Priority Queue"

(APQ) policies, where a job's priority increases linearly with its time in system [21, 59, 159, 211]. However, analysis of APQ policies has thus far been limited to systems with finitely-many possible rates of priority accumulation, and principally to systems with 2 possible rates of priority accumulation. Expanding this analysis to handle continuous rates of priority accumulation would allow the analysis of the t/s policy, potentially demonstrating excellent $E[T^2]$ performance.

Bibliography

[1] Samuli Aalto, Urtzi Ayesta, and Rhonda Righter. On the Gittins index in the M/G/1 queue. *Queueing Systems*, 63(1):437–458, 2009. 3.9

[2] Joseph Abate and Ward Whitt. Asymptotics for M/G/1 low-priority waiting-time tail probabilities. *Queueing Systems*, 25(1-4):173–233, 1997. 7.3.1, 7.3.2, 7.4.3, 7.4.4

[3] Joseph Abate, Gagan L Choudhury, and Ward Whitt. Waiting-time tail probabilities in queues with long-tail service-time distributions. *Queueing systems*, 16(3-4):311–338, 1994. 7.4.3, 7.4.3, 7.4.4

[4] Joseph Abate, Gagan L. Choudhury, and Ward Whitt. Exponential approximations for tail probabilities in queues, I: Waiting times. *Operations Research*, 43(5):885–901, 1995. 7.3.1, 7.4.3, 7.4.4

[5] Larisa Afanaseva, Elena Bashtova, and Svetlana Grishunina. Stability analysis of a multi-server model with simultaneous service and a regenerative input flow. *Methodology and Computing in Applied Probability*, pages 1–17, 2019. 6.2

[6] Reza Aghajani and Kavita Ramanan. The limit of stationary distributions of many-server queues in the Halfin–Whitt regime. *Mathematics of Operations Research*, 45(3):1016–1055, 2020. 4.5.1

[7] E. Altman, U. Ayesta, and B. J. Prabhu. Load balancing in processor sharing systems. *Telecommunication Systems*, 47(1):35–48, Jun 2011. ISSN 1572-9451. doi: 10.1007/s11235-010-9300-8. 3.2

[8] 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. 4.5.2

[9] Danilo Ardagna, Simona Bernardi, Eugenio Gianniti, Soroush Karimian Aliabadi, Diego Perez-Palacin, and José Ignacio Requeno. Modeling performance of Hadoop applications: A journey from queueing networks to stochastic well formed nets. In *Algorithms and Architectures for Parallel Processing: 16th International Conference, ICA3PP 2016, Granada, Spain, December 14-16, 2016, Proceedings 15*, pages 599–613. Springer, 2016. 1.1

[10] Timothy G. Armstrong, Zhao Zhang, Daniel S. Katz, Michael Wilde, and Ian T. Foster. Scheduling many-task workloads on supercomputers: Dealing with trailing tasks. In *2010 3rd Workshop on Many-Task Computing on Grids and Supercomputers*, pages 1–10, 2010. 5.3.3

[11] Edward Arthurs and Joseph S. Kaufman. Sizing a message store subject to blocking criteria. In *Proceedings of the third international symposium on modelling and performance evaluation of computer systems: Performance of computer systems*, pages 547–564, 1979. 5.3.2

[12] Nir Avrahami and Yossi Azar. Minimizing total flow time and total completion time with immediate dispatching. In *Proceedings of the Fifteenth Annual ACM Symposium on Parallel Algorithms and Architectures*, SPAA '03, pages 11–18, New York, NY, USA, 2003. ACM. ISBN 1-58113-661-7. doi: 10.1145/777412.777415. 2.3.3, 3.4.3

[13] François Baccelli and Serguei Foss. On the saturation rule for the stability of queues. *Journal of Applied Probability*, 32(2):494–507, 1995. doi: 10.2307/3215303. 4.5.5, 6.1, 6.2, 6.3.2, 6.4.1, 1, 6.4.7

[14] Eitan Bachmat and Hagit Sarfati. Analysis of size interval task assignment policies. *SIGMETRICS Perform. Eval. Rev.*, 36(2):107–109, August 2008. ISSN 0163-5999. doi: 10.1145/1453175.1453199. 3.2

[15] Nikhil Bansal. On the average sojourn time under M/M/1/SRPT. *Operations research letters*, 33(2):195–200, 2005. 2.3.1

[16] Nikhil Bansal and David Gamarnik. Handling load with less stress. *Queueing Systems*, 54(1):45–54, 2006. 2.3.1

[17] Nikhil Bansal and Mor Harchol-Balter. Analysis of SRPT scheduling: Investigating unfairness. In *Proceedings of the 2001 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems*, SIGMETRICS '01, pages 279–290, New York, NY, USA, 2001. ACM. ISBN 1-58113-334-0. doi: 10.1145/378420.378792. 2.3, 2, 2.10.4

[18] 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, Hershey, PA, USA, 2021. 4.5.4

[19] 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. 4.5.4, 4.8

[20] 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. 4.5.4

[21] Blair Bilodeau and David A. Stanford. High-priority expected waiting times in the delayed accumulating priority queue with applications to health care kpis. *INFOR: Information Systems and Operational Research*, 60(3):285–314, 2022. doi: 10.1080/03155986.2022.2038962. 7.14.3, 8.3.5

[22] John T. Blake and Michael W. Carter. An analysis of emergency room wait time issues via computer simulation. *INFOR*, 34(4):263–273, November 1996. 7.2.3

[23] T. Bonald, M. Jonckheere, and A. Proutiére. Insensitive load balancing. In *Proceedings of the Joint International Conference on Measurement and Modeling of Computer Systems*, SIGMETRICS '04/Performance '04, pages 367–377, New York, NY, USA, 2004. ACM. ISBN 1-58113-873-3. doi: 10.1145/1005686.1005729. 3.2

[24] F. Bonomi. On job assignment for a parallel system of processor sharing queues. *IEEE Transactions on Computers*, 39(7):858–869, July 1990. ISSN 0018-9340. doi: 10.1109/12.55688. 3.2

[25] Sem C Borst, Onno J Boxma, and R Nunez-Queija. Heavy tails: the effect of the service discipline. In *International Conference on Modelling Techniques and Tools for Computer Performance Evaluation*, pages 1–30. Springer, 2002. 2.3

[26] Sem C Borst, Onno J Boxma, Rudesindo Núñez-Queija, and AP Zwart. The impact of the service discipline on delay asymptotics. *Performance Evaluation*, 54(2):175–206, 2003.

[27] Onno J. Boxma and Bert Zwart. Tails in scheduling. *SIGMETRICS Performance Evaluation Review*, 34(4):13–20, 2007. 1.4.1, 1.4.2, 1.4.2, 2.3, 7.1, 7.2.1, 7.2.1, 7.2.4, 7.3.1, 7.4.4, 7.5.2, 7.14.1, 8.1.3

[28] 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. 4.5.2

[29] Maury Bramson, Yi Lu, and Balaji Prabhakar. Asymptotic independence of queues under randomized load balancing. *Queueing Systems*, 71(3):247–292, Jul 2012. ISSN 1572-9443. doi: 10.1007/s11134-012-9311-0. 3.2

[30] Percy H Brill. A brief outline of the level crossing method in stochastic models. *CORS Bulletin*, 34(4):9–21, 2000. 7.4.4

[31] 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. 1.3.3, 4.1, 4.5.5, 4.5.5, 5.3.2, 6.2, 6.3.1

[32] Richard B Bunt. Scheduling techniques for operating systems. *Computer*, 9(10):10–17, 1976. 2.2

[33] 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. 4.4.4, 4.5.5, 4.5.5, 5.1, 5.3.3, 2, 5.8, 6.10.1

[34] Patrick Chareka. A finite-interval uniqueness theorem for bilateral Laplace transforms. *International Journal of Mathematics and Mathematical Sciences*, 2007:6 pages, 2007. ISSN 0161-1712, 1687-0425. 7.8.1

[35] Y. Chen, S. Iyer, X. Liu, D. Milojicic, and A. Sahai. SLA decomposition: Translating service level objectives to system level thresholds. In *Fourth International Conference on Autonomic Computing (ICAC'07)*, page 10 pages, 2007. 7.2.3

[36] Yuan-Hsiu Chen and Pao-Ann Hsiung. Hardware task scheduling and placement in operating systems for dynamically reconfigurable SoC. In Laurence T. Yang, Makoto Amamiya, Zhen Liu, Minyi Guo, and Franz J. Rammig, editors, *Embedded and Ubiquitous Comput-*

ing – EUC 2005, pages 489–498, Berlin, Heidelberg, 2005. Springer Berlin Heidelberg. ISBN 978-3-540-32295-5. 2.2

[37] Hyun-Duk Cho, Ph D Principal Engineer, Kisuk Chung, and Taehoon Kim. Benefits of the big.LITTLE architecture. *EETimes, Feb*, 2012. 4.4.1

[38] Walfredo Cirne and Francine Berman. A model for moldable supercomputer jobs. In *Proceedings 15th International Parallel and Distributed Processing Symposium. IPDPS 2001*, page 8 pp., 2001. 5.2.3

[39] A Bruce Clarke. A waiting line process of Markov type. *The Annals of Mathematical Statistics*, pages 452–459, 1956. 6.3.3, 6.13.1

[40] Jacob Willem Cohen. *The single server queue*. Elsevier, 2012. 1.1

[41] Mark Crovella, Robert Frangioso, and Mor Harchol-Balter. Connection scheduling in web servers. In *USENIX Symposium on Internet Technologies and Systems*, 1999. 2

[42] Mark E. Crovella, Murad S. Taqqu, and Azer Bestavros. Heavy-tailed probability distributions in the World Wide Web. In *A Practical Guide To Heavy Tails*, chapter 1, pages 1–23. Chapman & Hall, New York, 1998. 1.1, 7.2.2

[43] JG Dai, AB Dieker, and Xuefeng Gao. Validity of heavy-traffic steady-state approximations in many-server queues with abandonment. *Queueing Systems*, 78(1):1–29, 2014. 4.5.1

[44] Daryl J Daley. Some results for the mean waiting-time and workload in GI/GI/k queues. *Frontiers in queueing: models and applications in science and engineering*, pages 35–59, 1997. 1.2.2

[45] Mark M. Davis. How long should a customer wait for service? *Decision Sciences*, 22(2): 421–434, 1991. 7.2.3

[46] Rodolpho G. de Siqueira and Daniel R. Figueiredo. A control-based load balancing algorithm with flow control for dynamic and heterogeneous servers. In *Anais do XXXV Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos*, Porto Alegre, RS, Brasil, 2017. SBC. 3.2

[47] Mohammad Delasay, Armann Ingolfsson, and Bora Kolfal. Modeling load and overwork effects in queueing systems with adaptive service rates. *Operations Research*, 64(4):867–885, 2016. 6.3.3, 6.13.1

[48] 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. 4.4.3

[49] Sherwin Doroudi. *Stochastic analysis of maintenance and routing policies in queueing systems*. PhD thesis, Carnegie Mellon University, 2016. 6.3.3

[50] Sherwin Doroudi, Esa Hyytiä, and Mor Harchol-Balter. Value driven load balancing. *Performance Evaluation*, 79:306–327, 2014. ISSN 0166-5316. doi: https://doi.org/10.1016/j.peva.2014.07.019. Special Issue: Performance 2014. 3.10.3, 3.10.3

[51] Douglas G Down and Rong Wu. Multi-layered round robin routing for parallel servers. *Queueing Systems*, 53(4):177–188, 2006. 1, 2.3.5, 3.2, 3.4.3

[52] Douglas G Down, H Christian Gromoll, and Amber L Puha. Fluid limits for shortest remaining processing time queues. *Mathematics of Operations Research*, 34(4):880–911, 2009. 2.3.1

[53] Allen B. Downey. Using queue time predictions for processor allocation. In *workshop on Job Scheduling Strategies for Parallel Processing*, pages 35–57. Springer, 1997. 5.2.3

[54] 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. 4.5.2

[55] 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. 4.5.2

[56] Said El Kafhali and Khaled Salah. Stochastic modelling and analysis of cloud computing data center. In *2017 20th Conference on Innovations in Clouds, Internet and Networks (ICIN)*, pages 122–126, 2017. doi: 10.1109/ICIN.2017.7899401. 1.1

[57] Patrick Eschenfeldt and David Gamarnik. Join the shortest queue with many servers. The heavy-traffic asymptotics. *Mathematics of Operations Research*, 43(3):867–886, 2018. 1.2.2

[58] Yoav Etsion and Dan Tsafrir. A short survey of commercial cluster batch schedulers. *School of Computer Science and Engineering, The Hebrew University of Jerusalem*, 44221:2005–13, 2005. 5.3.3, 6.1, 6.2

[59] Val Andrei Fajardo and Steve Drekic. Waiting time distributions in the preemptive accumulating priority queue. *Methodology and Computing in Applied Probability*, 19(1): 255–284, 2017. 7.14.3, 8.3.5

[60] Dror G. Feitelson and Larry Rudolph. Toward convergence in job schedulers for parallel supercomputers. In Dror G. Feitelson and Larry Rudolph, editors, *Job Scheduling Strategies for Parallel Processing*, pages 1–26, Berlin, Heidelberg, 1996. Springer Berlin Heidelberg. ISBN 978-3-540-70710-3. 1.3, 5.1, 5.3.3

[61] 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, New York, NY, USA, 2004. Springer. 4.4.4, 4.5.5, 4.5.5, 5.3.3, 6.1, 6.2

[62] Hanhua Feng, Vishal Misra, and Dan Rubenstein. Optimal state-free, size-aware dispatching for heterogeneous M/G/-type systems. *Performance Evaluation*, 62(1):475 – 492, 2005. ISSN 0166-5316. doi: https://doi.org/10.1016/j.peva.2005.07.031. Performance 2005. 3.2

[63] Dimitrios Filippopoulos and Helen Karatza. An M/M/2 parallel system model with pure space sharing among rigid jobs. *Mathematical and Computer Modelling*, 45(5):491 – 530, 2007. ISSN 0895-7177. 1.3.3, 4.1, 5.3.2, 6.2, 6.3.1

[64] R. D. Foley and D. R. McDonald. Join the shortest queue: Stability and exact asymptotics.

The Annals of Applied Probability, 11(3):569–607, 2001. ISSN 10505164. 1.2.2, 3.10.2

[65] Serguei Foss and Takis Konstantopoulos. An overview of some stochastic stability methods. *Journal of the Operations Research Society of Japan*, 47(4):275–303, 2004. 4.5.5, 6.1, 6.2, 6.3.2, 6.4.1, 6.4.7

[66] Eric J. Friedman and Shane G. Henderson. Fairness and efficiency in web server protocols. In *Proceedings of the 2003 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems*, SIGMETRICS '03, page 229–237, New York, NY, USA, 2003. Association for Computing Machinery. ISBN 1581136641. 1.4.2, 7.3.2

[67] Eric J Friedman and Gavin Hurley. Protective scheduling. Technical report, Cornell University Operations Research and Industrial Engineering, 2003. 1.4.2, 7.3.2

[68] S. W. Fuhrmann and Robert B. Cooper. Stochastic decompositions in the M/G/1 queue with generalized vacations. *Operations Research*, 33(5):1117–1129, October 1985. ISSN 0030-364X, 1526-5463. 7.3.3

[69] David Gamarnik and Petar Momčilović. Steady-state analysis of a multiserver queue in the Halfin-Whitt regime. *Advances in Applied Probability*, 40(2):548–577, 2008. 4.5.1

[70] Misikir Eyob Gebrehiwot, Samuli Aalto, and Pasi Lassila. Energy-aware server with SRPT scheduling: Analysis and optimization. In Gul Agha and Benny Van Houdt, editors, *Quantitative Evaluation of Systems*, pages 107–122, Cham, 2016. Springer International Publishing. ISBN 978-3-319-43425-4. 2.3

[71] 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. 1.3.2, 4.1, 4.5.5, 4.5.5, 5.2.1, 5.1, 5.3.4, 6.3.1

[72] John Gittins, Kevin Glazebrook, and Richard Weber. *Multi-armed bandit allocation indices*. John Wiley & Sons, 2011. 1.1, 1.2.6, 1.3.5, 2.10.2, 3.9, 3.10.2, 5.2.3, 5.3.1, 5.6.1, 5.10.1, 8.1.2, 8.2.1, 8.3.3

[73] John C Gittins. Bandit processes and dynamic allocation indices. *Journal of the Royal Statistical Society: Series B (Methodological)*, 41(2):148–164, 1979. 1.1, 1.2.6, 1.3.5, 2.10.2, 3.10.2, 5.10.1, 8.1.2, 8.2.1, 8.3.3

[74] Peter W Glynn, Assaf Zeevi, et al. Bounding stationary expectations of Markov processes. *Markov processes and related topics: a Festschrift for Thomas G. Kurtz*, 4:195–214, 2008. 6.9

[75] David A Goldberg and Yuan Li. Simple and explicit bounds for multi-server queues with universal $1/(1-\rho)$ scaling. *arXiv preprint arXiv:1706.04628*, 2017. 4.5.1

[76] Mingwei Gong and Carey Williamson. Simulation evaluation of hybrid SRPT scheduling policies. In *Modeling, Analysis, and Simulation of Computer and Telecommunications Systems, 2004.(MASCOTS 2004). Proceedings. The IEEE Computer Society's 12th Annual International Symposium on*, pages 355–363. IEEE, 2004. 2.3.4

[77] R.L. Graham, E.L. Lawler, J.K. Lenstra, and A.H.G.Rinnooy Kan. Optimization and approximation in deterministic sequencing and scheduling: a survey. In P.L. Hammer, E.L. Johnson, and B.H. Korte, editors, *Discrete Optimization II*, volume 5 of *Annals*

of Discrete Mathematics, pages 287–326. Elsevier, 1979. doi: https://doi.org/10.1016/S0167-5060(08)70356-X. 2.10.3, 8.3.1

[78] Isaac Grosof. Open problem—M/G/k/SRPT under medium load. *Stochastic Systems*, 9 (3):297–298, 2019. doi: 10.1287/stsy.2019.0042. 1, 8.3.2

[79] Isaac Grosof. The RESET technique for multiserver-job analysis. *ACM SIGMETRICS Performance Evaluation Review*, 2023. To appear. 6

[80] Isaac Grosof and Mor Harchol-Balter. 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*, ApPLIED 2023, New York, NY, USA, 2023. Association for Computing Machinery. ISBN 9798400701283. doi: 10.1145/3584684.3597264. 4.10.1, 4.10.4, 4.10.4

[81] Isaac Grosof, Ziv Scully, and Mor Harchol-Balter. SRPT for multiserver systems. *Performance Evaluation*, 127-128:154–175, 2018. ISSN 0166-5316. 2, 4.6.4, 5.2.2, 5.2.5, 5.3.1, 5.4.1, 5.5.2, 5.5.2, 5.5.2, 5.5.2, 5.5.2

[82] Isaac Grosof, Ziv Scully, and Mor Harchol-Balter. Load balancing guardrails: Keeping your heavy traffic on the road to low response times. *Proc. ACM Meas. Anal. Comput. Syst.*, 3(2), jun 2019. 3, 5.2.5

[83] Isaac Grosof, Mor Harchol-Balter, and Alan Scheller-Wolf. Stability for two-class multiserver-job systems. *arXiv preprint arXiv:2010.00631*, 2020. 1.3.2, 4.1, 4.4.4, 6.1, 6.3.1, 6.3.2, 6.12, 6.13.3

[84] Isaac Grosof, Mor Harchol-Balter, and Alan Scheller-Wolf. WCFS: A new framework for analyzing multiserver systems. *arXiv preprint arXiv:2109.12663*, 2021. Electronic companion (full version) of the paper by the same name in Queueing Systems. 4, 5.7, 5.7.2

[85] Isaac Grosof, Kunhe Yang, Ziv Scully, and Mor Harchol-Balter. Nudge: Stochastically improving upon FCFS. In *Abstract Proceedings of the 2021 ACM SIGMETRICS / International Conference on Measurement and Modeling of Computer Systems*, SIGMETRICS '21, page 11–12, New York, NY, USA, 2021. Association for Computing Machinery. ISBN 9781450380720. doi: 10.1145/3410220.3460102. 7

[86] Isaac Grosof, Mor Harchol-Balter, and Alan Scheller-Wolf. WCFS: A new framework for analyzing multiserver systems. *Queueing Systems*, 2022. 4, 5.2.1, 5.2.1, 5.1, 5.3.2, 5.4.2, 5.7, 5.8, 6.2, 6.3.1, 6.10, 6.11.2

[87] Isaac Grosof, Ziv Scully, Mor Harchol-Balter, and Alan Scheller-Wolf. Optimal scheduling in the multiserver-job model under heavy traffic. *Proc. ACM Meas. Anal. Comput. Syst.*, 6(3), dec 2022. doi: 10.1145/3570612. 5, 6.2, 6.3.1

[88] Isaac Grosof, Mor Harchol-Balter, and Alan Scheller-Wolf. New stability results for multiserver-job models via product-form saturated systems. *MAthematical performance Modeling and Analysis (MAMA)*, 2023. 4.1, 6.1, 6.3.1, 6.3.2, 6.12, 6.13.3

[89] Shenoda Guirguis, Mohamed A Sharaf, Panos K Chrysanthis, Alexandros Labrinidis,

and Kirk Pruhs. Adaptive scheduling of web transactions. In *Data Engineering, 2009. ICDE'09. IEEE 25th International Conference on*, pages 357–368. IEEE, 2009. 2.2

[90] Mian Guo, Quansheng Guan, and Wende Ke. Optimal scheduling of VMs in queueing cloud computing systems with a heterogeneous workload. *IEEE Access*, 6:15178–15191, 2018. 5.1, 5.3.4

[91] 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. 4.4.2, 4.5.3, 4.5.3

[92] Varun Gupta and Takayuki Osogami. On Markov–Krein characterization of the mean waiting time in M/G/K and other queueing systems. *Queueing Systems*, 68:339–352, 2011. 1.2.2

[93] Varun Gupta and Takayuki Osogami. Tight moments-based bounds for queueing systems. In *Proceedings of the ACM SIGMETRICS Joint International Conference on Measurement and Modeling of Computer Systems*, SIGMETRICS '11, page 133–134, New York, NY, USA, 2011. Association for Computing Machinery. ISBN 9781450308144. doi: 10.1145/1993744.1993792. 1.2.2

[94] Varun Gupta and Neil Walton. Load balancing in the nondegenerate slowdown regime. *Operations Research*, 67(1):281–294, 2019. doi: 10.1287/opre.2018.1768. 3.1

[95] Varun Gupta, Mor Harchol-Balter, Alan Scheller Wolf, and Uri Yechiali. Fundamental characteristics of queues with fluctuating load. In *Proceedings of the joint international conference on Measurement and modeling of computer systems*, pages 203–215, 2006. 6.3.3, 6.13.1

[96] Varun Gupta, Mor Harchol Balter, Karl Sigman, and Ward Whitt. Analysis of join-the-shortest-queue routing for web server farms. *Performance Evaluation*, 64(9):1062–1081, 2007. ISSN 0166-5316. doi: https://doi.org/10.1016/j.peva.2007.06.012. Performance 2007. 3.1, 3.2

[97] Varun Gupta, Mor Harchol-Balter, Jim G Dai, and Bert Zwart. On the inapproximability of M/G/K: why two moments of job size distribution are not enough. *Queueing Systems*, 64:5–48, 2010. 1.2.2

[98] Bruce Hajek. Hitting-time and occupation-time bounds implied by drift analysis with applications. *Advances in Applied Probability*, 14(3):502–525, 1982. doi: 10.2307/1426671. 6.9

[99] Shlomo Halfin and Ward Whitt. Heavy-traffic limits for queues with many exponential servers. *Operations research*, 29(3):567–588, 1981. 1.2.2

[100] M. Harchol-Balter, Cuihong Li, T. Osogami, A. Scheller-Wolf, and M.S. Squillante. Analysis of task assignment with cycle stealing under central queue. In *23rd International Conference on Distributed Computing Systems, 2003. Proceedings.*, pages 628–637, 2003. doi: 10.1109/ICDCS.2003.1203514. 1.2.2

[101] Mor Harchol-Balter. The effect of heavy-tailed job size distributions on computer system design. In *Proceedings of the ASA-IMS Conference on Applications of Heavy Tailed*

Distributions in Economics, Engineering and Statistics, Washington, DC, June 1999. 1.1, 1.2.6, 7.2.2

[102] Mor Harchol-Balter. Job placement with unknown duration and no preemption. *ACM SIGMETRICS Performance Evaluation Review*, 28(4):3–5, 2001. 3.1

[103] Mor Harchol-Balter. Task assignment with unknown duration. *Journal of the ACM*, 49 (2):260–288, March 2002. 2.10.4, 7.2.2

[104] Mor Harchol-Balter. *Performance modeling and design of computer systems: queueing theory in action*. Cambridge University Press, 2013. 1.2, 1.1, 1.2.2, 1.4.2, 2.7, 2.8, 2.8.3, 2.8.3, 2.8.3, 3.2, 3.5.3, 3.5.4, 4.6.6, 5.5.3, 6.13.3, 7.8.1, 7.14.3, 8.3.5

[105] Mor Harchol-Balter. Open problems in queueing theory inspired by datacenter computing. *Queueing Systems: Theory and Applications*, 97(1):3–37, 2021. 7.2.3

[106] Mor Harchol-Balter. The multiserver job queueing model. *Queueing Systems*, 100(3): 201–203, 2022. 5.3.2

[107] Mor Harchol-Balter and Rein Vesilo. To balance or unbalance load in size-interval task allocation. *Probability in the Engineering and Informational Sciences*, 24(2):219–244, 2010. doi: 10.1017/S0269964809990246. 3.1

[108] Mor Harchol-Balter, Mark E. Crovella, and Cristina D. Murta. On choosing a task assignment policy for a distributed server system. *Journal of Parallel and Distributed Computing*, 59(2):204–228, 1999. ISSN 0743-7315. doi: https://doi.org/10.1006/jpdc.1999.1577. 1.2.2, 1.2.3, 3.1, 3.2, 3.10.2, 7.2.2

[109] Mor Harchol-Balter, Karl Sigman, and Adam Wierman. Asymptotic convergence of scheduling policies with respect to slowdown. *Performance Evaluation*, 49(1):241–256, 2002. ISSN 0166-5316. doi: https://doi.org/10.1016/S0166-5316(02)00132-3. Performance 2002. 2.10.4

[110] Mor Harchol-Balter, Bianca Schroeder, Nikhil Bansal, and Mukesh Agrawal. Size-based scheduling to improve web performance. *ACM Trans. Comput. Syst.*, 21(2):207–233, May 2003. ISSN 0734-2071. doi: 10.1145/762483.762486. 2.2, 2, 3.2, 3.8.3

[111] 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, Dec 2005. ISSN 1572-9443. doi: 10.1007/s11134-005-2898-7. 1.2.2, 2.3.2

[112] Mor Harchol-Balter, Alan Scheller-Wolf, and Andrew R. Young. Surprising results on task assignment in server farms with high-variability workloads. In *Proceedings of the Eleventh International Joint Conference on Measurement and Modeling of Computer Systems*, SIGMETRICS '09, pages 287–298, New York, NY, USA, 2009. ACM. ISBN 978-1-60558-511-6. doi: 10.1145/1555349.1555383. 3.1, 3.2

[113] Yige Hong. Sharp zero-queueing bounds for multi-server jobs. *SIGMETRICS Perform. Eval. Rev.*, 49(2):66–68, jan 2022. ISSN 0163-5999. 1.3.3, 4.5.5, 5.3.2, 6.2, 6.3.1

[114] Yige Hong and Ziv Scully. Performance of the Gittins policy in the G/G/1 and G/G/k, with and without setup times. *arXiv preprint arXiv:2304.13231*, 2023. 2.10.2

[115] Yige Hong and Weina Wang. Sharp waiting-time bounds for multiserver jobs. In *Pro-*

ceedings of the Twenty-Third International Symposium on Theory, Algorithmic Foundations, and Protocol Design for Mobile Networks and Mobile Computing, MobiHoc '22, page 161–170, New York, NY, USA, 2022. Association for Computing Machinery. ISBN 9781450391658. doi: 10.1145/3492866.3549717. 1.3.3, 4.5.5, 6.2, 6.3.1

[116] Leora I. Horwitz, Jeremy Green, and Elizabeth H. Bradley. US emergency department performance on wait time and length of visit. *Annals of Emergency Medicine*, 55(2):133 – 141, 2010. ISSN 0196-0644. 7.2.3

[117] Esa Hyytiä, Samuli Aalto, and Aleksi Penttinen. Minimizing slowdown in heterogeneous size-aware dispatching systems. In *Proceedings of the 12th ACM SIGMETRICS/PERFORMANCE Joint International Conference on Measurement and Modeling of Computer Systems*, SIGMETRICS '12, pages 29–40, New York, NY, USA, 2012. ACM. ISBN 978-1-4503-1097-0. doi: 10.1145/2254756.2254763. 2.2, 2.8

[118] Esa Hyytiä, Aleksi Penttinen, and Samuli Aalto. Size- and state-aware dispatching problem with queue-specific job sizes. *European Journal of Operational Research*, 217(2): 357 – 370, 2012. ISSN 0377-2217. doi: https://doi.org/10.1016/j.ejor.2011.09.029. 3.6, 3.6.1, 3.7

[119] Minnesota Supercomputing Institute. Queues, 2020. URL `https://www.msi.umn.edu/queues`. 5.3.3

[120] 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. 1.3, 5.1, 5.3.3, 6.1, 6.2

[121] Bart Kamphorst and Bert Zwart. Heavy-traffic analysis of sojourn time under the foreground-background scheduling policy. *arXiv preprint arXiv:1712.03853*, 2017. 2.8.3, 2.8.3

[122] S Karlin and HM Taylor. A first course in stochastic processes, 1975. 1.1

[123] 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. 4.5.2

[124] J. Kiefer and J. Wolfowitz. On the theory of queues with many servers. *Transactions of the American Mathematical Society*, 78(1):1–18, 1955. ISSN 00029947. 1.2.3

[125] Sung Shick Kim. *M/M/s queueing system where customers demand multiple server use*. PhD thesis, Southern Methodist University, 1979. 5.3.2

[126] J. F. C. Kingman. A martingale inequality in the theory of queues. *Mathematical Proceedings of the Cambridge Philosophical Society*, 60(2):359–361, 1964. 7.3.2

[127] J. F. C. Kingman. Inequalities in the theory of queues. *Journal of the Royal Statistical Society: Series B (Methodological)*, 32(1):102–110, 1970. 1.2.2, 7.3.2

[128] JFC Kingman. Some inequalities for the queue GI/G/1. *Biometrika*, 49(3/4):315–324, 1962. 4.5.1

[129] John F. C. Kingman. On queues in which customers are served in random order. *Mathe-*

matical Proceedings of the Cambridge Philosophical Society, 58(1):79–91, January 1962. ISSN 0305-0041, 1469-8064. 7.3.3

[130] Leonard Kleinrock. *Queueing Systems, Volume 2: Computer Applications*. Wiley, New York, NY, 1976. ISBN 978-0-471-49111-8. 7.3.3

[131] Charles Knessl and Yongzhi Peter Yang. An exact solution for an M(t)/M(t)/1 queue with time-dependent arrivals and service. *Queueing systems*, 40:233–245, 2002. 6.3.3, 6.13.1

[132] Julian Köllerström. Heavy traffic theory for queues with several servers. I. *Journal of Applied Probability*, 11(3):544–552, 1974. doi: 10.2307/3212698. 4.5.1, 4.5.1

[133] Julian Köllerström. Heavy traffic theory for queues with several servers. II. *Journal of Applied Probability*, 16(2):393–401, 1979. doi: 10.2307/3212906. 4.5.1, 4.5.1, 4.5.1

[134] A. M. Lee and P. A. Longton. Queueing processes associated with airline passenger check-in. *Journal of the Operational Research Society*, 10(1):56–71, 1959. doi: 10.1057/jors.1959.5. 1.2.2

[135] Stefano Leonardi and Danny Raz. Approximating total flow time on parallel machines. In *Proceedings of the twenty-ninth annual ACM symposium on Theory of computing*, pages 110–119. ACM, 1997. 1.1, 1.2.1, 2.2, 2.2, 2.3, 2.3.3, 2.6.2, 3.4.3

[136] Stefano Leonardi and Danny Raz. Approximating total flow time on parallel machines. *Journal of Computer and System Sciences*, 73(6):875 – 891, 2007. ISSN 0022-0000. doi: https://doi.org/10.1016/j.jcss.2006.10.018. 1.1, 2.2, 2.2, 2.3, 2.3.3, 2.6.2

[137] Quan-Lin Li, John C. S. Lui, and Yang Wang. *A Matrix-Analytic Solution for Randomized Load Balancing Models with PH Service Times*, pages 240–253. Springer Berlin Heidelberg, Berlin, Heidelberg, 2011. ISBN 978-3-642-25575-5. doi: 10.1007/978-3-642-25575-5_20. 3.2

[138] Minghong Lin, Adam Wierman, and Bert Zwart. Heavy-traffic analysis of mean response time under shortest remaining processing time. *Performance Evaluation*, 2011. ISSN 0166-5316. doi: https://doi.org/10.1016/j.peva.2011.06.001. 2.3.1, 2.7, 2.7, 3.5.5

[139] 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. 4.5.2

[140] Greg Linden. Marissa Mayer at Web 2.0, 2006. 1.4

[141] 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. 1.2.3

[142] John D. C. Little. A proof for the queuing formula: L = λw. *Operations Research*, 9(3):383–387, 1961. doi: 10.1287/opre.9.3.383. 1.4.3

[143] Zhen Liu and Rhonda Righter. Optimal load balancing on distributed homogeneous unreliable processors. *Operations Research*, 46(4):563–573, 1998. doi: 10.1287/opre.46.4.563. 3.2

[144] Richard Loulou. Multi-channel queues in heavy traffic. *Journal of Applied Probability*, 10(4):769–777, 1973. doi: 10.2307/3212380. 4.5.1

[145] David M Lucantoni and Marcel F Neuts. Some steady-state distributions for the MAP/SM/1 queue. *Stochastic Models*, 10(3):575–598, 1994. 6.3.3

[146] Syed Hamid Hussain Madni, Muhammad Shafie Abd Latiff, Mohammed Abdullahi, Shafi'i Muhammad Abdulhamid, and Mohammed Joda Usman. Performance comparison of heuristic algorithms for task scheduling in iaas cloud computing environment. *PLOS ONE*, 12(5):1–26, 05 2017. doi: 10.1371/journal.pone.0176321. 6.1, 6.2

[147] S. T. Maguluri and R. Srikant. Scheduling jobs with unknown duration in clouds. *IEEE/ACM Transactions on Networking*, 22(6):1938–1951, 2014. 1.3.2, 4.5.5, 4.5.5, 6.3.1

[148] 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. 1.3.2, 4.1, 4.4.4, 4.5.5, 4.5.5, 5.2.1, 5.1, 5.3.4, 5.8, 5.10.3

[149] Rahul Mangharam, Mustafa Demirhan, Ragunathan Rajkumar, and Dipankar Raychaudhuri. Size matters: Size-based scheduling for MPEG-4 over wireless channels. In *Multimedia Computing and Networking 2004*, volume 5305, pages 110–123. International Society for Optics and Photonics, 2003. 2.2

[150] 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. 4.4.1

[151] William A Massey. Asymptotic analysis of the time dependent M/M/1 queue. *Mathematics of Operations Research*, 10(2):305–327, 1985. 6.3.3, 6.13.1

[152] Robert McNaughton. Scheduling with deadlines and loss functions. *Management Science*, 6(1):1–12, 1959. doi: 10.1287/mnsc.6.1.1. 2.10.3, 8.3.1

[153] Sean Meyn. *Control techniques for complex networks*. Cambridge University Press, 2008. 6.7.1

[154] I. Mitrani and P.J.B. King. Multiprocessor systems with preemptive priorities. *Performance Evaluation*, 1(2):118 – 125, 1981. ISSN 0166-5316. doi: https://doi.org/10.1016/0166-5316(81)90014-6. 2.3.2

[155] Isi Mitrani and Ram Chakka. Spectral expansion solution for a class of Markov models: Application and comparison with the matrix-geometric method. *Performance Evaluation*, 23(3):241–260, 1995. 6.3.3

[156] M. Mitzenmacher. The power of two choices in randomized load balancing. *IEEE Transactions on Parallel and Distributed Systems*, 12(10):1094–1104, Oct 2001. ISSN 1045-9219. doi: 10.1109/71.963420. 3.2

[157] Masakiyo Miyazawa. Rate conservation laws: a survey. *Queueing Systems*, 15(1):1–58, 1994. 4.6.4

[158] Jeffrey C. Mogul and John Wilkes. Nines are not enough: Meaningful metrics for clouds. In *Proceedings of the Workshop on Hot Topics in Operating Systems (HotOS19)*, pages 136 – 141, USA, 2019. 7.2.3

[159] Maryam Mojalal, David A Stanford, and Richard J Caron. The lower-class waiting time distribution in the delayed accumulating priority queue. *INFOR: Information Systems and*

Operational Research, 58(1):60–86, 2020. 7.14.3, 8.3.5

[160] Behnam Montazeri, Yilong Li, Mohammad Alizadeh, and John Ousterhout. Homa: A receiver-driven low-latency transport protocol using network priorities. In *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication*, SIGCOMM '18, pages 221–235, New York, NY, USA, 2018. ACM. ISBN 978-1-4503-5567-4. doi: 10.1145/3230543.3230564. 3.2, 3.8.3

[161] Evsey Morozov and Alexander Rumyantsev. Stability analysis of a MAP/M/s cluster model by matrix-analytic method. In Dieter Fiems, Marco Paolieri, and Agapios N. Platis, editors, *Computer Performance Engineering*, pages 63–76, Cham, 2016. Springer International Publishing. ISBN 978-3-319-46433-6. 6.2

[162] Debankur Mukherjee, Sem Borst, Johan van Leeuwaarden, and Phil Whiting. Universality of power-of-d load balancing schemes. *SIGMETRICS Perform. Eval. Rev.*, 44(2):36–38, September 2016. ISSN 0163-5999. doi: 10.1145/3003977.3003990. 3.2

[163] Debankur Mukherjee, Sem C Borst, and Johan SH Van Leeuwaarden. Asymptotically optimal load balancing topologies. *Proceedings of the ACM on Measurement and Analysis of Computing Systems*, 2(1):1–29, 2018. 3.10.3

[164] Jayakrishnan Nair, Adam Wierman, and Bert Zwart. Tail-robust scheduling via limited processor sharing. *Performance Evaluation*, 67(11):978 – 995, 2010. ISSN 0166-5316. 7.2.1, 7.2.3, 7.3.1

[165] 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. 4.4.1

[166] Marcel F Neuts. The single server queue with Poisson input and semi-Markov service times. *Journal of Applied Probability*, 3(1):202–230, 1966. 6.3.3, 6.13.1

[167] GF Newell. Queues with time-dependent arrival rates. III—a mild rush hour. *Journal of Applied Probability*, 5(3):591–606, 1968. 6.3.3

[168] GF Newell. Queues with time-dependent arrival rates. II—the maximum queue and the return to equilibrium. *Journal of Applied Probability*, 5(3):579–590, 1968.

[169] Gordon Frank Newell. Queues with time-dependent arrival rates I—the transition through saturation. *Journal of Applied Probability*, 5(2):436–451, 1968. 6.3.3

[170] Misja Nuyens and Wemke van der Weij. Monotonicity in the limited processor-sharing queue. *Stochastic Models*, 25(3):408–419, 2009. doi: 10.1080/15326340903088545. 4.4.2, 4.5.3

[171] Misja Nuyens and Adam Wierman. The foreground–background queue: a survey. *Performance evaluation*, 65(3-4):286–307, 2008. 2.2

[172] Misja Nuyens, Adam Wierman, and Bert Zwart. Preventing large sojourn times using SMART scheduling. *Operations Research*, 56(1):88–101, 2008. 1.4.2, 7.2.3, 7.3.1, 7.3.2

[173] Natalia Osipova, Urtzi Ayesta, and Konstantin Avrachenkov. Optimal policy for multi-class scheduling in a single server queue. In *2009 21st International Teletraffic Congress*, pages 1–8, Paris, France, September 2009. IEEE. ISBN 978-1-4244-4744-2. 7.3.3

[174] Takayuki Osogami, Mor Harchol-Balter, Alan Wolf, and Li Zhang. Exploring threshold-based policies for load sharing. *Forty-second Annual Allerton Conference on Communication, Control, and Computing*, 2004. 1.2.2

[175] Takayuki Osogami, Mor Harchol-Balter, and Alan Scheller-Wolf. Analysis of cycle stealing with switching times and thresholds. *Performance Evaluation*, 61(4):347–369, 2005. ISSN 0166-5316. doi: https://doi.org/10.1016/j.peva.2004.09.003. 1.2.2

[176] Jamol Pender, Richard H. Rand, and Elizabeth Wesson. Queues with choice via delay differential equations. *International Journal of Bifurcation and Chaos*, 27(04):1730016, 2017. doi: 10.1142/S0218127417300166. 3.10.3

[177] Edwin Peng. Exact response time analysis of preemptive priority scheduling with switching overhead. *ACM SIGMETRICS Performance Evaluation Review*, 49(2):72–74, 2022. 2.10.4, 6.10.6

[178] 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. 4.4.3

[179] Efrat Perel and Uri Yechiali. Queues where customers of one queue act as servers of the other queue. *Queueing Systems*, 60:271–288, 2008. 6.3.3

[180] 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. 1.3.2, 4.5.5

[181] 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. 1.3.2, 4.5.5, 4.5.5, 5.2.1, 5.1, 5.3.4, 5.10.3, 6.3.1

[182] I.A. Rai, E.W. Biersack, and G. Urvoy-Keller. Size-based scheduling to improve the performance of short tcp flows. *IEEE Network*, 19(1):12–17, 2005. doi: 10.1109/MNET.2005.1383435. 2.10.1

[183] Idris A. Rai, Guillaume Urvoy-Keller, and Ernst W. Biersack. Analysis of LAS scheduling for job size distributions with high variance. *SIGMETRICS Perform. Eval. Rev.*, 31(1): 218–228, jun 2003. ISSN 0163-5999. doi: 10.1145/885651.781055. 2.10.1

[184] 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. 4.5.2

[185] Rhonda Righter and J. George Shanthikumar. Scheduling multiclass single server queueing systems to stochastically maximize the number of successful departures. *Probability in the Engineering and Informational Sciences*, 3(3):323–333, 1989. doi: 10.1017/S0269964800001194. 2.2, 2.8, 2.8.3

[186] Alexander Rumyantsev. Stability of multiclass multiserver models with automata-type phase transitions. In *Proceedings of the second international workshop on stochastic modeling and applied research of technology (SMARTY 2020)*, volume 2792, pages 213–

225, 2020. 1.3.2, 6.3.1

[187] Alexander Rumyantsev and Evsey Morozov. Stability criterion of a multiserver model with simultaneous service. *Annals of Operations Research*, 252(1):29–39, 2017. 4.1, 4.5.5, 4.5.5, 5.1, 5.3.2, 6.1, 6.2, 6.3.1, 6.3.2, 6.12, 6.13.3

[188] Alexander Rumyantsev, Robert Basmadjian, Sergey Astafiev, and Alexander Golovin. Three-level modeling of a speed-scaling supercomputer. *Annals of Operations Research*, pages 1–29, 2022. 1.3.2, 5.3.2, 6.2

[189] Daan Rutten and Debankur Mukherjee. Load balancing under strict compatibility constraints. *Mathematics of Operations Research*, 2022. 3.10.3

[190] Daan Rutten and Debankur Mukherjee. Mean-field analysis for load balancing on spatial graphs. *arXiv preprint arXiv:2301.03493*, 2023. 3.10.3

[191] T. Sakurai. Approximating M/G/1 waiting time tail probabilities. *Stochastic Models*, 20 (2):173–191, 2004. 7.3.1, 7.4.3, 7.4.4

[192] 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. 4.5.2

[193] Alan Scheller-Wolf. *Finite moment conditions for stationary content processes with applications to fluid models and queues*. PhD thesis, Columbia University, 1996. URL `https://www.proquest.com/dissertations-theses/finite-moment-conditions-stationary-content/docview/304249458/se-2?accountid=9902`. 4.6.6

[194] Linus Schrage. A proof of the optimality of the shortest remaining processing time discipline. *Operations Research*, 16(3):687–690, 1968. 1.1, 1.2.5, 2.1, 2.2, 2.3, 2.3.5, 3.1, 3.2, 3.4.2, 3.4.2, 3.5.4, 3.5.6, 4.10.2, 5.1, 5.3.1, 7.2.3

[195] Linus E Schrage. The queue M/G/1 with feedback to lower priority queues. *Management Science*, 13(7):466–474, 1967. 1.2.3, 2.8.3, 2.8.3, 2.8.3, 7.3.3

[196] Linus E. Schrage and Louis W. Miller. The queue M/G/1 with the Shortest Remaining Processing Time discipline. *Operations Research*, 14(4):670–684, 1966. 1.2, 1.2.3, 1.2.5, 2.1, 2.2, 2.2, 2.3, 2.5.1, 2.5.1, 2.5.2, 2.6.3, 2.7, 5.3.1, 5.4.1, 7.3.3, 8.1.1

[197] Bianca Schroeder and Mor Harchol-Balter. Evaluation of task assignment policies for supercomputing servers: The case for load unbalancing and fairness. *Cluster Computing*, 7:151–161, 2004. 3.1

[198] Ziv Scully. WINE: A new queueing identity for analyzing scheduling policies in multiserver systems, 2021. URL `https://ziv.codes/pdf/wine-talk.pdf`. INFORMS Annual Meeting. 5.2.5, 4, 5.5.3, 5.6.4

[199] Ziv Scully and Mor Harchol-Balter. The Gittins policy in the M/G/1 queue. In *2021 19th International Symposium on Modeling and Optimization in Mobile, Ad hoc, and Wireless Networks (WiOpt)*, pages 1–8, 2021. 1.1, 1.2.6, 1.3.5, 2.10.2, 3.10.2, 5.3.1, 5.6.1, 5.10.1, 5.10.3, 8.1.2, 8.2.1, 8.3.3

[200] Ziv Scully and Mor Harchol-Balter. How to schedule near-optimally under real-world

constraints. *arXiv preprint arXiv:2110.11579*, 2021. 2.10.4

[201] 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):16:1–16:30, April 2018. ISSN 2476-1249. doi: 10.1145/3179419. 1.1, 1.4.2, 2.8.3, 2.8.3, 6.13.3, 7.2.4, 7.3.2, 7.3.3, 7.14.3, 8.3.5

[202] Ziv Scully, Isaac Grosof, and Mor Harchol-Balter. The Gittins policy is nearly optimal in the M/G/k under extremely general conditions. *Proc. ACM Meas. Anal. Comput. Syst.*, 4 (3), Nov. 2020. 1.2.6, 2.7, 2.10.2, 4, 5.2.5, 5.3.1, 5.5.2, 5.5.2, 5.5.2, 5.5.2, 4, 5.5.2, 5.5.3, 5.5.1, 5.5.3, 5.5.2, 5.5.3, 5.6.1, 5.6.2, 5.6.3, 5.6.3, 5.6.4, 5.6.3, 5.6.4, 5.6.4, 5.6.4, 5.6.3, 5.6.4, 8.1.2, 8.2.1, 8.3.2

[203] Ziv Scully, Lucas van Kreveld, Onno J. Boxma, Jan-Pieter Dorsman, and Adam Wierman. Characterizing policies with optimal response time tails under heavy-tailed job sizes. *Proceedings of the ACM on Measurement and Analysis of Computing Systems*, 4(2), June 2020. ISSN 2476-1249, 2476-1249. 7.3.1

[204] 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. 2.10.2, 5.2.5, 5.3.1, 5.5.2, 5.5.2, 5.5.2, 5.5.2, 5.10.1

[205] Ziv Scully, Isaac Grosof, and Michael Mitzenmacher. Uniform bounds for scheduling with job size estimates. In *13th Innovations in Theoretical Computer Science Conference, ITCS*, 2022. 3, 8.2.1, 8.3.3

[206] Karl Sigman and David D Yao. Finite moments for inventory processes. *The Annals of Applied Probability*, pages 765–778, 1994. 4.6.6

[207] Andrei Sleptchenko, Aart van Harten, and Matthieu van der Heijden. An exact solution for the state probabilities of the multi-class, multi-server queue with preemptive priorities. *Queueing Systems*, 50(1):81–107, May 2005. ISSN 1572-9443. doi: 10.1007/s11134-005-0359-y. 1.2.2, 2.3.2

[208] Leszek Sliwko. A taxonomy of schedulers–operating systems, clusters and big data frameworks. *Global Journal of Computer Science and Technology*, 2019. 6.1, 6.2

[209] Kut C. So and Jing-Sheng Song. Price, delivery time guarantees and capacity selection. *European Journal of Operational Research*, 111(1):28 – 49, 1998. ISSN 0377-2217. 7.2.3

[210] Srividya Srinivasan, Rajkumar Kettimuthu, Vijay Subramani, and Ponnuswamy Sadayappan. Characterization of backfilling strategies for parallel job scheduling. In *Proceedings. International Conference on Parallel Processing Workshop*, pages 514–519, 2002. 4.4.4, 4.5.5, 4.5.5, 5.3.3, 6.10.1

[211] David A Stanford, Peter Taylor, and Ilze Ziedins. Waiting time distributions in the accumulating priority queue. *Queueing Systems*, 77(3):297–330, 2014. 1.4.3, 7.3.2, 7.3.3, 7.14.3, 8.3.5

[212] Alexander L Stolyar. Maxweight scheduling in a generalized switch: State space collapse and workload minimization in heavy traffic. *The Annals of Applied Probability*, 14(1): 1–53, 2004. 1.3.2

[213] Alexander L. Stolyar and Kavita Ramanan. Largest weighted delay first scheduling: Large deviations and optimality. *Annals of Applied Probability*, 11(1):1–48, 2001. ISSN 10505164. 7.2.1

[214] Wei Tang, Zhiling Lan, Narayan Desai, Daniel Buettner, and Yongen Yu. Reducing fragmentation on torus-connected supercomputers. In *2011 IEEE International Parallel Distributed Processing Symposium*, pages 828–839, 2011. 5.3.3

[215] Wei Tang, Dongxu Ren, Zhiling Lan, and Narayan Desai. Adaptive metric-aware job scheduling for production supercomputers. In *2012 41st International Conference on Parallel Processing Workshops*, pages 107–115, 2012. 5.3.3

[216] 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. 4.4.2, 4.5.3, 4.5.3

[217] Oleg M. Tikhonenko. Generalized erlang problem for service systems with finite total capacity. *Problems of Information Transmission*, 41(3):243–253, 2005. 5.3.2

[218] 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. (document), 1.1, 1.3, 4.6, 4.4.4, 2, 5.2.1, 5.3, 6.2, 7.2.2

[219] Timothy L. Urban. Establishing delivery guarantee policies. *European Journal of Operational Research*, 196(3):959 – 967, 2009. ISSN 0377-2217. 7.2.3

[220] Nico M. van Dijk. Blocking of finite source inputs which require simultaneous servers with general think and holding times. *Operations Research Letters*, 8(1):45 – 52, 1989. ISSN 0167-6377. 5.3.2

[221] Aart Van Harten and Andrei Sleptchenko. On Markovian multi-class, multi-server queueing. *Queueing systems*, 43(4):307–328, 2003. 4.5.2

[222] Benny Van Houdt. On the stochastic and asymptotic improvement of First-Come First-Served and Nudge scheduling. *Proceedings of the ACM on Measurement and Analysis of Computing Systems*, 6(3):1–22, 2022. 1.4.6, 7.14.2

[223] Rein Vesilo, Mor Harchol-Balter, and Alan Scheller-Wolf. Scaling properties of queues with time-varying load processes: extensions and applications. *Probability in the Engineering and Informational Sciences*, 36(3):690–731, 2022. 6.3.3

[224] Chad Vizino, Nathan Stone, John Kochmar, and J. Ray Scott. Batch scheduling on the cray XT3. *CUG 2005*, 2005. 5.3.3

[225] Juan Wang and Wenming Guo. The application of backfilling in cluster systems. In *2009 WRI International Conference on Communications and Mobile Computing*, volume 3, pages 55–59, 2009. 5.1, 5.3.3, 6.10.1

[226] Weina Wang, Qiaomin Xie, and Mor Harchol-Balter. Zero queueing for multi-server jobs. *Proc. ACM Meas. Anal. Comput. Syst.*, 5(1), feb 2021. doi: 10.1145/3447385. 1.3.3, 5.3.2, 6.2, 6.3.1

[227] Richard R. Weber. On the optimal assignment of customers to parallel servers. *Journal of Applied Probability*, 15(2):406–413, 1978. doi: 10.2307/3213411. 3.1, 3.2

[228] 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, October 1964. ISSN 0030364X, 15265463. 7.3.3

[229] Ward Whitt. Blocking when service is required from several facilities simultaneously. *AT&T Technical Journal*, 64(8):1807–1856, 1985. 5.3.2

[230] Ward Whitt. Approximations for the GI/G/m queue. *Production and Operations Management*, 2(2):114–161, 1993. doi: https://doi.org/10.1111/j.1937-5956.1993.tb00094.x. 1.2.2

[231] Adam Wierman and Mor Harchol-Balter. Classifying scheduling policies with respect to unfairness in an M/GI/1. In *ACM SIGMETRICS Performance Evaluation Review*, pages 238–249. ACM, 2003. 2.2, 2.3, 3.8.3

[232] Adam Wierman and Bert Zwart. Is tail-optimal scheduling possible? *Operations Research*, 60(5):1249–1257, October 2012. ISSN 0030-364X, 1526-5463. 1.4.1, 1.4.2, 7.1, 7.2.4, 7.3.1, 7.5.2, 7.14.1, 8.1.3

[233] Adam Wierman, Mor Harchol-Balter, and Takayuki Osogami. Nearly insensitive bounds on smart scheduling. In *ACM SIGMETRICS Performance Evaluation Review*, pages 205–216. ACM, 2005. 2.2, 2.6.3, 2.6.3, 2.8.1, 2.8.2, 2.9, 3.8.3

[234] Wayne Winston. Optimality of the shortest line discipline. *Journal of Applied Probability*, 14(1):181–189, 1977. doi: 10.2307/3213271. 3.1, 3.2

[235] Ronald W. Wolff. Poisson arrivals see time averages. *Operations Research*, 30(2):223–231, 1982. doi: 10.1287/opre.30.2.223. 2.5.1, 2.6, 3.5.1, 3.5.3, 3.5.3, 7.8.3

[236] Rong Wu and Douglas G Down. Scheduling multi-server systems using foreground-background processing. In *The Forty-second Allerton Conference*. Citeseer, 2004. 2.3.5, 6

[237] Ruhan Xie and Ziv Scully. Reducing heavy-traffic response time with asymmetric dispatching. *MAthematical performance Modeling and Analysis (MAMA)*, 2023. 3.1, 3.10.2

[238] SF Yashkov and AS Yashkova. Processor sharing: A survey of the mathematical theory. *Automation and Remote Control*, 68(9):1662–1731, 2007. 4.4.2

[239] Ury Yechiali and Pinhas Naor. Queuing problems with heterogeneous arrivals and service. *Operations Research*, 19(3):722–734, 1971. 6.3.3

[240] Jiheng Zhang and Bert Zwart. Steady state approximations of limited processor sharing queues in heavy traffic. *Queueing Systems*, 60(3):227–246, 2008. 4.4.2, 4.5.3, 4.5.3

[241] 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. 4.5.3

[242] 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. 4.5.3, 4.5.3

[243] Zhisheng Zhao, Debankur Mukherjee, and Ruoyu Wu. Exploiting data locality to improve

performance of heterogeneous server clusters. *arXiv preprint arXiv:2211.16416*, 2022. 3.10.3

[244] Xingyu Zhou, Jian Tan, and Ness Shroff. Flexible load balancing with multi-dimensional state-space collapse: Throughput and heavy-traffic delay optimality. *Performance Evaluation*, 127-128:176 – 193, 2018. ISSN 0166-5316. doi: https://doi.org/10.1016/j.peva.2018.10.003. 3.2