Scheduling for Today's Computer Systems: Bridging Theory and Practice

Adam Wierman

2007
CMU-CS-07-126

School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

Thesis Committee:
Mor Harchol-Balter, chair
John Lafferty
Bruce Maggs
Alan Scheller-Wolf
Ward Whitt

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

This research was supported, in part, by an NSF Graduate Research Fellowship and a Siebel Scholar award. Additional funding was provided by a grant from the Pittsburgh Digital Greenhouse and NSF grants CCR-0133077, CCR-0311383, and CCR-9457766.

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: scheduling; queueing; fairness; tail behavior; response time; classification; multiserver; SRPT; LAS; FB; PS; FSP; PSJF; PLCFS; SJF; SMART; FOOLISH; protective; symmetric; closed system; M/G/1; web servers; routers; wireless; access points; databases

This thesis is dedicated to

my wife Sonya,
who's love and encouragement made this thesis possible,

and my parents,
who have always supported me in everything I've attempted.

Abstract

Scheduling is a fundamental technique for improving performance in computer systems. From web servers to routers to operating systems, how the bottleneck device is scheduled has an enormous impact on the performance of the system as a whole. Given the immense literature studying scheduling, it is easy to think that we already understand enough about scheduling. But, modern computer system designs have highlighted a number of disconnects between traditional analytic results and the needs of system designers. In particular, the idealized policies, metrics, and models used by analytic researchers do not match the policies, metrics, and scenarios that appear in real systems.

The goal of this thesis is to take a step towards modernizing the theory of scheduling in order to provide results that apply to today's computer systems, and thus ease the burden on system designers. To accomplish this goal, we provide new results that help to bridge each of the disconnects mentioned above. We will move beyond the study of idealized policies by introducing a new analytic framework where the focus is on scheduling heuristics and techniques rather than individual policies. By moving beyond the study of individual policies, our results apply to the complex hybrid policies that are often used in practice. For example, our results enable designers to understand how the policies that favor small job sizes are affected by the fact that real systems only have estimates of job sizes. In addition, we move beyond the study of mean response time and provide results characterizing the distribution of response time and the fairness of scheduling policies. These results allow us to understand how scheduling affects QoS guarantees and whether favoring small job sizes results in large job sizes being treated unfairly. Finally, we move beyond the simplified models traditionally used in scheduling research and provide results characterizing the effectiveness of scheduling in multiserver systems and when users are interactive. These results allow us to answer questions about the how to design multiserver systems and how to choose a workload generator when evaluating new scheduling designs.

Thesis Committee

Mor Harchol-Balter (Chair)
Computer Science Department
Carnegie Mellon University

John Lafferty
Computer Science Department and Machine Learning Department
Carnegie Mellon University

Bruce Maggs
Computer Science Department
Carnegie Mellon University

Alan Scheller-Wolf
Tepper School of Business
Carnegie Mellon University

Ward Whitt
Department of Industrial Engineering and Operations Research
Columbia University

Acknowledgements

I have spent nearly ten years at Carnegie Mellon, working through the undergraduate and graduate programs, and I consider myself extremely lucky to have had the opportunity to interact with such an amazing group of scholars and students. Carnegie Mellon is an exceptional environment and I can only hope that I will serve as a worthy representative of my alma matter throughout my future career.

Computer Science at Carnegie Mellon provides an amazingly supportive environment for its students, so I have many people that I wish to thank. First, I would like to thank my advisor, Mor Harchol-Balter. There is no aspect of research in which she did not provide excellent guidance. She has taught me how to take stock of the big picture, not only when presenting my work, but also as a guide for new directions. Her advice also extended beyond research to career planning and teaching, and I am extremely grateful for the time she spent working with me on these areas. Finally, Mor's boundless and infective energy is an enormous inspiration. There were many times during my graduate career when I came into a meeting with Mor frustrated about a problem or stuck on a proof, but I had a new excitement about the work by the end of the meeting (even when we had not actually made any concrete progress).

Next, I would like to thank Alan Scheller-Wolf, who in many ways acted as an advisor for the operations research half of my work. His guidance, both about research and beyond, has always been thoughtful and insightful. I also want to thank the other members of my thesis committee: John Lafferty, Bruce Maggs, and Ward Whitt, who each provided valuable feedback that helped me look at aspects of the thesis in a new light. In the same breath, I would like to thank Anupam Gupta for his advice over the years on research, teaching, and life.

I have had a amazing group of coauthors over my graduate career, and each of them deserves my gratitude. I would especially like to express my gratitude to Takayuki Osogami. Through research collaborations, presentation critiques, career advice, and more, Taka played a key role in my graduate career. And, equally importantly, grad school just would not have been as fun without him around. Bianca Schroeder also deserves a big thank you, not only for her role in our research collaborations, but also for her critiques of my presentations and for her help with my job search. Collaborations with Misja Nuyens, Bert Zwart, and Jorgen Olsen were amazingly fruitful given that we hardly ever were on the same continent. Thank you for taking the extra effort to collaborate via email. Onno Boxma and Ivo Adan gave me the opportunity to spend half a year working at the EURANDOM institute in Eindhoven, which not only led to a number of interesting collaborations, but also provided me a wonderful opportunity to experience a different culture first hand, in a way that a vacation can never provide. Also, thank you to the many graduate students and post-docs at EURANDOM who welcomed me and made me feel at home, especially Erik Winands, Marcel van Vuuren, and Maak Holmes.

I have been lucky to have the support of many generous people throughout my graduate career. I would like to give a special thank you everyone who attended SQUALL over the years, especially Varun Gupta, Paul Enders, and David McWherter. I would also like to thank the support staff in the department, which smoothly took care of all the administrative issues so that I could focus on my research, especially Charlotte Yano, Sharon Burks, and Catherine Copetas.

Of course, life did not end at the walls of Wean Hall, and I would like to acknowledge Brian Knudsen, Helen Lafferty, Mike Rossi, and Rob Reeder, who have been great friends and have kept me sane over the years. Also, I'd like to thank everyone who has helped with the Random Distance Run, especially the new RDR guru, Gaurav Veda.

Finally, I would like to thank my family. I would not be where I am today without the support and love of my parents, and this thesis would not be what it is without the encouragement and patience of my wife Sonya. She has truly been a partner in my research and writing, letting me vent my complaints and frustrations and keeping me on task (and off the golf course) when there was nothing I wanted to do less than work on writing the thesis. She is my inspiration in research and life, and I am excited to find out what lies ahead for us.

Table of Contents

List of Figures

List of Tables

PART I

Motivation and Background

CHAPTER 1

Introduction

Scheduling policies are implicitly (or explicitly) used everywhere that a resource needs to be allocated. Whenever there is contention for a limited resource, a queue builds, and a scheduling policy is used to determine the order in which the resource is allocated to satisfy requests.

This happens almost everywhere we venture in our daily lives. From restaurants and supermarkets, to banks and amusement parks, we queue for service in a variety of ways. In many convenience stores there is a single cash register where people line up, and are then served in First Come First Served (FCFS) order. In large supermarkets, there are many registers and some are dedicated to serving only customers with a small number of items. On the other hand, in restaurants everyone gets a little bit of service all of the time, which can be viewed as a form of Processor Sharing (PS). In fact, even deciding what order you will do the things on your to-do list can be thought of as a scheduling policy. Do you want to finish the most urgent tasks first, i.e. use Earliest Deadline First (EDF), or work on the tasks that you can finish the quickest, i.e. use Shortest Remaining Processing Time (SRPT)?

In addition to these everyday examples, scheduling policies are fundamental components of a variety of modern computer systems. Applications such as web servers, routers, operating systems, supercomputers, databases, and disks all face a constant barrage of requests and need to determine how best to allocate resources to users. In all these cases, queues of service requests build and a scheduling policy (discipline) is used to decide the order of service.

Wherever scheduling policies are used, they can have a dramatic impact on system "performance." In particular, at a high level, requests experience delay as a result of waiting in queues for service at a limited resource; thus how requests are scheduled at this resource is a fundamental determinant of delay. In fact, the delay experienced by requests can differ by orders of magnitude across scheduling policies.

This means that scheduling is especially important in computer systems, because users of computer systems are extremely demanding and unforgiving. In our daily lives, we are willing to accept some delay while we queue for service, but, in computer systems, users demand service that is both instantaneous and predictable. For example, web users become dissatisfied if response times for requests exceed 5 seconds and view delays of greater than 10 seconds as intolerable [65]. Further, meeting the high expectations of users in computer systems is crucial because it is often effortless for users to switch to a competitor's product. If we again consider the example of a web user, the competition is always "just a click away." Thus, a fundamental

design goal of computer systems today is to minimize the *response times* of users, i.e. the time between the moment a user makes a request and the moment the request is completed.

Scheduling is a fundamental tool for minimizing response times (reducing delays) and, as a result, the study of scheduling policies has a long history including a vast literature of analytic results. However, in recent years, the field has been going through a resurgence. This resurgence is a result of a variety of "scheduling success stories" in computer systems. In particular, at all levels of computer systems, designers have dramatically reduced user response times by making small changes to the scheduling policy used at the bottleneck resource. We will provide an overview of a few examples of success stories in Section 1.1, where we will see that the essence of these scheduling success stories is very simple (Section 1.2). In particular, in all the examples, system designers identify the bottleneck resource in a system, determine how it is scheduled, and then design a new scheduling policy in order to improve performance. This third step is really the defining aspect of the success story, and is the focus of this thesis.

In Section 1.3 we will begin to explore the issues involved in designing a new scheduling policy for a computer system. This task is not easy – there is an enormous variety of scheduling policies from which to choose. As a result, computer system designers are often guided by analytical results about scheduling policies, and we discuss what traditional analytical results about scheduling suggest for computer system design in Section 1.3.1. However, we will see in Sections 1.3.2 and 1.3.3 that there are many gaps between the analytical results and what happens in practice. Because of these disconnects between theory and practice, traditional analytic results do not apply for the policies that system designers use in practice. This fact is problematic because analytic results can be an important tool for system design.

The goal of this thesis is to develop a modernized theory of scheduling that can provide analytic results that apply to today's computer systems, easing the burden on system designers. To accomplish this goal, we will provide new results that help to bridge the disconnect between the analytical results and the needs of practitioners. We detail our approach for bridging these disconnects in Section 1.4 and then we conclude the introduction by providing an overview of the thesis in Section 1.5.

1.1 Scheduling success stories

Across applications, computer system designers have been suggesting new designs for the scheduling policies at the core of systems. This increasing focus has led to *scheduling success stories* at all levels of computer systems. In web servers [96, 182], routers, [179, 180], wireless networks [102, 136], peer-to-peer systems [178], operating systems [74], databases [138, 139], and beyond, researchers have made simple changes to the scheduling policies used at the bottleneck resources in computer systems that have led to dramatic improvements in user response times. Not only do these new designs result in improved response times, they do so without requiring the purchase of additional resources.

Example: Static Web Servers

The scheduling success story that is perhaps the easiest to explain is that of web servers. If we consider a web server that serves primarily static requests, its operation is very simple at a high level. Static requests are typically of the form "get me a file." In order to fulfill such a request, the web server must retrieve the file and then send the file over the outgoing link. Typically the amount of bandwidth at the web server is the bottleneck device since purchasing more bandwidth is much more expensive than upgrading the disks or CPUs at the web server [142, 59]. Even a modest web server

can saturate a T3 or 100Mbps Ethernet connection. Thus, much of the delay experienced by requests for files is a result of queueing for bandwidth.

*In standard web server designs, such as Apache [225] and Flash [168] servers, the bandwidth is allocated by cycling through the queued files, giving each a small slice of service. Specifically, each connection between a client and the web server has a corresponding socket buffer into which the web server writes the contents of the requested file. The sockets are then drained in a cyclic manner where a handful of packets from each socket are sent before moving to the next socket. This behavior is typically modeled using the Processor Sharing (*PS*) scheduling policy, which gives an equal share of the service capacity to each job in the queue at all times.*

Now comes the success story. Harchol-Balter et al. [96] have recently achieved dramatic reductions in user response times at static web servers by adjusting the scheduling policy used in web servers. They modified the way sockets are drained in order to implement a version of SRPT *and found that not only were response times much smaller [96], but also the performance in periods of overload was improved [203] and the response times of large files did not suffer as a result of the bias towards small files [25]. Further, following the initial work of Harchol-Balter et al., other researchers have gone on to design improvements to the scheduling policy, thus providing even more dramatic improvements over standard web server designers [182, 131, 130, 88]. We will talk in more detail about the actual policies in these designs later in the chapter.*

□

Example: Network Edge Routers

From a user-level perspective, a user is sending a sequence of packets, i.e. a flow, on a path through a number of routers in the network. Thus, a router must share resources between a number of competing flows. Typically, one of these routers is the bottleneck link along the path and contributes the majority of the network delay. Often times, this bottleneck router is the edge router, i.e. the router on the edge of the core of the network. Further, the bottleneck resource of the bottleneck router is the most frequently the outgoing bandwidth of the router [179], since it is much more expensive to overprovision bandwidth than it is to overprovision other resources. (Though, the complexity of scheduling policies used at routers is limited due to the fact that routers must make scheduling decisions very quickly so as to operate at line speed.) Thus, a key determinant of the delay experienced on a network path is how the bandwidth at the bottleneck router in the network is scheduled.

In standard router designs, variants of fair queueing are commonly used to allocate bandwidth to packets from competing flows at a router. These policies guarantee that the average bandwidth given to each flow through the router is approximately equal. This means that PS *and variants such as Generalized* PS *(*GPS*) and Discriminatory* PS *(*DPS*) serve as a good model of the scheduling policy at the router [250, 35, 36].*

Many techniques have been applied to try to reduce the delay in routers, including admission control, active queue management, and others. However, recently, Rai et al. [179] were able to dramatically reduce response times of flows by implementing a simple scheduling change in routers – they implement a policy that gives priority to flows with the least attained service, i.e. the flow that has had the fewest packets sent so far. With only this small change to the way flows are scheduled Rai et al. were able to reduce response times by an order of magnitude [179, 180]. We will provide more details about the policy they implement later in the chapter.

□

Example: Wireless Access Points

Wireless networks offer a number of interesting challenges when compared with traditional wired networks. One of the key challenges is that the wireless channel is a shared resource among all the users of a given access point. In order to achieve reliable transfers, users must reserve exclusive access to the shared channel. As a result of this and other concerns, wireless networks are severely bandwidth-limited, and the wireless link itself is the bottleneck resource. Thus, a key determinant of user response times is how the wireless channel is scheduled.

Allocation of the shared channel is performed in a centralized manner by the network access point. Typically, the access point polls clients to give them channel access grants, traditionally granting access in a round-robin manner so as to guarantee fairness among competing users. So, again, PS *is a good model of the scheduling policy being used at a standard wireless access point.*

But, there are many reasons why this allocation strategy is inefficient and many recent designs have dramatically reduced response times in wireless networks using simple changes to the scheduling policies used in wireless access points [102, 136]. These recent designs again apply variants of the SRPT *policy to schedule user requests. We will provide more details about the specific variants they implement later in the chapter.*

□

We could easily continue to list other recent scheduling success stories in applications such as operating systems [74], databases [138, 139], peer-to-peer systems [178], and beyond; however from these examples we can already see that changing the scheduling policy at the bottleneck device in computer systems can lead to dramatic performance gains at the system-level. Further, from these examples, we can already observe that there are many similarities between these success stories.

1.2 The essence of a scheduling success story

Though every system is different and each of the scheduling success stories we just described has its own nuances, the essence of these scheduling success stories is the same across systems. There is a consistent three-step design process that is followed.

1. The first step is to determine the bottleneck resource of the system. Identifying the bottleneck resource is one of the fundamental steps in system design. Knowing what resource is the bottleneck allows designers to focus on the part of the system responsible for the majority of delay in the system. This resource is exactly where scheduling changes will have the most dramatic effect on system-level performance. We saw that in static web servers the bottleneck is typically the limited bandwidth that the server has bought from its ISP [142, 59]. Similarly, in network routers and wireless access points bandwidth is again typically the bottleneck [179, 250]. In operating system scheduling, the bottleneck is typically the CPU [74], while in databases the bottleneck can either be the CPU or the database locks depending on the workload [138].

2. Once the bottleneck device is known, the next step is to understand how the bottleneck device is currently scheduled. This knowledge helps to determine what performance improvements are possible. It turns out that, across computer systems, the status-quo is typically to use a very simple scheduling

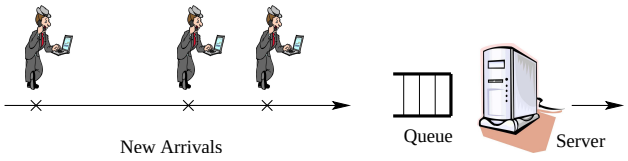


Figure 1.1: *An illustration of a single server queue.*

policy to schedule the bottleneck device, often either First Come First Served (FCFS) or Processor Sharing (PS). In particular, as we saw in our examples, it is most common that the bottleneck device is scheduled using a form of PS. This happens because most systems time-share, giving each request a small slice of service and cycling through the requests in a round robin fashion. For example, PS is a good approximation of the way web servers and routers (at a flow level) allocate bandwidth [96]. Additionally, operating systems tend to use variants of PS to schedule jobs at the CPU.

3. After the bottleneck has been identified and it has been determined how the bottleneck is being scheduled, the last step in a "scheduling success story" is to design and implement an improved scheduling discipline for the bottleneck resource. However, the details of the improved policy are very much application dependent. This step is the defining aspect of the scheduling success story. After the first two steps, system designers know which resource is the bottleneck and how it is being allocated, but the question is then, *what is the best, or at least an improved, design for a new scheduling policy?*

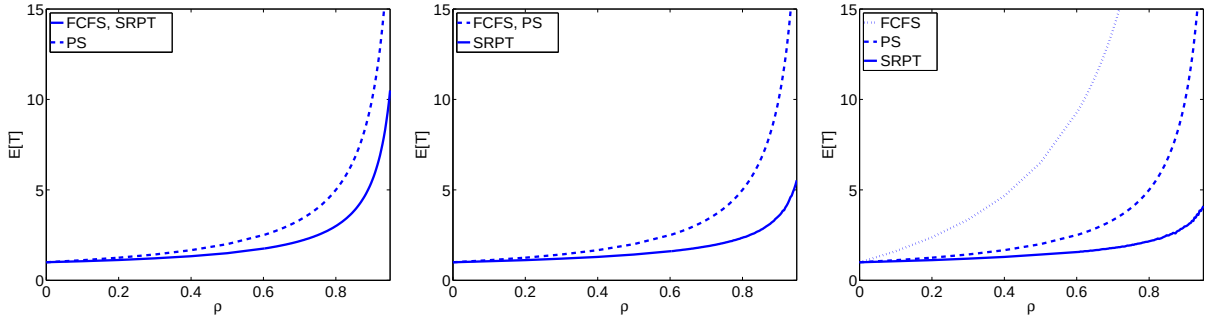
1.3 Choosing a scheduling policy

The defining aspect in recent scheduling success stories is the decision of which scheduling policy to implement at the bottleneck resource. This is not an easy decision – there are an infinite variety of possible scheduling disciplines to choose from. As a result, though it is possible to develop a new design through an ad hoc process of tuning and testing new proposals, performance modeling is often a key tool in developing a new scheduling discipline for the bottleneck device. In particular, by considering a single server queue (as illustrated in Figure 1.1) as a model of the bottleneck resource, designers can make use of a vast literature of analytic results about scheduling in order to better predict the performance of new design proposals. So, all a system designer needs to do after determining which scheduling discipline is used by the bottleneck device is to pick up one of the many books on scheduling [61, 119, 120, 176] and look for an improved policy.

Well, *at least it seems that easy.* In reality, there are many gaps between what the traditional analytic results provide and what system designers need in practice. These gaps mean that the results proven in theory do not end up applying to the systems that are built in practice.

1.3.1 What traditional theory says

The study of scheduling has a long history, including an extremely diverse set of application areas and models. The classical case of minimizing response times at a single server with a single queue is the case that is most relevant to our scheduling success stories. In any scheduling book, one of the first results that



(a) Deterministic (b) Exponential (c) Weibull

Figure 1.2: *Mean response time, $E[T]$, is shown as a function of load, ρ, under* SRPT, PS, *and* FCFS *in an M/GI/1 queue. The service distribution has mean 1 and is (a) Deterministic, (b) Exponential, and (c) Weibull with a variance of 10.*

is presented is that a simple, greedy policy is optimal in the single server model. In particular, Shortest Remaining Processing Time (SRPT) minimizes both the mean queue length and the mean response time regardless of the arrival sequence or job sizes [202, 201]. SRPT works by devoting the full service capacity to the job with the smallest remaining size. Thus, it greedily works on the job that can be completed the quickest.

However, simply knowing that SRPT minimizes the mean response time tells us nothing about how much improvement can be gained from using SRPT instead of other common policies like PS or FCFS. Therefore, we also need to understand the quantitative comparison of response times under SRPT, PS, and FCFS under practical workload assumptions.

To provide such a comparison, the traditional model that is used is a single server queue with an infinite buffer (see Figure 1.1) where the interarrival times of requests are assumed to be independent and identically distributed (i.i.d.) random variables, the single server works at a constant speed, and the service times (processing times, job sizes) of arrivals are assumed to be i.i.d. Moreover, the sequences of interarrival times and service times are assumed to be independent. This model is one of the most basic queueing models, and is referred to as the GI/GI/1 queue. The first GI indicates that the arrival process is a sequence of generally-distributed, independent random variables; the second GI indicates that the service requirements are generally-distributed, independent random variables; and the 1 indicates that there is 1 server. Most typically, scheduling policies are studied in the simpler M/GI/1 model, where the M stands for "Markovian" and indicates that the arrival process is Poisson, i.e. has exponentially distributed interarrival times.

A huge variety of scheduling policies have been studied in the M/GI/1 setting, and there are a number of excellent books that summarize the important results [61, 119, 120, 176]. We will survey many of the results in Chapter 3. However, let us now provide a simple comparison of SRPT, PS, and FCFS in order to illustrate the performance gains that are possible.

To provide this comparison we will use an M/GI/1 queue with three different service distributions. In Figure 1.2, we show a comparison between SRPT, PS, and FCFS under (a) a deterministic distribution where all jobs have size 1, (b) an exponential distribution, which is a common distribution because it is analytically tractable, and (c) a high-variance Weibull distribution. Figure 1.2 illustrates that SRPT provides an enormous improvement in mean response time in some settings; however the improvement depends very

much on properties of the workload. In particular, the improvement depends very strongly on the variability of service demands (job sizes) and the system load (utilization). Using SRPT instead of PS or FCFS provides only limited improvement in mean response time when job sizes are not very variable or if the system load is low. However, the improvement is dramatic if job sizes are highly variable or the system is moderate or highly loaded. The reason for this is simple. First, if the system is lightly loaded, then the queue lengths are small regardless of the scheduling policy used, so reordering jobs in the queue can only have a limited impact on response times. However, if the system is moderate or heavily loaded, queue lengths can be very large, and reordering jobs in the queue can have a big impact. Second, variability is important because when job sizes are highly variable, one large job can wreak havoc under FCFS. In particular, under FCFS, many small jobs can get stuck behind a large job in the queue. Further, one large job can have a negative effect on PS because a large job will stay in the system for a long time and, thus, limit the service capacity devoted to other jobs. In contrast, under SRPT, small jobs are unaffected by large jobs because they bypass larger jobs in the queue.

To summarize, the key observation from Figure 1.2 is that, SRPT can provide enormous improvements, but only under certain system loads and job size distributions. Thus, in order to understand whether SRPT will provide significant improvements for a given application, it is important to first understand the system load the application will experience and the distribution of service demands.

Let us start with the system load. Though it is not uncommon for designers to try to overprovision computer systems, the unpredictability and burstiness of computer system workloads means that it is common to experience extended periods of moderate-to-heavy load. For example, in web applications, surges in traffic as a result of special promotions, an abrupt increase in a site's popularity, or many other reasons, result in extended periods of high load. As a result, even well-provisioned systems spend a significant amount of time running in moderate or high-load, and maybe even in overload. These periods of high load are typically some of the most important times to provide users responsive service, and so scheduling policies for computer applications need to be designed with the high load periods in mind.

Next, let us consider the service demand distributions experienced by computer applications. Over the last decade, there has been an explosion of workload characterization research in the computer system community. This research has led to the growing realization that heavy-tailed and highly variable distributions, such as the Weibull and Pareto, are everywhere in computer system workloads [16, 62, 208, 172, 174]. Examples can be found in UNIX process lifetimes [91, 68], web file sizes [173, 62], and the number of embedded files in web sites [16, 28].

So, in designing scheduling policies for computer applications, the most important workload setting to consider is a highly loaded server with highly variable job sizes. This is exactly the setting where the performance improvements of SRPT are most dramatic, so, SRPT is the clear choice for use in computer systems according to traditional theoretic results.

1.3.2 What happens in practice

We just saw that theoretical results motivate the use of SRPT in computer systems – SRPT minimizes the mean response time and provides dramatic improvements over common policies like PS under practical workloads. Thus, it would seem that one should always use SRPT scheduling, regardless of the application that is being considered.

But, the picture painted by the theoretical results is a bit deceiving. In particular, the reality of computer

systems is far more complex than the simple models and idealized scheduling policies one finds in the books on scheduling.

To illustrate this briefly, notice the setting that was used in Figure 1.2 to analyze the performance improvements obtained by SRPT: Figure 1.2 shows the mean response time in an M/GI/1 queue using SRPT scheduling. Though we have seen that the assumption of highly variable Weibull service demands is fairly realistic, it is easy to find fault with each of the other assumptions. Real systems cannot implement pure SRPT; real arrival processes are not Poisson; real systems care about more than mean response time; and real systems do not always use a single server. To drive this point home, let us consider a few applications in detail.

Example: Static Web Servers

We saw earlier that the bandwidth purchased from the ISP is typically the bottleneck in web servers that serve primarily static content. Further, we saw that bandwidth is allocated to requests according to PS *scheduling. Additionally, the workload in web applications is typically highly variable and web applications must be designed with high load periods in mind. Thus, web servers seem to be a perfect place to use* SRPT.

SRPT *is indeed the motivation for a number of recent web server designs [96, 182, 131, 130, 87, 88]. However, many complications of the real systems prohibit these new designs from using pure* SRPT. *In particular, the proposed designs use the remaining sizes of the files being served in order to prioritize. However, these remaining sizes are only estimates of the remaining service demand of a request because the network delays, which are are not known exactly, also affect the service demands. As a result, many proposals do not use only the remaining sizes of the files, but also attempt to estimate the propagation delay to the users making the requests [182, 131, 130]. Another complication is the fact that implementations have tended to use only 5-10 priority levels instead of using a continuum possible priority levels (as pure* SRPT *does) as a result of the overheads associated with maintaining priorities and switching between jobs [96, 182]. As a result of these, and other, adjustments to* SRPT, *the policies that are actually implemented may perform significantly worse than pure* SRPT, *and the magnitude of the differences is not understood.*

Not only are there many reasons why pure SRPT *cannot be implemented in web servers, there are a number of reasons why designers do not want to use pure* SRPT. *A primary reason is that mean response time, although important, is not the only performance measure of interest. Designers also need to provide fairness and QoS guarantees. Further, it is often important to provide service differentiation between high priority customers (who have paid for improved service) and standard customers. As a result of these competing performance metrics, many design suggestions for web servers have used hybrids of* SRPT *and variants of* PS *[87, 88].*

In addition to adjustments to the policy that is implemented, there are many complexities of the practical workloads that are not accounted for by the traditional M/GI/1 queue. First of all, web users are interactive. When using a web site a user will click on a link and then wait for the response before clicking on the next link. Therefore, the arrival process is dependent on the departure process, unlike in the M/GI/1. Further, users are impatient, and will click on the refresh button if the response time for a certain page is too long. The effect of this is to abandon requests that are in the queue and have already received service, thus wasting bandwidth. This is an issue that is ignored by the M/GI/1 model.

□

Example: Network Edge Routers

We saw that the bandwidth is typically the bottleneck in edge routers and that the way a standard router allocates bandwidth can be viewed, at a flow level, as PS*. Like the case of web servers, routers are typically highly loaded and flow sizes tend to be highly variable. Thus, routers again seem like a perfect place to apply* SRPT *scheduling.*

However, it is impossible to apply SRPT *in routers because the sizes/lengths of flows (i.e. the number of packets in the flows) are unknown a priori. It is not even possible to estimate the lengths of flows accurately; all that is known about a flow is how much service it has received so far. But, it turns out that the amount of service received so far provides some indication of the remaining length of the flow. In particular, if a flow has received a large amount of service already, it is likely to require an even larger amount of service in order to complete [208]. Using this information, designers have proposed policies that give priority to the flows that have received the least service so far, e.g. variants of the Foreground-Background (*FB*) policy [179, 180] and the Multi-level Processor Sharing (*MLPS*) policy [6, 5]. However, such proposals have been shown to starve large flows, e.g. streaming videos, of service in addition to increasing jitter when compared with standard router designs [180]. Thus, hybrid designs combining aspects of* PS *with* FB *and* MLPS *have been proposed [180].*

Not only is it impossible to apply SRPT *in routers, it is obvious that the M/GI/1 queue is an overly simplistic model of router workloads. Like in web servers, users of routers are interactive and impatient. When using a web site a user will click on a link and then wait for the response before clicking on the next link. Further, if a request is delayed too long a user will abandon the request, e.g. by hitting the refresh button in her browser. The effect of these abandonments is actually quite dramatic: 20% of network traffic has been shown to correspond to aborted transfers [251]. Another important aspect of network traffic that is ignored in the M/GI/1 model is the fact that workload characteristics tend to be time-varying, e.g. time of day effects.*

□

Example: Wireless Access Points

We saw that the wireless link is the bottleneck resource in wireless networks and that PS *serves as a good model of the way a traditional access point typically allocates the wireless link bandwidth. Further, we described that recently suggested designs have been able to dramatically improve performance using variants of* SRPT *scheduling. But, for many reasons, these variants are far from pure* SRPT*.*

The main reason that pure SRPT *is not implemented is that the channel conditions are variable. In a multiuser wireless network, at any given point there are likely to be users that have "good" channel conditions, which allow data to be sent at a high rate, and users that have "bad" channel conditions, which limit data transfer speeds. If one ignores the channel conditions and uses pure* SRPT*, the throughput of the network as a whole will be much lower than if one is opportunistic when scheduling requests. As a result, the designs that are implemented tend to be hybrids of* SRPT *where both the channel conditions and the remaining size of requests are taken into account [102, 136]. However, note that the fact that channel conditions vary over time means that the remaining sizes used to schedule are not exact.*

In addition to variable channel conditions, wireless networks are subject to a number of other complexities that are not accounted for in traditional theoretical models. In particular, power management is a fundamental design constraint and needs to play a role in wireless link scheduling.

Further, multi-channel network designs are increasingly being used, and these networks are better modeled with a multiserver queue than with a single server queue. Finally, many of the complexities of network traffic that we have discussed for web servers and network routers also play a role in wireless networks, e.g. interactive and impatient users.
□

We could have easily continued to list other examples such as disks, databases, and supercomputing centers, however the above examples are enough to make the point that there are many aspects of real systems that differ from the traditional theoretical models. Further, the result of these differences is that pure SRPT, the "optimal" policy according to traditional theoretical results, is never used in practice. Thus, the theoretical results we just described do not immediately apply to real system designs.

1.3.3 Gaps between theory and practice

The previous two sections have illustrated a mismatch between the traditional theoretical research on scheduling and the use of scheduling in modern computer system designs. To summarize a few of the differences that we saw, notice that we repeatedly observed that real computer systems can never implement the pure, idealized SRPT policy that is optimal in theory. Two of the reasons for this are that (i) it is rare that real implementations know exact remaining sizes and (ii) real implementations must be adjusted to account for the overheads associated with preemption. Not only is it unrealistic to consider pure SRPT, it is unrealistic to assume a Poisson arrival process since, in reality, users are interactive: users typically must wait to receive one request before making another, thus the arrival process is dependent on the completion process. Similarly, it is increasingly unrealistic to consider only a single server queue – server farm and multi-core architectures are increasingly prevalent. Finally, considering mean response time as the only performance metric is also unrealistic. In real systems, mean response time is definitely important, but it is also important to be "fair" and to provide QoS guarantees (among a long list of other metrics).

This laundry list of differences is really only the tip of the iceberg. However, from this list and the applications we looked at in detail, three different themes are emerging. Though these three themes are not all inclusive, they cover a wide range of gaps between traditional theoretic results and the needs of system designers.

- **The idealized policies studied traditionally in theory cannot be used in practice.**
  For example, pure SRPT is never implemented in practice. Instead, the policies that are implemented use estimates of remaining size, use only 5-10 priority levels, or are hybrids of SRPT and PS-type policies. Each of these variants of SRPT will not provide response times that are as small as under pure SRPT, however traditional theoretic results do not provide any information about how much performance will suffer.

- **Many performance measures that are important in practice are not studied in theory.**
  Mean response time is typically the focus of theoretical scheduling research, however in practice QoS and fairness metrics are also important. Additionally, power management, reliability, and many other performance measures are important. Once these other measures are considered, SRPT is no longer the clear choice. Worries pervade that SRPT is unfair to large job sizes due to its bias towards small jobs pervade. Similarly, worries about providing good QoS guarantees for large job sizes are common. Traditional theoretical results cannot be used to address such worries.

- **The traditional, simplified theoretical models include many unrealistic assumptions.** The M/GI/1 model is at the heart of a majority of research studying the performance of scheduling policies, but both the M (Markovian arrivals) and the 1 (single server) are often unrealistic.[1] For example, real arrival processes tend to be bursty and real users tend to be interactive and impatient. Further, many modern system designs make use of multiserver architectures, e.g. server farms and multi-core processers. Though SRPT is optimal in the M/GI/1 setting, once one considers interactive, impatient users and multiserver settings, SRPT is no longer the optimal policy for mean response time. Further, the performance of SRPT in these more complex settings has not been studied in the traditional theoretical literature.

Each of these themes brings into question the usefulness of traditional theoretical results to modern computer system design. Traditional results suggest that SRPT is optimal and can provide dramatic improvements in response time, but these results apply only in simplified settings to pure SRPT. Once realistic settings and policies are considered, the performance improvements that come from using SRPT will be much less dramatic, and the exact degree of improvement is not understood. Further, results about how SRPT performs for other metrics of interest are simply not available; thus it is not easy to dismiss worries about, for example, providing QoS guarantees and fairness under SRPT. *The bottom line is that the traditional analytic results about scheduling provide limited help for system designers because of the may gaps between theoretical models and real system designs.*

1.4 Bridging the gaps between theory and practice

The fact that traditional analytic results do not apply to real system designs is a problem because analytic results can (and should) be invaluable to system designers during the development process. Without analytic results, designers need to test every every new design proposal using extensive experiments over a wide array of settings, which can be prohibitive during early stages of design. Further, even after such testing, designers are left without performance *guarantees* for the new designs.

However, analytic results could potentially provide provable guarantees to guide the design process. For example, if (as in the case of web servers) estimates of remaining sizes must be used instead of the true values, it is important to know how the accuracy of the estimates will affect response times under the policy. Understanding the impact of the accuracy of the estimates is key because there is often an overhead involved with making estimates and the amount of overhead is dependent on the accuracy needed for the estimates. A good example of this tradeoff is the task of estimating network delays for files being sent by a web server. This is one example of how analytic results can aid the design process, but there are many others. Analytic results are also important when deciding what level of QoS guarantees can be provided. Analytic results can help in determining the appropriate number of priority levels to use in order to balance between minimizing the overheads and maximizing performance. Analytic results are also fundamental to capacity planning of computer systems. We could list many other examples here, but the main point is simple: without analytic results, the task of a system designer is made much more difficult. Further, this difficulty is magnified by the enormous variety of possible scheduling policies from which to choose.

[1] Of course, one could argue that the GI (generally distributed, independent job sizes) is also unrealistic, but we feel this is less of an issue than either the M or the 1.

The goal of this thesis provide analytic results that apply to today's computer systems. Our results will bridge most (but not all) of the gaps between theory and practice that we have described so far. In particular, we will take steps towards resolving the issues in each of the three themes we described in the previous section.

- **Moving beyond idealized policies**
  We have seen that the idealized policies studied in theory are not used in practice and, instead, a wide variety of variants of these policies are used. Thus, traditional analytic results are inadequate for system designers. One natural approach to remedy this situation is to study each of the variants that are used in practice directly. However, using such an approach, it would be impossible to keep up with the new designs being developed across all levels of computer systems. So, we need a different approach.

  The approach we propose in this thesis is to move beyond the study of individual, idealized policies. Instead, we will formalize many common scheduling heuristics and techniques into classifications of scheduling policies, and then prove bounds on the performance of scheduling policies in each of these classifications. For example, SRPT is characterized by the fact that it uses the scheduling technique of "prioritizing based on remaining sizes" to apply the heuristic of "prioritizing small jobs." So, instead of studying SRPT, we will define and analyze a class of policies that prioritizes based on remaining sizes and a class of policies that prioritizes small jobs.

  This new style of scheduling research is motivated by the fact that, though the idealized policies studied in theory are not used in practice, the policies that are implemented in practice tend to be motivated by the theoretical policies. Thus, real system designs tend to apply the same heuristics and techniques found in the idealized policies. As a result, this new style of scheduling research has both practical and theoretical benefits. On the practical side, the scheduling classifications we define include, not only idealized policies like SRPT, but also the variants of these idealized policies that are actually used in practice. Thus, by providing results about scheduling classifications we are eliminating the need to analyze, one-by-one, each individual policy implemented in computer systems. On the theoretical side, analyzing scheduling techniques and heuristics adds structure to the space of scheduling policies that cannot be attained through the analysis of individual policies alone.

- **Moving beyond mean response time**
  Though mean response time is an important metric for computer systems, system designs must do more than simply provide small response times. We have seen that there are a wide variety of other performance measures that are also important. It is not enough if a new design provides improved mean response times, it must also guarantee fairness, provide QoS guarantees, and do many other things. But, unfortunately, the performance of scheduling policies with respect to many of these measures is not understood. In order to begin to bridge this gap, in this thesis we will focus on two measures of broad importance: the distribution of response time and fairness.

  Extending our understanding of scheduling policies beyond the mean response time to the distribution of response times is essential for applicability in modern computer systems because users can become even more frustrated by highly variable service than by having large response times on average [65, 255]. For example, reducing the jitter in streaming applications is at least as important as reducing the response time of the flow. Further, as we have discussed, it is increasingly important to

provide QoS guarantees, and knowledge about the response time distribution of scheduling policies is fundamental to this task. In this thesis, we not only provide many new results characterizing the response time distribution under scheduling policies; we also provide the first analytic results studying the response time distribution under scheduling classifications.

In addition, extending our discussion beyond mean response time to the fairness of scheduling policies is essential to the applicability of our results in real systems. One of the fundamental worries about designs that are motivated by SRPT is that large job sizes will have unfairly large and variable response times as a result of the priority given to small job sizes [30, 210, 215, 223]. Such worries are difficult to address because of the amorphous nature of "fairness," and thus there is no analytic work characterizing the fairness of scheduling policies. In this thesis, we introduce a variety of novel measures of fairness that are motivated by computer applications. In addition, we provide the first analysis of the fairness of scheduling policies. Not only that, we extend our analysis to handle classifications of scheduling policies as well.

- **Moving beyond the M/GI/1**
  Due to the difficulty of the analysis of scheduling policies, traditionally they have been analyzed only in fairly simple models, primarily the M/GI/1 queue. Though this model allows for general job sizes, as we have discussed, the assumptions of Poisson arrivals and a single server are often overly restrictive. We will move beyond these restrictive assumptions and study scheduling in settings where the arrivals are generated by interactive users and in settings where the system uses a multiserver architecture.

  One fundamental difference between arrivals to real systems and Poisson arrivals is that real users are interactive. That is, they must wait for their previous request to complete before submitting a new request. This interactivity introduces dependencies between the arrival and completion processes which is not present in the M/GI/1 model. We will characterize the impact of these dependencies on the performance of scheduling policies. We will illustrate that if the dependencies are strong enough, the effectiveness of scheduling can be limited, but that in many practical settings scheduling is still very beneficial.

  Another important difference between real systems and the M/GI/1 is that real systems are increasingly using multiserver architectures. The reason for this is that buying a single fast server is much more expensive than buying a large number of slower servers. The use of multiserver architectures has a huge effect on the impact of scheduling. For instance, in multiserver architectures SRPT is not optimal (in general) for mean response time. Further, while in a single server a single large job can block the server if small jobs are not given priority, in a multiserver system small jobs can bypass a single large job even without being given high priority. This intuition suggests that scheduling may not be as effective in multiserver settings. However, we will illustrate that clever scheduling in multiserver systems can still be very beneficial. Further, we will present results about how scheduling affects the design of multiserver systems, i.e. how scheduling impacts the number of servers that should be used.

1.5 An overview of the thesis

The thesis is organized into five parts. In Part I, we provide the motivation and background for the thesis. In Parts II-IV, we focus on bridging each of the three major disconnects we have described in this chapter. In Part II we focus on moving beyond individual policies to scheduling classifications; in Part III we move beyond mean response time and focus on a diverse set of other metrics; and in Part IV we focus on moving beyond the M/GI/1 model to understand the effect of more realistic assumptions on the performance of scheduling policies. Finally, in Part V, we will conclude the thesis by discussing the impact of the results in the thesis for system design and by discussing a number of future research directions motivated by the thesis.

When reading the thesis, the five parts need not be read in their entirety or in order. A reader familiar with stochastic scheduling can easily skim the majority of the remainder of Part I. Further, Parts II, III, and IV are largely independent of one another. The scheduling classifications introduced in Part II appear in Parts III and IV, however an informal understanding of the classifications should be all that is necessary when reading these parts. In addition, each of the chapters in Parts II, III, and IV can be read independently of one another.

1.5.1 Synopsis of Part I: Motivation and Background

In Part I of the thesis (which you are currently reading) we provide the motivation for the thesis (Chapter 1), as well as an overview of the analytic model at the heart of the thesis (Chapter 2) and the common, idealized policies studied in the literature (Chapter 3). Chapter 3 is especially important for the remainder of the thesis because it includes a survey of classical results and analytic techniques for studying common scheduling policies. Additionally, Chapter 3 includes a number of new results characterizing the mean response time of policies in heavy traffic.

1.5.2 Synopsis of Part II: Moving beyond idealized policies

After attaining the necessary background on the model and on classical results for common scheduling policies, we move to the heart of the thesis. In Part II, our focus is on bridging the gap between the policies studied in theory and the policies implemented in practice. For example, though many recent designs have been motivated by the optimality of SRPT for mean response time, none have implemented pure SRPT. Thus, the analytic results about SRPT do not apply to the resulting system designs. Perhaps the most straightforward approach for bridging this disconnect would be to model the details of the policies that are actually implemented in practice and then study these, more realistic policies, analytically. However, the wide variety of applications and, thus, policies used in practice, means that such an approach is unmanageable. So, instead we will develop a new approach: we define scheduling classifications that formalize the scheduling heuristics and techniques applied by system designers. For example, instead of studying pure SRPT, we will study classifications that formalize the heuristic of "prioritizing small jobs" and the technique of "prioritizing based on remaining size." By studying these scheduling classifications we can attain results for the policies that are used in a wide variety of different applications at once and eliminate the need to analyze these policies one-by-one. For example, though SRPT is not implemented in practice, many variations of policies that "prioritize small jobs" are implemented.

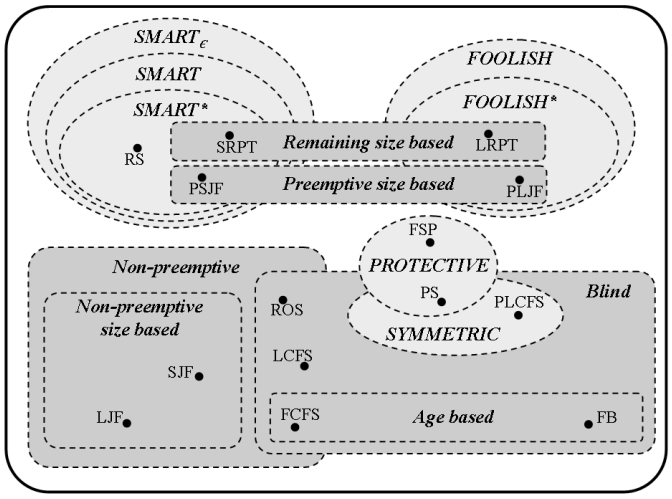


Figure 1.3: *An illustration of the scheduling classifications studied in this thesis. The heuristic-based classifications introduced in Chapter 4 are shown in ovals and the technique-based classifications introduced in Chapter 5 are shown in rectangles. An overview of the acronyms for scheduling policies in this figure can be found in Table 2.1.*

1.5.2.1 A wide variety of classifications

We will study scheduling classifications that formalize a wide variety of scheduling heuristics and techniques. The heuristic of "prioritizing small job sizes" is perhaps the most common scheduling heuristic, but many other heuristics are also used. We introduce and study classifications of scheduling policies that formalize four common scheduling heuristics:

(i) The SMART classification formalizes the heuristic of prioritizing small jobs
(ii) The FOOLISH classification formalizes the heuristic of prioritizing large jobs
(iii) The SYMMETRIC classification is a broad generalization of the classical PS policy
(iv) The PROTECTIVE class formalizes a notion of fairness.

Further, there are a wide variety of scheduling techniques that are used. We will formalize four technique-based classifications:

(i) Preemptive size based policies
(ii) Remaining size based policies
(iii) Age based policies
(iv) Non-preemptive policies

These eight classifications are illustrated in Figure 1.3, which also includes a number of variants of these classifications that we will discuss in Part II. These classifications cover a wide variety of scheduling heuristics and techniques that have been applied across a range of applications including web servers, routers, disks, operating systems, and others. The SMART class was introduced by Wierman et al. [241];

the FOOLISH class is novel to this thesis; the SYMMETRIC class was introduced by Kelly [113]; and the PROTECTIVE class was introduced by Henderson and Friedman [78]. Further, the classes of remaining size based policies, preemptive size based policies, and age based policies were introduced by Wierman et al. in [238, 240], while the class of non-preemptive policies has been studied often in the literature, see for example [119, 120, 247].

In Part II, we will focus on proving bounds on the response times under each of these classifications. For example, we will prove that all SMART policies have mean response time within a factor of 2 of optimal (SRPT) in the M/GI/1 model. These classifications will then serve as building blocks throughout the remainder of the thesis, and we will return to them in Part III to discuss their performance with respect to other performance metrics and in Part IV to discuss their performance in more general models than the M/GI/1. For example, in Part III, we will prove that all SMART policies have a response time distribution that is asymptotically equivalent to that of SRPT in the GI/GI/1 model.

1.5.2.2 The benefits of studying scheduling classifications

By studying these scheduling classifications instead of studying individual, idealized policies, we attain many important benefits, both of theoretical and practical interest.

From a theoretical point of view, the analysis of classifications exposes the performance impact of scheduling techniques and heuristics, providing a deeper understanding of scheduling than the analysis of only idealized policies. For example, we will prove that no SMART policy can be fair under all workloads, which provides an interesting impossibility result: in order to provide near optimal mean response times (by giving priority to small jobs) a policy must sacrifice fairness.

From a practical point of view, the analysis of classifications provides analytic results for the policies that are actually implemented in real system designs. In particular, though real designs cannot implement the idealized policies studied in theory, the idealized policies serve as motivation for the policies the designs used in practice. Thus, the practical designs are based on the same scheduling heuristics and techniques as the idealized policies. So, by defining classifications that formalize these heuristics, practical system designs can be included within the classifications; and thus results proven about the classifications can apply to the designs used in practice.

Further, these classifications themselves can provide results that aid in the design process. For example, we will define a class of policies called SMART$_\epsilon$ that includes all policies that "prioritize small jobs" using inexact job size information. The performance bounds we prove for the SMART$_\epsilon$ class will be in terms of the accuracy of the job size estimates. Thus, we will illustrate the tradeoff between the accuracy of estimates used to prioritize and the response times that result. As we saw in our web server example, this tradeoff can be a key design criteria.

1.5.3 Synopsis of Part III: Moving beyond mean response time

Throughout Part II of the thesis, we focus almost entirely on mean response time. This focus allows us to move beyond the study of idealized scheduling policies to study scheduling classifications, and thus provide result for the policies that are actually used in practice. Our focus on mean response time is not unusual: traditionally, in the stochastic scheduling community mean response time is the primary metric of interest. However, it is not enough for system designs to provide improved mean response times, there are a wide variety of other important metrics as well. For example, fairness and QoS guarantees are often important, as

is minimizing power and maximizing availability.

In Part III of the thesis we will move beyond mean response time and consider a wide variety of other metrics that are important across computer systems. Clearly, we cannot hope to provide results for every important metric, so we will focus on two measures that have broad applicability: the distribution of response time and "fairness." Our goal in studying these measures is not only to provide results for traditional, idealized policies, but also to provide results for the scheduling classifications introduced in Part II.

1.5.3.1 The distribution of response time

Extending our discussion beyond mean response time to the distribution of response time is essential for applicability in modern computer systems. Computer users demand not only response times that are fast on average, but also response times that are predictable. In fact, they become even more frustrated by highly variable service times than by having large response times on average. Another motivation for studying the distribution of response times is that QoS guarantees are increasingly important in computer systems, and providing QoS guarantees requires an understanding of the distribution of response times.

Unfortunately, understanding the distribution of response time is known to be a difficult task. In particular, exact derivations of the response time distribution are only possible in very specialized settings such as the M/M/1 and only under very simple scheduling policies, such as FCFS. Due to the difficulty of exact analysis, a common approach to studying the distribution of response times is to study an asymptotic scaling of the distribution. In particular, the most common scaling is to study the tail behavior of the distribution of response time, T, i.e. $Pr(T > x)$ as $x \to \infty$. This is a natural scaling to consider because it provides bounds on the likelihood of large delays, which are exactly what is necessary in QoS and capacity planning applications. Further, this scaling is useful for system design because it often provides an understanding of the most likely way for a large delay to occur, i.e. the "critical event" in large deviations parlance. This knowledge can then be used to limit the occurrence of such events.

Our goal in Chapter 6 will be to characterize the response time distribution under both common individual policies and scheduling classifications. The study of the response time distribution is not new to this thesis, so there are is already a large literature analyzing the tail behavior of response time under many common policies. We will extend this literature by generalizing the results for some common policies from the M/GI/1 queue to the GI/GI/1 queue, e.g. FB. But, our primary goal is to analyze the tail behavior under scheduling classifications, and to this end we will provide results for the SMART, FOOLISH, PROTECTIVE, and non-preemptive classifications. These results provide a number of interesting contrasts. For instance, we will show that SMART policies provide an asymptotically optimal response time tail when the service distribution is heavy-tailed, but provide a response time tail that is as heavy as possible under light-tailed service distributions. In contrast, FOOLISH policies have response time tails that are as heavy as possible, i.e. $Pr(T > x)$ decays as slowly as possible, under both light-tailed and heavy-tailed service distributions. Similarly, non-preemptive policies have as heavy a response time tail as possible under heavy-tailed service distributions, but can have an asymptotically optimal response time tail under light-tailed service distributions.

These results have an immediate impact for system design. In particular, they highlight the need for understanding the tail behavior of job sizes before making design decisions about which scheduling policy to use. Further, the derivations of the results provide insight into the causes of large delays under different policies. For example, under non-preemptive policies, the analysis formalizes the idea that when a tagged job experiences a long delay, it is likely due to a large job being at the server when the tagged job arrives. In

contrast, under SMART policies, the analysis illustrates that when a tagged job experiences a long delay it is likely the result of a burst of arrivals (having smaller sizes) arriving just after the arrival of the tagged job.

1.5.3.2 Fairness

Extending our discussion beyond mean response time to consider fairness metrics is also important for the applicability of our results in practice. Fairness metrics are important in any system where there are human users. Although typically not the primary metric of interest, it is important that low priority users are not starved of service. This is a particular concern for designs that try to improve response times by giving priority to small jobs at the expense of large jobs. Large jobs are typically some of the more important tasks, for instance the large jobs at e-commerce sites are the shopping cart transactions, and thus it is important that these jobs are not starved of service. In fact, worries about fairness to large jobs have plagued designs suggesting the use of SRPT-like policies in web servers and wireless networks.

Though it is clear that fairness is important, fairness is an amorphous concept and thus difficult to define. Further, fairness can take on many different meanings depending on the application being considered. As a result, there is almost no work studying the fairness of scheduling policies.

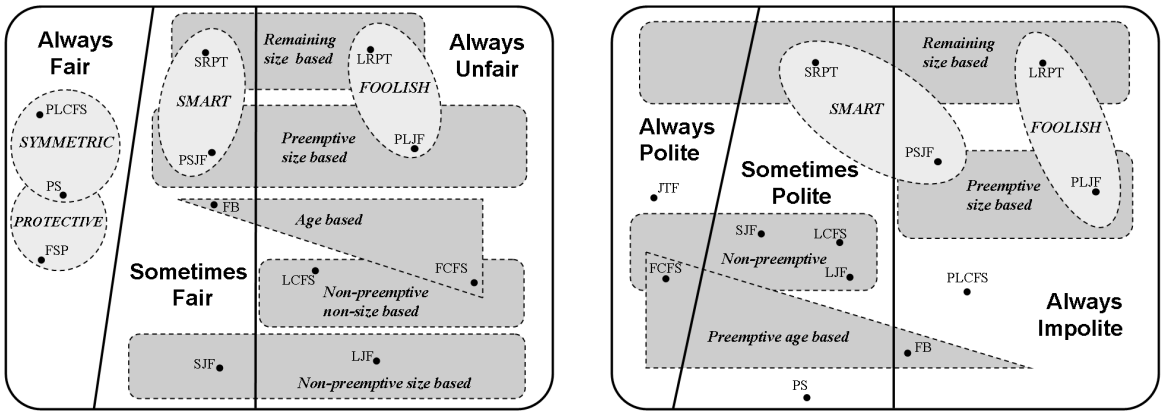
In Chapter 7, we develop a two notions of fairness, each motivated by different application requirements. First, we develop a framework for studying *proportional fairness* and then we develop a framework for studying *temporal fairness*. Proportional fairness refers to the idea that all job sizes should receive equitable service, i.e. no job size should have disproportionately large response times. Temporal fairness refers to the idea that it is fair (polite) to respect the seniority of jobs, i.e. it is in some sense unfair for a job that just arrived to jump in front of a job waiting in the queue.

When studying both proportional and temporal fairness our approach is similar. We will start by providing both intuitive and mathematical motivation for our proposed frameworks. Then, we will investigate how common, individual policies perform. And, finally, we will build on the results for individual policies and study the fairness of scheduling classifications.

A brief overview of some of the results that we will obtain is shown in Figure 1.4. This figure illustrates a wide array of interesting results. For example, notice that SRPT is Sometimes Fair and Sometimes Polite. This means that under some loads and service distributions, SRPT provides both proportional fairness and temporal fairness while optimizing mean response time. These results are counter to the intuitive worries that SRPT will be unfair to large job sizes as a result of its bias towards small jobs. Further, SRPT is not alone in this behavior. Figure 1.4 illustrates that all policies that prioritize towards small job sizes (i.e. SMART policies) have similar behavior. Further, this fairness behavior is superior to that of many other common scheduling heuristics and techniques. Thus, these results indicate that the conflict between providing small mean response times and being fair is not as severe as many people have worried.

1.5.4 Synopsis of Part IV: Moving beyond the M/GI/1

Throughout Parts II and III, the model underlying our analysis is primarily the M/GI/1 queue. Limiting our focus to this model allows us to consider general classifications of scheduling policies and a broad range of performance metrics; however, as we have discussed, there are many ways in which the M/GI/1 model is an unrealistic model for computer systems. While the assumption of general, independent job sizes is fairly broad, both the assumption that the arrival process is Poisson and the assumption of a single server can be unrealistic in many settings. Thus, it is important that we understand how the effectiveness of



(a) Proportional fairness (b) Politeness

Figure 1.4: *An illustration of the classifications of common prioritization techniques and heuristics with respect to proportional and temporal fairness (politeness). Much more detail on these classifications will be provided in Chapter 7.*

scheduling changes under generalizations of this model. Providing this understanding is the focus of Part IV. Specifically, we will generalize both the M and the 1. We will consider a model where the arrival process is governed by interactive users and we will consider multiserver architectures.

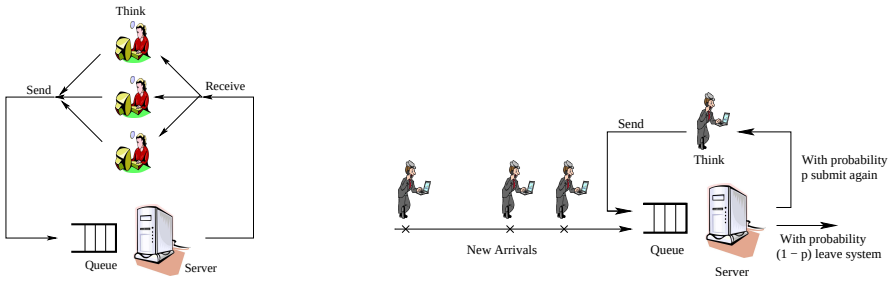
1.5.4.1 Interactive users

In real computer systems, the arrival process is far from Poisson. One fundamental difference we have discussed is that users of computer systems are interactive: they must wait for their previous request to complete before submitting a new request. This interactive behavior introduces a dependency between job completions and job arrivals that is missing entirely from the M/GI/1 model.

Intuitively, these dependencies will lessen the effectiveness of scheduling. In the M/GI/1 model, scheduling is the only way to avoid having large queues build when a large job is at the server; however in systems with interactive users, users cannot submit a new request until their previous request has completed, so there is an inherent limit on how quickly queues can build. As a result, one might make the intuitively appealing claim that scheduling is not beneficial when users are interactive.

In Chapter 8, we will investigate the validity of this claim using the closed and partly-open system models (see Figure 1.5 for an illustration of these models). We will find that, it is true that in *some* settings scheduling is ineffective. When systems have a small population of simultaneous users and they tend to have long interactive sessions (> 10 requests per session), scheduling does not lead to significant performance gains. However, *we will show that such settings are the exception rather than the rule.* We investigate a number of workloads from real systems and find that in a majority of the workloads, users have short to moderate length sessions (1-7 requests per session), and thus scheduling can still provide significant performance gains.

These results provide an important cautionary tale for system designers: it is important to understand user behavior both when designing new systems and when evaluating new systems. When designing a



(b) Closed system (c) Partly-open system

Figure 1.5: *Illustrations of the closed and partly-open system models.*

system, scheduling is an effective design choice in settings where users interact with systems for moderate length sessions, but if user session lengths are long, designers should use other techniques. In addition, when evaluating new systems, designers need to be careful to choose a workload generator based on a system model with the appropriate user model.

1.5.4.2 Multiserver architectures

There is a growing trend in computer systems to use multiple, slower, cheaper servers instead of a single, faster, more expensive server. For instance, in web servers, server farm architectures are increasingly prevalent and in wireless networks multi-channel designs are becoming commonplace. Further, multi-core processors are beginning to reach home users. Given the growing adoption of multiserver architectures, it is important to understand how the benefits of scheduling in the M/GI/1 model translate to multiserver systems. To this end, we will investigate the effectiveness of scheduling in the M/GI/k.

Intuitively, scheduling will be less effective in multiserver settings than in single server settings. In single server settings, scheduling is the only mechanism for preventing many small jobs from queueing behind a single large job. However, in multiserver systems this is no longer the case – if a large job is queued at one server smaller jobs may bypass this job by using other servers. As a result, one might claim that using multiserver designs is a replacement for using scheduling-based designs. We will show that this is not necessarily the case, i.e. that scheduling still can provide significant performance benefits in multiserver settings, even if these performance gains are not as dramatic as in the M/GI/1 setting.

Our goal in Chapter 9 is to provide analytic results relating the effectiveness of scheduling in multiserver architectures to that in single server settings. However, the analysis of multiserver systems is a notoriously difficult problem in queueing theory. Outside of FCFS scheduling little is understood analytically, and even when studying FCFS researchers typically result to approximate techniques. Thus, we will not be able to provide results in the same generality as we do for the rest of the thesis. Instead, our results in this chapter will focus on one important aspect of scheduling: prioritization. In particular, we will focus on systems where there are a finite number of priority levels and jobs are served preemptively according to their priority. Such schemes are commonly used in practice in order to provide service differentiations, e.g. certain customers pay more in order to receive high-priority service at some high demand resource. Further, understanding priority queues is the first step towards understanding more complicated priority based policies such as SRPT.

In Chapter 9, we will develop a new analytical approach that provides the first near-exact analysis of priority queues in multiserver settings. Our new analytic technique allows us to obtain many insights about the effectiveness of prioritization in multiserver settings. For example, we will contrast the performance of systems with k servers each having speed $1/k$ with the performance of systems with a single server of speed one. Further, among a long list of other topics, we will study the effectiveness of prioritizing small jobs in a multiserver environment and how best to design a multiserver system given a fixed total service rate, i.e. we will determine "how many servers are best?"

1.5.5 Synopsis of Part V: Further discussion and conclusion

Finally, we will conclude the thesis in Part V. Our goal in this part is to summarize the contributions of the thesis and discuss the implications of the results in the thesis for both computer system designers and scheduling researchers. To this end, we will review a number of specific examples of the impact the results in the thesis have for system design and theoretical scheduling research. Then, we will end the thesis by highlighting a number specific future research directions that are motivated by the results in the thesis.

CHAPTER 2

The basic model of the thesis

The study of scheduling has a long history, including an extremely diverse set of application areas from manufacturing, to computer systems, to call centers, to airport gate and flight scheduling, and beyond. The wide ranging applications of scheduling has led to an enormous analytic literature studying a wide variety of models and performance metrics. Models range from simple single server setups where the job sizes are independent of the order they are served and the arrival times and job sizes are known in advance, to complex job shop models where jobs must receive service in a specific order from different stations in the system, have precedence constraints governing the order in which they can be served at each station, and have sizes that depend on the order they are served. In addition to the wide range of models that have been studied, there are a metrics that are used to quantify the performance of different scheduling policies. In some cases minimizing response times and queue lengths is important, while in others minimizing the makespan (the time until the last job completes) is the goal. In others, jobs have due dates that they must complete before.

As a result of the wide range of models and performance measures that are of interest, a wide range of analytic techniques have been used to analyze these scheduling policies. A large community of researchers approaches scheduling from a deterministic perspective, where there are a finite number of arrivals to be scheduled and the goal is to understand the worst case performance of scheduling disciplines. In the deterministic scheduling world, no assumptions are made about the job sizes or arrival sequence, and thus results characterize the worst case performance of scheduling policies. The goal is to prove results such as: the mean response time of policy P is never more than $O(\log n)$ away from optimal, where n is the number of jobs that were scheduled. In contrast, a separate community of researchers approaches scheduling using stochastic techniques. Here, distributional assumptions are made about the job sizes and arrival sequence and then the performance of policies is analyzed under these assumptions. The goal here is to prove results that provide formulas for the mean response time as a function of the distributional assumptions about the arrival process and the job sizes.

In this thesis, we will use primarily stochastic techniques. The reason for this choice is that our goal is to understand how the policies used in real implementations perform in practice. Thus, we are less interested in providing worst case guarantees and more interested in understanding the performance of scheduling policies under realistic workloads.

However, even within the stochastic scheduling community there is a huge variety of "standard" modeling assumptions and techniques that are adopted by different subfields. While queueing theory researchers

can often provide beautiful and simple formulas to analyze scheduling policies under simple model assumptions, for more complex models obtaining such formulas is often impossible. As a result, researchers often either apply computational techniques (e.g. matrix analytic methods) or study asymptotic scalings of the systems. Further, there are a wide variety of asymptotic scalings of the systems that researchers have considered, from fluid and diffusion scalings to large buffer and many-sources large deviations scalings.

With this cacophony of possible models, it is required that we begin by spending some time introducing the model, notation, and probability distributions that we will use throughout the thesis. That is the purpose of the remainder of this chapter. We will start by providing an overview of the model we consider (Section 2.1). Then, we will summarize the key performance metrics that we will study (Section 2.2). Next, we provide a summary of the notation that is most commonly used in the thesis (Section 2.3). Finally, we provide an overview of the probability distributions that appear frequently in the thesis (Section 2.4).

2.1 An overview of the model

In this thesis, we are interested in evaluating the performance impact of scheduling changes at the (single) bottleneck resource in computer systems. Thus, we consider primarily single server queues, though we will also extend our discussion to multiserver queues in Chapter 9. We apply primarily *stochastic techniques* to analyze scheduling policies in these queues. In particular, we will be considering single and multiserver queues where the interarrival times and service demands (a.k.a. processing times or job sizes) of jobs/customers are assumed to be independent and identically distributed (i.i.d.) random variables. Moreover the sequences of interarrival times and service demands are assumed to be independent of one another.

To specify the distributional assumptions made about the interarrival and service demands, we make use of Kendall's notation [115]. A queue is referred to by an expression of the form $A/B/k\ C$ where A and B stand for the distributions of the interarrival times and service demands respectively, k stands for the number of servers in the system, and C denotes the scheduling policy that governs the queue. We will primarily be considering single server queues, so k will typically be 1. Further, A will either take the shape of M, which stands for memoryless (exponential) interarrival times (i.e. a Poisson arrival process) or GI, which denotes general i.i.d. interarrival times. Similarly, B will primarily take the shape of M or GI. However, we will also consider deterministic service times, which we denote D, phase-type service, which we denote PH, and a variety of heavy-tailed distributions such as the Weibull and Pareto. Background and notation for each of these distributions can be found in Section 2.4.

Since our focus in this thesis is on understanding the effects of scheduling, C will take a wide variety of forms. Table 2.1 provides a brief overview of some of the most common scheduling disciplines, however it is far from a complete list of the policies we will discuss in the thesis. We will provide much more background about these policies in Chapter 3, where we will survey known performance results and analytic techniques. However, for now, let us just note that Table 2.1 includes both scheduling policies that are *blind*, i.e. that do not use job size information to schedule, and policies that prioritize based on job size information. Further, Table 2.1 includes both policies that are *preemptive*, i.e. allow the job at the server to be interrupted and restarted later, and policies that are *non-preemptive*, i.e. never interrupt a job that has begun service.

Two important assumptions we make about the policies we consider are the following. We assume the *preempt-resume* model, thus we allow any preempted job to be resumed from where it was left off without penalty. Further, the scheduling policies we consider will almost exclusively be *work conserving*, which

Policy	Description
FB	Foreground-Background preemptively serves those jobs that have received the least amount of service so far. For details, see Section 3.2.5.
FCFS	First Come First Served serves jobs in the order they arrive. For details, see Section 3.1.1.
FSP	Fair Sojourn Protocol preemptively serves the job in the system that would have the smallest remaining size if the system were using PS. For details, see Section 4.5.
LCFS	Last Come First Served non-preemptively serves the job that arrived the most recently. For details, see section 3.1.3.
LJF	Longest Job First non-preemptively serves the job in the system with the largest original size. For details, see Section 3.2.6.3.
LRPT	Longest Remaining Processing Time preemptively serves the job in the system with the largest remaining processing time. For details, see Section 3.2.6.1.
PLCFS	Preemptive Last Come First Served preemptively serves the most recent arrival. For details, see Section 3.1.2.
PLJF	Preemptive Longest Job First preemptively serves the job in the system with the largest original size. For details, see Section 3.2.6.2.
PS	Processor Sharing serves all customers simultaneously, at the same rate. For details, see Section 3.1.4.
PSJF	Preemptive Shortest Job First preemptively serves the job in the system with the smallest original size. For details, see Section 3.2.3.
ROS	Random Order of Server non-preemptively chooses a random job from the queue to serve. For details, see Section 3.1.3.
RS	RS preemptively serves the job in the system with the smallest product of remaining size and original size. For details, see Section 5.2.
SJF	Shortest Job First non-preemptively serves the job in the system with the smallest original size. For details, see Section 3.2.2.
SRPT	Shortest Remaining Processing Time preemptively serves the job with the shortest remaining service requirement. For details, see Section 3.2.4.

Table 2.1: *A brief introduction to some of the most common scheduling disciplines in the thesis. This list is far from complete, but provides an indication of the variety of policies we will study.*

means that the server is always serving with its full capacity whenever there is at least one job in the system. In other words, the server never idles while there is work in the queue.

2.2 Performance metrics of interest

We will study a wide range of performance metrics in the thesis, but the primary metric of interest will be the *response time* of a customer, which is defined as the time between when a job arrives and when the job completes service. Response times are of primary importance across computer applications because users of computer systems are extremely demanding and unforgiving. While in our daily lives we are usually willing to accept some delay while we queue for service, in computer systems users demand service that is both instantaneous and predictable. For instance, if an online shopping site is loading very slowly (i.e. has large response times for pages), customers will become frustrated and take their business elsewhere since the competition is always "just a click away." Similarly, providing small response times is fundamental to routers, operating systems, disks, etc. User studies have consistently found that web users become dissatisfied if response times for requests exceed 5 seconds and view delays of greater than 10 seconds as intolerable [65].

We will denote the response time of a job under policy P by T^{P}. Note that response time is referred to by a variety of names in different communities. In particular, the terms *sojourn time* and *flow time* are sometimes used instead of response time. When studying the response time under policy P, we will often be content to derive the mean response time, $E[T]^{\mathsf{P}}$. However, in some cases, we will also study higher moments of response time $E[T^i]^{\mathsf{P}}$, as well as the distribution of response time $P(T^P > x)$.

In addition to T^{P}, it will many times be of interest to consider the conditional response time experienced by a job of size x under policy P, which we denote by $T(x)^{\mathsf{P}}$and refer to as the "conditional response time." One can think of $T(x)^{\mathsf{P}}$as the response time experienced by a tagged job of size x that arrives to a stationary queue. This is quantity is often used as a first step in deriving T^{P}, i.e., often in order to derive T^{P}it is necessary to first condition on the size of the job that is arriving. However, beyond its use as a stepping stone, $T(x)^{\mathsf{P}}$will be of fundamental importance when we study the fairness of scheduling policies in Chapter 7.

Beyond response time, we will also also be interested in a number of other metrics that we will introduce later in the thesis. But, one other metric worth mentioning here is the *slowdown* or *stretch*. The slowdown of a job is a weighted response time measure defined as the response time of the job divided by the size of the job. Thus, the slowdown experienced by a job of size x, denoted $S(x)$, is equal to $T(x)/x$. Again, we will consider both the overall slowdown, denoted S^{P}, and the conditional slowdown, $S(x)^{\mathsf{P}}$.

In order to analyze the response time and slowdown of scheduling policies, it will typically be useful to decompose the response time into two pieces: (i) the *waiting time* and the (ii) *residence time*. The waiting time of a job under policy P, denoted W^{P}, is the time between when the job arrives and when the job first receives service. The residence time of a job under policy P, denoted R^{P}, is the time between when the job first receives service and when the job completes service. Notice that the residence time may include time that the job is not receiving service, e.g. under SRPT a large job may begin to receive service but then be interrupted by the arrival of a small job. Similarly to $T(x)^{\mathsf{P}}$, the conditional waiting time, $W(x)^{\mathsf{P}}$and the conditional residence time, $R(x)^{\mathsf{P}}$will also be of interest.

2.3 Commonly used notation

Our discussion so far has already resulted in a lot of notation, and we have only skimmed the surface. Further complicating things is the fact that many different researchers use very different "standard" notation

for the same quantities. For the remainder of this chapter, we will try to summarize the notation that is used commonly in this thesis. Our hope is that by summarizing all this information in one place, there will be a single point of reference for the reader in the event of notation overload.

2.3.1 Basic mathematical notation

Let us first introduce our notation for some common mathematical functions:

$E[Y]$	the expectation of Y
$Var[Y]$	the variance of Y
$\mathcal{L}_Y(s) = E[e^{-sY}]$	the Laplace transform of Y
$\mathcal{M}_Y(s) = E[e^{sY}]$	the moment generating function of Y
$\mathcal{Z}_Y(z) = E[z^Y]$	the z-transform of Y

Table 2.2: *A brief overview of some of the most notation used in the thesis.*

Next, let us introduce some asymptotic notation that will show up frequently in the thesis. We use the notation $f(x) \sim g(x)$ to indicate that $\lim_{x\to\infty} f(x)/g(x) = 1$. Further, we use the notation $f(x) = O(g(x))$ as $x \to a$ to indicate that $\limsup_{x\to a} f(x)/g(x) < \infty$. Similarly we use the notation $f(x) = \Omega(g(x))$ as $x \to a$ to indicate that $\liminf_{x\to a} f(x)/g(x) > 0$. If $f(x) = O(g(x))$ and $f(x) = \Omega(g(x))$ we say that $f(x) = \Theta(g(x))$. Finally, we say that $f(x) = o(g(x))$ if $\lim_{x\to\infty} f(x)/g(x) = 0$.

In addition to these standard quantities, there are a few less commonly used quantities that will appear frequently. We will often use the squared coefficient of variation to quantify the variability of a distribution

$$C^2[Y] \quad = \quad \frac{Var[Y]}{E[Y]^2}, \text{ the squared coefficient of variation of } Y$$

The squared coefficient of variability is a normalized measure of variability under which $C^2[Y] = 1$ when Y is an exponential random variable. Thus, all distributions with $C^2 < 1$ are less variable than the exponential and all distributions with $C^2 > 1$ are more variable than the exponential.

Another less common distributional statistic that we will use are the cumulant moments. Cumulants have appeared only sporadically in queueing [71, 85, 137], tending to be used in large deviation limits. Cumulants are a descriptive statistic similar to moments. Formally, the cumulant moments of a random variable X, $\kappa_i[X]$ $i = 1, 2, \ldots$, are defined in terms of the moments of X, $E[X^i]$, as follows:

$$e^{\kappa_1[X]t + \frac{\kappa_2[X]t^2}{2!} + \cdots} = 1 + E[X]t + \frac{E[X^2]t^2}{2!} + \ldots \tag{2.1}$$

From this definition it follows that the cumulants of X can be generated from the cumulant generating function,

$$\mathcal{K}_X(s) = \log(\mathcal{L}_X(s)).$$

That is, $(-1)^i \mathcal{K}_X^{(i)}(0) = \kappa_i[X]$. Further, it follows that the cumulants of any distribution can be found from

the raw moments, μ_n, as follows

$$\kappa_n = \mu_n - \sum_{j=1}^{n-1} \binom{n-1}{j} \mu_j \kappa_{n-j} \tag{2.2}$$

where $\kappa_1 = \mu_1$ [116]. Immediately from (2.2), we can see that the cumulants capture many of the standard descriptive statistics. The first cumulant is the mean; the second cumulant is the variance; the third cumulant is the third central moment and thus measures the skewness of the distribution; and the fourth cumulant measures the kurtosis of the distribution. See [116] for tables of the relationships between higher order cumulants, raw moments, and central moments.

Although not immediately evident from the definition, cumulants have many properties that both raw and central moments lack. For instance, letting c be a constant, $\kappa_1[X + c] = \kappa_1[X] + c$ but for $i \geq 2$, $\kappa_i[X + c] = \kappa_i[X]$. Thus, the first cumulant is shift-equivariant, but all others are shift-invariant. Other nice properties of cumulants include homogeneity and additivity. Homogeneity states that $\kappa_i[cX] = c^i \kappa_i[X]$. Additivity states that for independent random variables X and Y, $\kappa_i[X+Y] = \kappa_i[X]+\kappa_i[Y]$. We will see in Chapter 7 that these properties make cumulant moments very convenient for characterizing the distribution of $T(x)$.

2.3.2 Queueing-specific notation

Probably the most fundamental pieces of the queueing model are the interarrival distribution and the service distribution. We will use the following notation to describe these distributions. We will denote a generic service time (job size) as X. The cumulative distribution function (c.d.f.) of X will be denoted by $F(x)$, with $\overline{F}(x) = 1 - F(x)$. Further, the probability density function (p.d.f.) of X will be denoted by $f(x)$. It will turn out that the *failure rate* (hazard rate) of job sizes will play an important role in the performance of a number of scheduling policies. We denote the failure rate of X by $\mu(x) = \frac{f(x)}{\overline{F}(x)}$. Moving to the interarrival times, we will let A denote a generic interarrival time, and we will denote the average arrival rate by λ.

Another fundamental quantity is the system load (utilization), which we define as $\rho = \lambda E[X] = E[X]/E[A]$. We will almost exclusively be discussing stationary queues, thus we will require that $\rho < 1$.

Beyond the above quantities, there are many other random variables that are of interest. In fact, we have already defined a number of important performance metrics. We summarize these below for easy reference:

T	a generic response time
$T(x)$	the response time for a job of size x
S	the slowdown (stretch)
$S(x) = T(x)/x$	the slowdown for a job of size x
W	the waiting time
$W(x)$	the waiting time for a job of size x
R	the residence time
$R(x)$	the residence time for a job of size x

Table 2.3: *A brief overview of some of the most common performance metrics in the thesis.*

It is worth spending a moment discussing these measures. First of all, notice that the response time of a job is simply the sum of the residence time and the waiting time of the job, thus we have that $T(x) \stackrel{\mathrm{d}}{=} R(x) + W(x)$. Further, though we will most commonly be interested in studying T, we will also use $T(x)$ to understand how the response time of a job depends on the size of a job. Since $T(x)$ grows linearly with x, since the response time of a job includes (at minimum) the job size, we will often use figures showing $E[T(x)]/x$ in order to contrast the behavior of conditional response time across different job sizes. This is a useful measure because, in many cases, it is appropriate for response times to be proportional to job size (i.e. small jobs should have small response times and large jobs should have large response times). We will discuss this in much more detail in Chapter 7.

There are also a number of other important system measures that will appear throughout the thesis. In particular, we will denote the *work in the system* seen by an arrival as Q, the *number of jobs in system* seen by an arrival as N, and the *number of jobs in the queue* seen by an arrival as N_q.

One quantity, above all others, will prove to be fundamental to the analysis of many scheduling policies: the length of a busy period. A *busy period* is simply the time between when a server switches from idle to busy and the time when the server next becomes idle. We will denote the length of a generic busy period as B, and we will denote the length of a busy period started by a job of size x as $B(x)$. In addition to these two types of busy periods, we will find that a number of other types of busy periods are useful in the analysis of priority-based scheduling disciplines. But, we will wait until a bit later (Section 3.2.1) to introduce these other types of busy periods.

In addition to busy periods, two other quantities that are fundamental to the analysis of scheduling policies are the *age* and *excess* (a.k.a. residual life) of the service distribution. These quantities are best defined by considering a job in service as seen by an arriving job to a simple FCFS queue. The age of the job in service is the amount of service it has received so far, and the excess of the job in service is the remaining size of the job. To define age and excess more formally, let us consider a renewal process where the distribution of renewal times follows the service distribution. Now consider a random point in time t, which corresponds to the job that arrived to the FCFS queue. The age at time t is the time since the last renewal (the attained service) and the excess at time t is the time until the next renewal (the remaining size). A fundamental result in renewal theory is that the age and excess follow the same distribution. We will denote the age by $\mathcal{A}$ and the excess by $\mathcal{E}$. The moments, the p.d.f., and the transform of $\mathcal{E}$ are well known and will occur frequently in this thesis:

$$\begin{aligned} E[\mathcal{E}^i] &= \frac{E[X^{i+1}]}{(i+1)E[X]} \\ f_{\mathcal{E}}(x) &= \frac{\overline{F}(x)}{E[X]} \\ \mathcal{L}_{\mathcal{E}}(s) &= \frac{1 - \widetilde{X}(s)}{sE[X]} \end{aligned}$$

2.4 Commonly used distributions

At the core of our queueing model are the distributions that the interarrival times and service times of the queue follow. Since many of these distributions are not common across computer science, we will spend

a significant amount of time here introducing the distributions that are most commonly used in the thesis. By summarizing the notation here, we can avoid introducing this notation repeatedly throughout the thesis. Further, this section can serve as a single point of reference for the reader while working through the thesis.

Our discussion in this section will focus on two classes of distributions: *phase type distributions*, which include exponential, Coxian, hyperexponential, and Erlang distributions, and *heavy-tailed distributions*, which include Pareto, Weibull, subexponential, and regularly varying distributions. For each class, we will discuss properties of the class as a whole in addition to providing background on the most common individual distributions in the class. We will start with a discussion of phase-type distributions and then move to heavy-tailed distributions.

2.4.1 Phase-type distributions

Probably the most fundamental distribution in queueing is the exponential distribution. An exponential distribution with rate λ is defined by

$$\begin{aligned} \overline{F}(x) &= e^{-\lambda x} \\ f(x) &= \lambda e^{-\lambda x} \end{aligned}$$

and has moments $E[X^i] = \frac{i!}{\lambda^i}$. The defining characteristic of an exponential distribution is the fact that it has a constant failure rate: $\mu(x) = \lambda$. This means that the exponential is "memoryless," i.e. regardless of how long an exponentially distributed task has been run, it's remaining size is identically distributed.

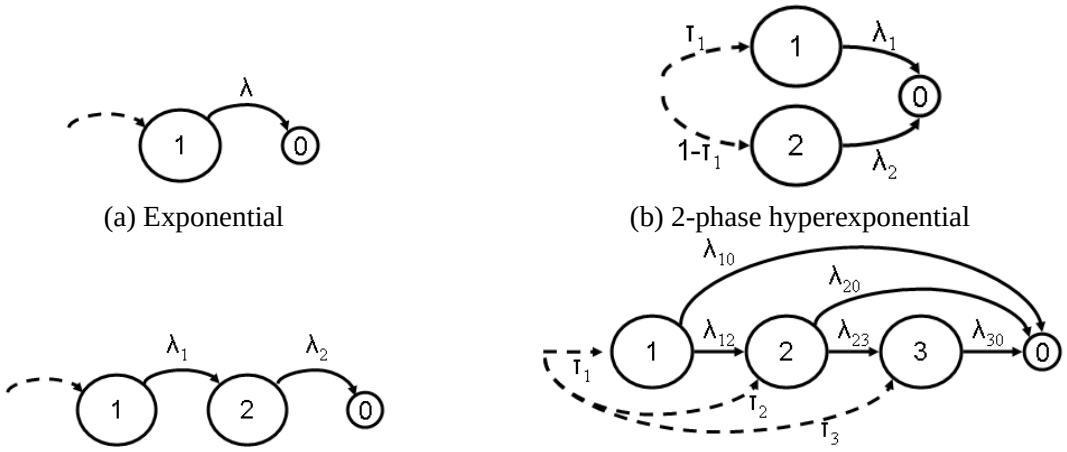
The prominence of the exponential distribution in queueing is a result of the fact that when one assumes the interarrival times and service times of a queue both follow exponential distributions, the queue length of the system can easily be modeled using a simple Markov chain. However, in most settings, job sizes are far from exponential. In computer and telecommunication applications job sizes tend to be much more variable than an exponential, and in manufacturing and inventory applications job sizes tend to be much less variable than an exponential. Thus, this simple Markov chain is often unrealistic.

However, one would still like to be able to use a Markov chain to analyze systems with non-exponential service and interarrival times. To do this, it is necessary to use some mixture of exponential distributions to model a distribution with more/less variability than an exponential. The class of phase type distributions (PH) captures many such mixtures.

At its most general, a PH distribution is simply the distribution of time until absorption into state 0 in a Markov chain. Thus, it can be viewed as an arbitrarily complex mixture of exponential random variables (the lengths of the visits to each state). To characterize a PH distribution, we can use the infinitesimal generator of its underlying Markov chain, denoted $\mathbf{T}$, and a distribution of the starting point in the chain, denoted $\vec{\tau}$. We will use the notation $X \sim PH(\vec{\tau}, \mathbf{T})$ to indicate that X is distributed as the time until absorbtion into state 0 in a Markov chain on states $(0, 1, \ldots, n)$ with initial probability vector $(1 - \vec{\tau}\vec{1}, \vec{\tau})$ and infinitesimal generator

$$\begin{pmatrix} 0 & \vec{0} \\ \vec{t} & \mathbf{T} \end{pmatrix}$$

where $\vec{t} = -\mathbf{T}\vec{1}$.



(a) Exponential (b) 2-phase hyperexponential

(c) 2-phase Erlang (d) 3-phase Coxian

Figure 2.1: *Simple examples of phase-type distributions.*

Given the definition of PH distributions, it is not hard to write the c.d.f. and p.d.f. of PH distributions:

$$\begin{aligned} \overline{F}(x) &= \vec{\tau}\exp(\mathbf{T}x)\vec{1} \\ f(x) &= \vec{\tau}\exp(\mathbf{T}x)\vec{t} \end{aligned}$$

where $\vec{t} = -\mathbf{T}\vec{1}$ and $\exp(\mathbf{X}) = \sum_{i=0}^{\infty} \frac{1}{i!}\mathbf{X}^i$.

Further, the moments of PH distributions can be written quite succinctly

$$E[X^i] = i!\vec{\tau}(-\mathbf{T}^{-1})^i\vec{1}$$

Obviously the class of PH distributions is quite broad. In fact, the set of PH distributions is dense in the set of nonnegative, continuous distributions [154]. In practice though, using PH distributions that rely on large complex Markov chains is computationally prohibative; thus it is common to use only a small portion of the set of PH distributions for analysis.

We provide some examples of PH distributions in order to solidify the general definition we have just stated. The simplest form of a PH distribution is the exponential distribution. The exponential distribution is a PH distribution with $\mathbf{T} = -\lambda$ and $\vec{\tau} = 1$. Specializing the formulas for the c.d.f. and p.d.f. then returns the well-known formulas stated above.

Beyond the exponential distribution three of the most common classes of PH distributions are Erlang distributions, hyperexponential distributions, and Coxian distributions. Before we define these classes, note that these PH distributions are all defined by *acyclic* Markov chains. That is, no state in the Markov chain is visited more than once. Further, these PH distributions are typically indexed by the number of non-absorbing states in the Markov chain. For example, an n-phase Erlang distribution is an Erlang distribution where the Markov chain defining the distribution has n non-absorbing states.

2.4.1.1 The Erlang distribution

The Erlang distribution was developed by Agner K. Erlang to examine the number of telephone calls occurring at the same time at switching stations. An n-phase Erlang distribution is simply the sum of n i.i.d. exponential distributions. Thus, the Markov chain has n non-absorbing states labeled $1, \ldots, n$ and state i transitions only to state $i-1$. To illustrate this, consider the example of a two phase Erlang where the both phases have rate λ (see Figure 2.1). In this case, we have that $\vec{\tau} = (0, 1)$ and

$$\mathbf{T} = \begin{pmatrix} -\lambda & 0 \\ \lambda & -\lambda \end{pmatrix}$$

In general, we can write the c.c.d.f. and p.d.f. of the Erlang as follows:

$$\begin{aligned} \overline{F}(x) &= \sum_{i=0}^{n-1} e^{-\lambda x} \frac{(\lambda x)^i}{i!} \\ f(x) &= \frac{\lambda^n x^{n-1} e^{-\lambda x}}{(n-1)!} \end{aligned}$$

Further, it is easy to see that the mean of an Erlang is n/λ and the variance is n/λ^2. Thus, Erlang distributions are all less variable (i.e. have a smaller coefficient of variation) than an exponential distribution with the same mean. Specifically, $C^2[X] = 1/n \leq 1$ for Erlang distributions.

2.4.1.2 The hyperexponential distribution

An n-phase hyperexponential distribution is simply the mixture of n exponential distributions. Thus, the Markov chains consist of n non-absorbing states, all of which transition only to the absorbing state. To illustrate this, consider the example of a two phase hyperexponential where the two phases have rates λ_1 and λ_2 and the probability of beginning in phase 1 is τ_1 (see Figure 2.1). In this case, we have that $\vec{\tau} = (\tau_1, 1-\tau_1)$ and

$$\mathbf{T} = \begin{pmatrix} -\lambda_1 & 0 \\ 0 & -\lambda_2 \end{pmatrix}$$

In general, if we consider an n-phase hyperexponential with rates λ_i and corresponding initial probabilities τ_i for $i = 1, \ldots, n$, we have that

$$f(x) = \sum_{i=1}^{n} \tau_i \lambda_i e^{-\lambda_i x}$$

Thus, the mean of a hyperexponential is $\sum_{i=1}^{n} \frac{\tau_i}{\lambda_i}$ and the second moment is $\sum_{i=1}^{n} \frac{2\tau_i}{\lambda_i^2}$. Clearly, hyperexponential distributions are always more variable than an exponential distribution with the same mean, i.e. $C^2[X] \geq 1$ for all hyperexponential distributions.

2.4.1.3 The Coxian distribution

Unlike the Erlang and hyperexponential distributions that we just discussed, the class of Coxian distributions includes both low variability distributions and highly variable distributions. Thus, they are useful in a variety of settings. We define a PH distribution as a Coxian distribution when the Markov chain used to define

the distribution is acyclic and maintains the property that every state has at most one transition to a non-absorbing state. A simple example of this is the three phase Coxian distribution diagramed in Figure 2.1. Formally, this Coxian distribution is defined by $\vec{\tau} = (\tau_1, \tau_2, \tau_3)$ and

$$\mathbf{T} = \begin{pmatrix} -\lambda_{10} - \lambda_{12} & \lambda_{12} & 0 \\ 0 & -\lambda_{20} - \lambda_{23} & \lambda_{23} \\ 0 & 0 & -\lambda_{30} \end{pmatrix}$$

2.4.2 Heavy-tailed distributions

Over the past decade there has been growing empirical support for the assertion that the service time distributions in computer systems should be modeled by distributions with power-tails and infinite variance. Examples where power-tails and infinite variance have emerged are numerous [16, 62, 172, 174]; for example, Crovella and Bestavros [62] find that the file sizes and transmission times of files in the Internet are well modeled by power-tail distributions with infinite variance.

The emergence of these "heavy-tailed" distributions has had a huge impact on the design of scheduling policies for computer systems; thus an important focus for queueing theorists has been to study this impact theoretically. Unfortunately though, the class of PH distributions is not appropriate for such a study. Despite the fact that the class of PH distributions is dense in the set of all non-negative distributions, in practice PH distributions cannot model distributions with power-tails and infinite variance because, for any finite sized Markov chain, the tail of a PH distribution will decay with an exponential rate and all the moments of the distribution will be finite. However, there is no shortage of heavy-tailed distributions for queueing theorists to use, though none have proven to be as useful for computational queueing theory as PH distributions have proven to be. Instead, heavy-tailed results tend to be asymptotic in nature, characterizing only the asymptotic tail behavior of distributions.

Formally, the class of heavy-tailed distributions is defined as follows.

Definition 2.1 *We say that a distribution $\overline{F}$ is **heavy-tailed** if for all $s > 0$*

$$E[e^{sX}] = \infty,$$

or equivalently if for all $\epsilon > 0$,

$$\lim_{x \to \infty} \frac{\overline{F}(x)}{e^{-\epsilon x}} = \infty.$$

*In contrast, we say a distribution is **light-tailed** if $E[e^{sX}] < \infty$ for some $s > 0$.*

Clearly, many distributions are heavy-tailed, e.g. the Weibull, Pareto, and Lognormal distributions. However, the breadth of the class of heavy-tailed distributions typically makes it difficult to use for analysis. Instead, individual heavy-tailed distributions or subclasses of heavy-tailed distributions are often used. In this section, we will introduce two of the most common heavy-tailed distributions: the Weibull and the Pareto distributions. These two distributions will be used to provide illustrative examples throughout the thesis. In addition, in this section we introduce two large classes of heavy-tailed distributions that have properties which facilitate their use in analysis: the class of regularly varying distributions and the class of subexponential distributions.

2.4.2.1 The Weibull distribution

Introduced by Waloddi Weibull, the Weibull distribution is often used in the field of data analysis due to its flexibility. Depending on parameters, the Weibull can behave as a Normal distribution, an exponential distribution, or a heavy-tailed distribution. Similarly, its usefulness in queueing stems from its flexibility. Depending on parameters it can have a decreasing, increasing, or even a constant failure rate. Further, the Weibull has recently emerged as a good model of empirical distributions in many computer applications. See for example, [28, 69, 127, 174] and the references therein.

A Weibull distribution is summarized by two parameters: a shape parameter, α, and a scale parameter, λ. We will write $X \sim Wei(\alpha, \lambda)$ to indicate that X follows a Weibull distribution. The c.d.f. and p.d.f. of the Weibull distribution are defined as follows:

$$\begin{aligned} \overline{F}(x) &= e^{-\left(\frac{x}{\lambda}\right)^{\alpha}} \\ f(x) &= \frac{\alpha x^{\alpha-1}}{\lambda^{\alpha}} e^{-\left(\frac{x}{\lambda}\right)^{\alpha}} \end{aligned}$$

It is relatively straightforward to calculate the moments of a Weibull from the c.d.f. In particular, we can write the moments as follows:

$$E[X^i] = \lambda^i \Gamma\left(1 + \frac{i}{\alpha}\right)$$

where $\Gamma(a) = \int_0^{\infty} x^{\alpha-1} e^{-x} dx$ and can be thought of as a continuous version of the factorial function. In particular, $\Gamma(n) = (n-1)!$ for any positive integer n.

Further, the failure rate of the Weibull is as follows:

$$\mu(x) = \frac{\alpha x^{\alpha-1}}{\lambda^{\alpha}}$$

Notice that $Wei(\alpha = 1, \lambda)$ is equivalently an exponential distribution with rate $1/\lambda$. Thus, the failure rate of a Weibull is constant if $\alpha = 1$. In addition, we can see that the failure rate is decreasing when $\alpha < 1$ and increasing when $\alpha > 1$. Further, it is easy to see that $C^2[X] > 1$ when $\alpha < 1$ and $C^2[X] < 1$ when $\alpha > 1$. To get a feeling for the how variable a Weibull with $\alpha < 1$ can be, notice that for $\alpha = 1/n$ where n is limited to positive integer values, we have that $C^2[X] = \binom{2n}{n} - 1$. Thus, as α decreases the distribution becomes more variable very quickly. Typical observed values for α in computer applications range between 1/3 and 2/3 which correspond to $C^2[X]$ values in the range of 3 to 19.

2.4.2.2 The Pareto distribution

The Pareto distribution, named after Italian economist Vilfredo Pareto, is a power-law distribution that arises across an amazing variety of real-world situations. Pareto originally used the distribution to describe the wealth of individuals, since it models the property that a large portion of the wealth is owned by a tiny fraction of the population. More simply, the Pareto distribution captures the "Pareto principle," a.k.a. the "80-20 rule," which states that 80% of the population owns 20% of the wealth. However, following its use in economics, the Pareto distribution has been found to be broadly applicable. It has been used to describe the frequency of words, the populations of cities, the sizes of sand particles, the size of areas burnt by forest fires, and many other phenomenon. Even within computer systems, the Pareto distribution serves as a good model of many phenomenon including, but not limited to, UNIX process lifetimes [91, 68], web file sizes

[173, 62], number of embedded files in web sites [16, 28].

A Pareto distribution is summarized by 2 parameters. A shape parameter α and a scale parameter x_L. We will write $X \sim Pareto(\alpha, x_L)$ to indicate that the r.v. X follows a Pareto distribution. The distribution of a Pareto is as follows (for $x > x_L$):

$$\begin{aligned} \overline{F}(x) &= \left(\frac{x_L}{x}\right)^{\alpha} \\ f(x) &= \frac{\alpha x_L^{\alpha}}{x^{\alpha+1}} \end{aligned}$$

From the above, it is easy to see that the i-th moment of a Pareto distribution will only be finite if $\alpha > i$. Specifically, we can write moments of the Pareto as follows:

$$E[X^i] = \frac{\alpha x_L^i}{\alpha - i} \quad \text{for } i < \alpha$$

Thus, the variance of the Pareto (assuming $\alpha > 2$) is

$$Var[X] = \frac{\alpha x_L^2}{(\alpha-1)^2(\alpha-2)}$$

which gives

$$C^2[X] = \frac{1}{\alpha(\alpha-2)}$$

Note however that it is quite typical for the variance and squared coefficient of variation to be infinite in real world situations: the Pareto distributions that emerge in computer system applications typically have $\alpha \in (0.9, 1.3)$.

Another important statistic of the Pareto distribution is the failure rate, which can be written as

$$\mu(x) = \frac{\alpha}{x}$$

The key observation about the above is that the Pareto has a decreasing failure rate (DFR) for all α.

In many computer systems where Pareto distributions arise there is some natural upper bound on the maximum size in the distribution. For instance, in measuring the distribution of file sizes, the size of the disk they reside on serves as an upper bound on the size. Thus, often times, a bounded Pareto distribution is used instead of the unbounded Pareto, e.g. [93, 92, 25]. The only difference between a bounded and an unbounded Pareto is that the bounded Pareto is defined on a finite range. In particular, one additional parameter is needed to define a bounded Pareto distribution, x_U, which is the upper bound on the distribution. We will write $X \sim BP(\alpha, x_L, x_U)$ to indicate that X follows a bounded Pareto distribution. The c.d.f. and p.d.f. of a bounded Pareto are:

$$\begin{aligned} \overline{F}(x) &= \frac{\left(\frac{x_L}{x}\right)^{\alpha} - \left(\frac{x_L}{x_U}\right)^{\alpha}}{1 - \left(\frac{x_L}{x}\right)^{\alpha}} \\ f(x) &= \frac{\alpha x_L^{\alpha}}{x^{\alpha+1}(1 - (x_L/x_U)^{\alpha})} \end{aligned}$$

Clearly, the range of the bounded Pareto is $[x_L, x_u]$. Thus, the bounded Pareto is *not* a heavy-tailed distribution, though it is highly variable and maintains the so called "heavy-tailed property" mentioned in [93, 92, 25] whereby more than one half of the load is made up by the largest 1% of the jobs.

There are a few key differences between the bounded Pareto and the unbounded Pareto. First, it is important to notice that all moments of the bounded Pareto are finite regardless of α. We can write them as follows:

$$E[X^i] = \begin{cases} \frac{\alpha x_L^i}{\alpha - i}\left(\frac{1-(x_L/x_U)^{\alpha-i)}}{1-(x_L/x_U)^{\alpha}}\right), & \text{for } \alpha > i; \\ \frac{\alpha x_L^i}{1-(x_L/x_U)^{\alpha}} \log\left(\frac{x_U}{x_L}\right), & \text{for } \alpha = i; \\ \frac{\alpha x_L^{\alpha}}{i-\alpha}\left(\frac{x_U^{\alpha-i} - x_L^{\alpha-i}}{1-(x_L/x_U)^{\alpha})}\right), & \text{for } \alpha < i. \end{cases}$$

Further, the failure rate of the bounded Pareto is not strictly decreasing – it increases unboundedly near x_U:

$$\mu(x) = \frac{\alpha}{x(1-(x/x_U)^{\alpha})}$$

2.4.2.3 Regularly varying distributions

The class of *regularly varying distributions* is a generalization of the Pareto distributions that we just discussed. Intuitively, a distribution is regularly varying if its tail asymptotically decays according to a power-tail. Formally, we have the following:

Definition 2.2 *We say that $\overline{F}$ is a **regularly varying distribution** with index α, denoted $\overline{F} \in \mathcal{RV}(\alpha)$, if*

$$\overline{F}(x) = L(x)x^{-\alpha}$$

*where $L(x)$ is a **slowly varying function**, i.e. $L(x)$ is such that for all $y > 0$*

$$\lim_{x\to\infty} \frac{L(yx)}{L(x)} = 1$$

Regular variation is a topic of research in its own right, with broad ranging applications in fields such as complex analysis and number theory in addition to its uses in probability theory. Even within probability theory, regular variation has found application in a number of areas, of which queueing is only one.

To begin developing an understanding of the properties of regularly varying distributions. It is useful to start by discussing slowly varying functions. Examples of slowly varying functions are constants and logarithms. In general, slowly varying functions are exactly those functions that can be treated as a constant in the asymptotic evaluation of integrals of power functions. In particular, for $\alpha > 1$,

$$\int_x^{\infty} y^{-\alpha} L(y)dy \sim \frac{1}{\alpha - 1} x^{1-\alpha} L(x)$$

Building on this characterization of slowly varying functions, we obtain a number of nice properties of regularly varying distributions. In particular, it is easy to see that regularly varying functions asymptotically behave like $1/x^{\alpha}$ with respect to integration and differentiation. This is typically referred to as Karamata's Theorem [33]. Formally, it states that

Theorem 2.1
If $\overline{F} \in \mathcal{RV}(\alpha)$ with $\alpha > 1$ then $\overline{G}(x) = \int_x^\infty \overline{F}(t)dt \in \mathcal{RV}(\alpha - 1)$ and further

$$x\overline{F}(x) \sim (\alpha - 1)\overline{G}(x)$$

A number of interesting properties of regularly varying distributions follow easily from Karamata's Theorem. In particular, it is straightforward to see that

$$E[X^i] < \infty \quad \Leftrightarrow \quad \alpha > i$$

Further, if $X \in \mathcal{RV}(\alpha_1)$ and $Y \in \mathcal{RV}(\alpha_2)$ then $X + Y \in \mathcal{RV}(\min(\alpha_1, \alpha_2))$. Similarly, if $X \in \mathcal{RV}(\alpha)$ and Y has $E[Y^{\alpha+\epsilon}] < \infty$ for some $\epsilon > 0$ then $XY \in \mathcal{RV}(\alpha)$.

Though the above properties of regularly varying distributions are important, probably the most useful tool for studying queues with regularly varying service distributions is the following Tauberian theorem, which relates the asymptotic behavior of a regularly varying distribution function with the behavior of its transform. Before stating the theorem, let us define $\phi_n(s)$ as the following function of the transform:

$$\phi_n(s) = (-1)^{n+1}\left(\mathcal{L}_X(s) - \sum_{j=0}^{n} E[X^i]\frac{(-s)^j}{j!}\right)$$

where $E[X^n] < \infty$.

Theorem 2.2
Let n be an integer such that $n < \alpha < n+1$, L be a slowly varying function, and $C \geq 0$. Then the following are equivalent

$$\begin{aligned} \phi_n(s) &\sim (C + o(1))s^\alpha L(1/s) \quad s \downarrow 0 \\ \overline{F}(x) &\sim (C + o(1))\frac{(-1)^n}{\Gamma(1-\alpha)}x^{-\alpha}L(t) \end{aligned}$$

The proof of this theorem was provided by Bingham and Doney [32] for the case of $C > 0$ and by Boxma and Dumas [43] for the case when $C = 0$. When α is integer, things become more complex and we refer the reader to [33].

The power of Theorem 2.2 in queueing settings comes from the fact that it is typically possible to derive explicit expressions for the transforms of various performance metrics, but inverting these transforms is often impossible. In such cases, Theorem 2.2 provides a simple way to obtain asymptotic information about the distribution from the transform.

There are many extensions of regularly varying distributions that have arisen in queueing applications. Two of these generalizations, introduced originally by Cline [56], will appear in this thesis: the class of *intermediate regular variation* and the class of *O-regular variation*. These extensions appear in many ways to be superficial generalizations of the class of regularly varying distributions; however they often turn out to provide precisely the characterization of the distribution necessary for proofs.

Definition 2.3 *We say that $\overline{F}(x)$ is of **intermediate regular variation**, denoted $\overline{F} \in \mathcal{IR}$, if*

$$\lim_{\epsilon \downarrow 0} \liminf_{x \to \infty} \frac{\overline{F}((1+\epsilon)x)}{\overline{F}(x)} = 1$$

Definition 2.4 *We say that $\overline{F}(x)$ is **O-regularly varying**, denoted $\overline{F} \in \mathcal{OR}$, if for some $\Lambda > 1$,*

$$0 < \liminf_{t \to \infty} \frac{\overline{F}(\lambda t)}{\overline{F}(t)} \leq \limsup_{t \to \infty} \frac{\overline{F}(\lambda t)}{\overline{F}(t)} < \infty \text{ for all } \lambda \in [1, \Lambda]$$

It is easy to see that if $X \in \mathcal{RV}$ then $X \in \mathcal{IR}$, and further, if $X \in \mathcal{IR}$ then $X \in \mathcal{OR}$. In addition, Cline [56] shows that $\mathcal{IR}$ and $\mathcal{OR}$ both have Karamata Theorems that parallel Theorem 2.1 for regularly varying distributions.

2.4.2.4 Subexponential distributions

We now move to a class of heavy-tailed distributions that generalizes all the heavy-tailed distributions we have studied so far: the class of *subexponential distributions*. The class of subexponential distributions was introduced independently by Chistyakov [54] and Chover et al. [55] in order to study the asymptotic properties of branching processes. However, the class has proven useful far beyond the domain of branching processes. The reason for the usefulness of subexponential distributions is that, intuitively, subexponential distributions are the class of distributions where a large sum is most likely the result of a single large summand – other summands make only a negligible contribution to the sum. This "catastrophe principle" has led to subexponentiality becoming a common paradigm for insurance mathematics and queueing.

Let us now state the formal definition of the class of subexponential distributions. To do so, we will denote the n-fold convolution of a distribution function F by $F^{n*}(x)$.

Definition 2.5 *We say that $\overline{F}$ is **subexponential**, denoted $\overline{F} \in \mathcal{S}$, if for some $n \geq 2$*

$$\overline{F}^{n*}(x) \sim n\overline{F}(x) \tag{2.3}$$

Equivalently, $\overline{F}$ is subexponential if for some $n \geq 2$

$$P(X_1 + \ldots + X_n > x) \sim P(\max(X_1, \ldots, X_n) > x) \tag{2.4}$$

In many cases the relation (2.4) is more appealing than (2.3); however both are useful in practice. To intuitively see why the two are equivalent notice that the distribution of $\max(X_1, \ldots, X_n)$ satisfies

$$P(\max(X_1, \ldots, X_n) > x) = 1 - (1 - \overline{F}(x))^n \sim n\overline{F}(x)$$

Clearly, the class of subexponential distributions is very broad. We have already commented that the class includes Pareto, Weibull, and regularly varying distributions. In addition, the class includes an array of other common heavy-tailed distributions. For example, Lognormal distributions are subexponential. (Recall that Lognormal distributions have $P(X > x) = P(e^{\mu+\sigma N} > x)$ where N is a standard normal random variable.)

It is important to point out that the class of subexponential distributions has a number of counterintuitive properties. For example, the class is *not* closed under convolution: if X and Y follow independent subexponential distributions, $X + Y$ is not necessarily subexponential. Similarly, the product XY is not necessarily subexponential. However, results from Leslie [128] and Cline & Samorodnitsky [57] prove that if Y is sufficiently well-behaved then both $X + Y$ and XY will be subexponential.

CHAPTER 3

An introduction to common policies

Our goal in this thesis is to develop a new style of scheduling research that can bridge the gaps between the needs of practitioners and the analytic results proven by theoreticians. As we have described in Chapter 1, there are many ways in which traditional theoretical results do not match the needs of system designers. To list a few, traditional results focus on idealized policies, focus on a limited set of metrics, and focus on simplistic models. However, before we can move forward and describe how to bridge these gaps, we need to understand both the traditional analytic results about common policies and the techniques used to derive these results. Providing this background is the goal of this chapter.

In particular, our goal in this chapter is threefold. First and foremost, our goal is to *introduce* the policies that form the basis of scheduling research in queueing. The scheduling community has studied a wide variety of policies, and we must first understand these idealized policies if we are to move beyond them to more practical policies. For instance, we must first understand how SRPT performs before we can study the impact of using only estimates of remaining sizes.

Secondly, our goal is to provide insight into the *performance* of these policies by summarizing and contrasting the results characterizing response times under these policies. In order to do this we will both (i) survey the existing literature deriving the moments and transform of response time under a variety of scheduling policies, and (ii) derive new results characterizing the behavioral properties of response time as a function of system load and job size variability under a variety of scheduling policies.

Finally, our third goal is to provide background into the *techniques* used to analyze response times under these policies. Thus, we will provide insight into which analytic techniques are most appropriate for which policies. This will be important for our analysis of scheduling classifications later in the thesis.

We will start the chapter in Section 3.1 by discussing a group of the simplest and most common policies: First-Come-First-Served (FCFS), Preemptive-Last-Come-First-served (PLCFS), non-preemptive blind policies, and Processor-Sharing (PS). These policies are some of the most commonly used policies both in computer systems and beyond. FCFS is used at a packet level in routers, as well as in lock queues in databases and at supercomputing centers. Further, of course, FCFS is used whenever you wait in a line in your everyday life. PLCFS is also quite common in computer systems – the operation of a stack is governed using PLCFS. But, PS is perhaps the most common scheduling policy in computer systems. From web servers to operating systems to flow scheduling at routers, PS is at the core of time-sharing applications at all levels of computer systems.

Though simple policies like FCFS and PS are traditionally the most common policies used in computer systems, more recently, there has been a growing trend towards using designs based on priority-based policies. There are a number of different flavors of priority-based policies. In Section 3.2 we will discuss policies that prioritize based on many different criteria including job sizes, job remaining sizes, or job ages (the attained service of the job). As we discussed in Chapter 1, applications of priority based policies have recently been suggested in web servers [96, 182], routers [179, 180], wireless networks [102], operating systems [74], databases [138] and beyond. Priority-based policies are being suggested primarily for two reasons. First, by giving priority to jobs with small sizes, it is possible to provide dramatic improvements in mean response time when compared with simple policies like PS and FCFS. For example, SRPT is known to optimize the mean response time. Second, it is often desirable to provide service differentiation in computer applications. For example, it is common to provide customers different levels of service depending on how much they pay.

Finally, we conclude this introduction with a small note to the reader. This chapter is meant primarily to provide an up-to-date background of results and techniques in stochastic scheduling, thus much of it may be familiar to readers who are experts in the field. However, be aware that there are a large number of results in this chapter that have only appeared in the past few years. In addition, there are many results that are new to this thesis. In particular, over the last three years a new understanding of the effects of variability and load on priority-based policies has begun to emerge. Specifically, there are a number of new results in this chapter characterizing the "near insensitivity" and the "heavy-traffic growth rate" of mean response time under policies that prioritize small jobs.

3.1 Simple policies

We will start our overview of common scheduling policies with the simplest and most common policies: First-Come-First-Served (FCFS), Preemptive-Last-Come-First-served (PLCFS), non-preemptive blind policies, and Processor-Sharing (PS). As we have already mentioned, these policies are some of the most common policies both in computer systems and beyond. Further, the analysis of these policies will provide an excellent overview of the building blocks necessary for analyzing more complex policies. However, be aware that the fact that these policies are governed by simple scheduling rules does not mean that the analysis of these policies is always simple! In particular, the last policy we discuss, PS, is one of the more difficult scheduling policies to study analytically.

3.1.1 First-Come-First-Served (FCFS) and the stationary workload

FCFS is the natural starting point for a discussion of scheduling in queues because in many practical settings FCFS (a.k.a. First-In-First-Out, FIFO) is the most natural scheduling discipline. Further, in theory, FCFS is probably the most commonly studied policy.

Under FCFS when a job j arrives to the system, it must wait behind all of the jobs that are already in the system; however, no later arrivals will receive service before j completes. Thus, it is easy to see that the

stationary response time of FCFS is simply the size of j plus the stationary workload, Q:

$$\begin{aligned} T^{FCFS} &\stackrel{\mathrm{d}}{=} X + Q \\ T(x)^{FCFS} &\stackrel{\mathrm{d}}{=} x + Q \end{aligned}$$

As a result, we can focus on providing results characterizing $W^{FCFS} \stackrel{\mathrm{d}}{=} Q^{FCFS}$. This is nice, because not only are results about Q^{FCFS} useful in analyzing FCFS, results about the behavior of Q^{FCFS} are useful far beyond FCFS. In particular, we will focus almost exclusively in this thesis on *work conserving* scheduling policies, and under work conserving disciplines the stationary workload Q is always the same regardless of the policy, i.e. $Q = Q^{FCFS}$. This fact means that the results provided in this section will be of great use in our analysis of other policies in this chapter.

The formula for $E[Q]$, the Pollaczek-Khinchin (P-K) mean value formula, is probably the most well-known result for the M/GI/1 queue:

$$E[Q] = \frac{\lambda E[X^2]}{2(1-\rho)} \tag{3.1}$$

This simple expression already is quite illustrative about the behavior of $E[T]^{FCFS}$. In particular, we can see that $E[T]^{FCFS} < \infty$ only when $E[X^2] < \infty$ and $\rho < 1$. Further, we can see that $E[T]^{FCFS}$ is linearly dependent on the second moment of the service distribution, thus for highly variable service distributions, such as the ones that appear in many computer systems, $E[T]^{FCFS}$ can be quite large. Further, the P-K mean value formula characterizes the behavior of $E[T]^{FCFS}$ as a function of load. We can see that the mean response time under FCFS explodes as $\rho \to 1$ at a rate of $\Theta(1/(1-\rho))$.

The derivation of the P-K mean value formula is actually quite straightforward. It also serves as a nice illustration of the *tagged job* technique, which analyzes response time by tracking the experience of a "tagged" arrival. In particular, let us consider what a tagged arrival sees when arriving to a stationary FCFS queue. First, the arrival sees N_q jobs in the queue, each having i.i.d. size X_i distributed according to the service distribution. Further, the tagged arrival sees the server busy with probability ρ and, when the server is busy, the job at the server has remaining size distributed as the excess of the service distribution, $\mathcal{E}$. Thus, we have that

$$\begin{aligned} E[Q] &= E\left[\sum_{i=1}^{N_q} X_i\right] + \rho E[\mathcal{E}] \\ &= E[N_q]E[X] + \rho E[\mathcal{E}] \\ &= \lambda E[Q]E[X_i] + \rho E[\mathcal{E}] \end{aligned}$$

where $E[N_q] = \lambda E[Q]$ via Little's Law. Next, it follows that

$$\begin{aligned} E[Q] &= \frac{\rho E[\mathcal{E}]}{1-\rho} \\ &= \frac{\lambda E[X^2]}{2(1-\rho)} \end{aligned}$$

Note that this means

$$E[T]^{FCFS} = E[X] + \frac{\lambda E[X^2]}{2(1-\rho)}$$

Beyond the mean of Q, it is also important to have information about higher moments and the distribution of Q. The P-K transform equation provides this information for the M/GI/1 queue:

$$\mathcal{L}_Q(s) = \frac{s(1-\rho)}{s - \lambda + \lambda L_X(s)} = \frac{1-\rho}{1-\rho\mathcal{L}_{\mathcal{E}}(s)} \tag{3.2}$$

This transform provides an enormous amount of information about Q. First and foremost, it allows the derivation of higher moments of Q. For instance, it is easy to calculate $E[Q^2]$, and thus $Var[Q]$:

$$\begin{aligned} E[Q^2] &= \frac{\lambda E[X^3]}{3(1-\rho)} + \frac{(\lambda E[X^2])^2}{2(1-\rho)^2} \\ Var[Q] &= \frac{\lambda E[X^3]}{3(1-\rho)} + E[Q]^2 \end{aligned}$$

Further, it facilitates the derivation of a recursive formula for higher moments. This is commonly referred to as the Takács recurrence formula:

$$E[Q^k] = \frac{\lambda}{1-\rho} \sum_{i=1}^{k} \binom{k}{i} \frac{E[X^{i+1}]}{i+1} E[Q^{k-i}]$$

In addition to its use in deriving the moments of Q, (3.2) also provides a useful new interpretation of Q. In particular, we can rewrite $\mathcal{L}_Q(s)$ as

$$\begin{aligned} \mathcal{L}_Q(s) &= \frac{1-\rho}{1-\rho L_{\mathcal{E}}(s)} \\ &= (1-\rho) \sum_{n=0}^{\infty} \rho^n L_{\mathcal{E}}(s)^n \end{aligned} \tag{3.3}$$

This form of the transform is quite intriguing because $(1-\rho)\rho^n$ is the distribution of N_q in the M/M/1 and thus (3.3) is saying that the work in this system can be viewed as coming from $N_q^{M/M/1}$ i.i.d. jobs with sizes distributed as the excess of the service distribution, i.e. $Q \stackrel{\mathrm{d}}{=} \sum_{i=0}^{N_q^{M/M/1}} \mathcal{E}_i$. It is not at all obvious why this should be true under FCFS, but we will see an explanation for this behavior later when we discuss Processor-Sharing (PS).

3.1.2 Preemptive-Last-Come-First-Served (PLCFS) and busy periods

Like FCFS, PLCFS is a very natural scheduling discipline for many applications. Under PLCFS, the server is always working on the most recent arrival to the system. So, upon arrival the job at the server is preempted, and may only resume service once the system is empty of newer arrivals. Thus, PLCFS acts as a *stack* where new jobs are placed on the top of the stack and the server is always working on the job at the

top of the stack.

The natural way to view the response time under PLCFS is in terms of the *busy period* started by an arriving job. A busy period is just the time from when a job arrives into an idle system until the system again becomes idle. To view T^{PLCFS} as a busy period, first consider that if a tagged job arrives to an idle system, then when the tagged job completes under PLCFS the system will again become idle, i.e. the response time is the same as the length of the busy period. Now, to see that this is true even when the tagged arrival does not arrive to an idle system; notice that the jobs in the system at the arrival of the tagged job are irrelevant to the response time of the tagged job. Thus, we can view the system as idle at the arrival of the tagged job without loss of generality. So, letting B be the length of a standard busy period and $B(x)$ be the length of a busy period started by a job of size x, we have that

$$\begin{aligned} T^{PLCFS} &\stackrel{\mathrm{d}}{=} B \\ T(x)^{PLCFS} &\stackrel{\mathrm{d}}{=} B(x) \end{aligned}$$

Thus, in this section, we will focus on the behavior of busy periods. Beyond their importance to PLCFS, the behavior of busy periods will be fundamental to the analysis of many other scheduling policies. In particular, we will see that busy periods are the building blocks used to study many priority based policies.

Analyzing the mean behavior of busy periods turns out to be quite straightforward. We will consider the experience of a tagged arrival j under PLCFS. As soon as the job arrives it begins service. However, that service is interrupted by any arriving job. In particular, each job that arrives while j is in service causes an interruption that consists of the arriving job j_a, and all arriving jobs until j_a completes. Thus, the interruption is of length equal to that of a busy period. Noting that the average number of arrivals while j is at the server is $\lambda E[X]$ we have

$$E[B] = E[X] + \lambda E[X]E[B]$$

which gives

$$E[B] = \frac{E[X]}{1-\rho}$$

This result can also be derived using renewal-reward techniques. The sequence of idle and busy periods in an M/GI/1 queue is clearly a renewal process. Suppose, we assign reward at a constant rate whenever the system is busy. Then the time average award earned is ρ, since this is the utilization of the queue. Further, the award earned in one idle-busy cycle is simply the length of the busy period. Finally, we know (because of the Poisson arrival process) that the expected idle time is $\frac{1}{\lambda}$. This gives:

$$\rho = \frac{E[B]}{E[B] + \frac{1}{\lambda}} \quad \Rightarrow \quad E[B] = \frac{E[X]}{1-\rho}$$

The analysis of $B(x)$ proceeds much like the derivation of $E[B]$. We will again consider the experience of a tagged job of size x. While x is at the server, other jobs arrive with rate λ and whenever a new job arrives, it causes an interruption of service to the tagged job of length equal in distribution to a busy period. Thus, letting A_x be the number of arrivals while the tagged job is at the server and B_i be i.i.d. standard busy

periods, we have

$$B(x) \stackrel{\mathrm{d}}{=} x + \sum_{i=1}^{A_x} B_i \tag{3.4}$$

from which it follows that

$$E[B(x)] \quad = \quad \frac{x}{1-\rho}$$

At this point, let us take a moment to contrast the behavior of the mean response time under PLCFS and FCFS. We will start by contrasting the behavior of $E[T]$ under the two policies. First of all, notice that both $E[T]^{PLCFS}$ and $E[T]^{FCFS}$ grow as $\Theta(1/(1-\rho))$ as $\rho \to 1$. So the behavior of mean response time as a function of load is similar under these policies. However, the effect of the second moment of the service distribution on $E[T]$ under these two policies is completely different. In particular, $E[T]^{PLCFS}$ is not affected by $E[X^2]$ while $E[T]^{FCFS}$ grows linearly with $E[X^2]$. Thus, in settings where the service distribution is highly variable $E[T]^{PLCFS}$ is much smaller than $E[T]^{FCFS}$. At the extreme, notice that $E[T]^{PLCFS} < \infty$ whenever $E[X] < \infty$ and $\rho < 1$, while $E[T]^{FCFS} = \infty$ when $E[X^2] = \infty$ even if $\rho < 1$. On the flip side however, $E[T]^{PLCFS}$ can be larger than $E[T]^{FCFS}$ if the service distribution is not variable. For instance, if the service distribution is deterministic, we have $E[T]^{FCFS} = (1-\rho/2)E[T]^{PLCFS}$, which shows that PLCFS can have response times as much as twice those of FCFS. Interestingly, it turns out that crossover point for $E[T]$ happens when $C^2[X] = 1$, i.e. when the service distribution is exponential. In particular,

$$E[T]^{PLCFS} < E[T]^{FCFS} \quad \Leftrightarrow \quad C^2[X] > 1$$

This should be intuitive because in the case of an M/M/1, preempting or not does not affect the distribution of the remaining service time of the job in service or the job being sent to the queue.

The conditional mean response times of PLCFS and FCFS also provide an interesting contrast. In both cases $E[T(x)]$ grows linearly with x, however it is clear that small job sizes prefer PLCFS to FCFS since as $x \to 0$, $E[T(x)]^{PLCFS} \to 0$ while $E[T(x)]^{FCFS}/x \to E[Q]$. In contrast, large job sizes prefer FCFS since $E[T(x)]^{FCFS}/x \to 1$ while $E[T(x)]^{PLCFS}/x \to 1/(1-\rho)$. Interestingly, the cutoff point is determined by the mean excess of the service distribution:

$$E[T(x)]^{PLCFS} \leq E[T(x)]^{FCFS} \quad \Leftrightarrow \quad x \leq E[\mathcal{E}] = \frac{E[X^2]}{2E[X]}$$

Thus, in the M/M/1 the cutoff point is simply $E[X]$ and as the service distribution becomes more highly variable the cutoff point increases.

Let us now move beyond the mean behavior of PLCFS and discuss the distributional behavior of B. Deriving the transform of B is not too much more complicated than deriving the mean behavior of B. In particular, we can make use of (3.4), which characterizes the distributional behavior of $B(x)$. We use (3.4) to derive the transform of $B(x)$, which we can then use to derive the transform of B. Beginning with $B(x)$, we have

$$\mathcal{L}_{B(x)}(s) \quad = \quad e^{-x(s+\lambda-\lambda\mathcal{L}_B(s))}$$

Next, we can decondition to obtain the transform of a standard busy period:

$$\begin{aligned}\mathcal{L}_B(s) &= \int_0^\infty e^{-x(s+\lambda-\lambda\mathcal{L}_B(s))} f(x)dx \\ &= \mathcal{L}_X(s+\lambda-\lambda\mathcal{L}_B(s))\end{aligned}$$

Notice that $\mathcal{L}_B(x)$ has a recursive form. However, it is still possible to use the transform in order to obtain higher moments of B. In particular, we have:

$$\begin{aligned}E[B^2] &= \frac{E[X^2]}{(1-\rho)^3} \\ Var[B] &= \frac{Var[X]}{(1-\rho)^3} + \lambda E[B]^3\end{aligned}$$

Similarly, for $B(x)$ we have:

$$\begin{aligned}E[B(x)^2] &= \left(\frac{x}{1-\rho}\right)^2 + \frac{\lambda x E[X^2]}{(1-\rho)^3} \\ Var[B(x)] &= \frac{\lambda x E[X^2]}{(1-\rho)^3}\end{aligned}$$

Before moving on, it is again worth taking a moment to contrast the behaviors of $Var[T]^{PLCFS}$ and $Var[T]^{FCFS}$. Similarly to what we saw in the case of the mean, $Var[T]^{PLCFS}$ does not depend on the third moment of the service distribution, while $Var[T]^{FCFS}$ does. Thus, in settings where $E[X^3]$ is large, $Var[T]^{PLCFS}$ is much smaller than $Var[T]^{FCFS}$. However, we can also notice that $Var[T]^{PLCFS}$ grows as $\Theta(1/(1-\rho)^3)$ as $\rho \to 1$ while $Var[T]^{FCFS} = \Theta(1/(1-\rho)^2)$. Thus, for high loads $Var[T]^{FCFS}$ is much lower than $Var[T]^{PLCFS}$.

3.1.3 Non-preemptive blind scheduling

FCFS is one example of a policy that is both non-preemptive, i.e. never interrupts a job receiving service, and blind, i.e. schedules without knowledge of job size information. However, there are many other common non-preemptive, blind scheduling policies. Two of the most common are Non-preemptive Last-Come-First-Served (LCFS), which serves the most recent arrival to the queue after each completion, and Random-Order-of-Service (ROS), which chooses a job to serve uniformly at random from the queue after each completion.

Though the operation of these non-preemptive blind policies seems very different, with a little thought it is easy to see that all non-preemptive blind policies behave equivalently with respect to the number of jobs in the system N. In particular, no matter how the choice of the next job to service is made after a completion, the distribution of the size of the job is the same because the decision is made without knowledge of the sizes of the job in the queue. Thus, the time until the next completion is equivalent under all of the policies. The fact that the distribution of the number of jobs in the system is the same also implies that the mean response time of all non-preemptive blind policies is the same using Little's Law.

Proposition 3.1
In an M/GI/1 queue any non-preemptive blind policy P has $N^P \stackrel{\mathrm{d}}{=} N^{FCFS}$ and thus

$$E[T]^P = E[T]^{FCFS} = E[X] + \frac{\lambda E[X^2]}{2(1-\rho)}$$

Though all non-preemptive blind policies have the same distribution on the number of jobs in the system and the same mean response time, it is important to point out the the distribution of response times can be very different. To illustrate this, let us consider the behavior of the variance of the waiting time, $Var[W]$, under FCFS, ROS, and LCFS.

We have already seen that

$$Var[W]^{FCFS} = \frac{\lambda E[X^3]}{3(1-\rho)} + \frac{\lambda^2 E[X^2]^2}{4(1-\rho)^2}$$

Under LCFS, one expects that very long response times are more likely than under FCFS, and thus the variance of the waiting time should be higher. We will derive the transform for waiting time under LCFS in order to illustrate how much higher the variance actually is.

To derive the waiting time under LCFS consider the experience of a tagged arrival to the system. An arriving job that finds the server idle experiences no waiting time. On the other hand, an arriving job j_a that finds the server busy takes precedence over all jobs waiting in the queue, but must wait for the job at the server j_s to complete. Further, j_a must wait for the busy period of arrivals started by the remaining size of j_s to complete since all later arrivals before j_s completes will receive priority over j_a. Thus, we have that

$$W^{LCFS} = B(\mathcal{E})1_{[\text{busy}]}$$

From the above, the transform and all moments of waiting time can be easily derived.

$$\begin{aligned} Var[W]^{LCFS} &= \frac{\lambda E[X^3]}{3(1-\rho)^2} + \frac{\lambda^2 E[X^2]^2}{4(1-\rho)^3} = \frac{Var[W]^{FCFS}}{1-\rho} \\ \mathcal{L}_W(s)^{LCFS} &= (1-\rho) + \frac{\lambda(1-\mathcal{L}_B(s))}{s+\lambda-\lambda\mathcal{L}_B(s)} \end{aligned}$$

Thus, while the mean waiting times of FCFS and LCFS are the same, the variances differ by a factor of $1/(1-\rho)$. This indicates a strong preference for FCFS over LCFS.

It turns out that all other non-preemptive blind policies have $Var[W]$ somewhere between $Var[W]^{FCFS}$ and $Var[W]^{LCFS}$. (See [247] page 282 for a simple proof.) For example, though the analysis of ROS is much more difficult than that for either FCFS or LCFS, $Var[W]^{ROS}$ has been derived in a few special cases. One such case is the M/M/1 [120], where we have that

$$Var[W]^{ROS} = \frac{Var[W]^{FCFS}}{1-\rho/2}.$$

3.1.4 Processor-Sharing (PS)

PS is probably the scheduling discipline most often used to model scheduling in computer systems. Unlike the policies we have discussed so far, PS is a *time-sharing* policy, i.e. the server is shared among multiple jobs at once instead of being devoted to a single job. In particular, every job in the system receives an equal share of the service capacity at all instants under PS. Thus, if the total service rate is 1 and there are N jobs in the system, then every job is being served at rate $1/N$. The observation that PS shares the server evenly at all times leads to PS being viewed as a "fair" policy in some sense. Another way to view PS is as an analytically tractable version of Round Robin (RR), which is a good approximation of what is done in many computer systems. Under RR, the job at the front of the queue receives a quantum δ of service and is then moved to the back of the queue. Thus, as $\delta \to 0$, RR becomes PS.

Though the workings of PS are very easy to state and understand, unlike the other simple policies we have discussed in this section, the analysis of the response time of PS is not straightforward.

Intuitively, we can understand the response time of PS as follows. Consider the experience of a tagged arrival j_x of size x. Since j_x arrives into a *stationary* system, j_x will share the server with $E[N^{PS}]$ other jobs, on average, until it completes. Thus,

$$E[T(x)]^{PS} = (E[N^{PS}] + 1)x = (\lambda E[T]^{PS} + 1)x \tag{3.5}$$

Integrating over x, we obtain

$$E[T]^{PS} = \rho E[T]^{PS} + E[X]$$

or equivalently

$$E[T]^{PS} = \frac{E[X]}{1-\rho} \tag{3.6}$$

Returning to (3.5) also gives

$$E[T(x)]^{PS} = \frac{x}{1-\rho} \tag{3.7}$$

The weak point in this argument is the assumption that a tagged arrival will share the system with a constant mean number of other jobs throughout its response time. Though, this assumption is by no means obvious, the expressions in (3.6) and (3.7) are correct and Mitrani points out the assumption can be made rigorous [147].

Amazingly, the expressions for $E[T]^{PS}$ and $E[T(x)]^{PS}$ exactly match the corresponding expressions for PLCFS that we derived earlier. Thus, as we discussed in Section 3.2.1, $E[T]^{PS}$ is independent of the variability of the service distribution and grows like $\Theta(1/(1-\rho))$ as $\rho \to 1$. Further, we can see that $E[T(x)]^{PS}$ is strictly linear in x. Thus, a job that is twice as long as another job will have twice as long a response time on average. This is a very appealing property with respect to "fairness," and we will return to it in Chapter 7.

So far, we have only discussed the mean response time of PS. We would obviously like to learn more about the PS queue, but to do so, we need to take a different view of the PS system. The derivation of $E[T]^{PS}$ that we have discussed so far relies on viewing the mean number of jobs in the PS system as constant throughout the response time of an arrival, but this is insufficient for studying other aspects of the

PS queue.

A more powerful viewpoint of the PS queue comes from "reversibility." This approach is detailed by Kelly [113] and Ross [193], among others. The idea is to show that the forward process of the PS queue is equivalent to the reverse process of the PS queue, and then to use this equivalence to analyze the system. Using this approach, we can derive the entire distribution of N^{PS}. Let $\vec{x} = (x_1, x_2, \ldots, x_n)$ be the queue state where x_i is the attained service (age) of the ith job in the queue.

Theorem 3.2
In an M/GI/1 queue

$$P(N^{PS} = n) \quad = \quad \rho^n(1-\rho)$$

Further, given there are n jobs in the queue, the age (excess) of each job is i.i.d. and follows the equilibrium distribution. Thus,

$$P(\vec{x} = (x_1, x_2, \ldots, x_n) | N = n) = \prod_{i=1}^{n} \frac{\overline{F}(x_i)}{E[X]}$$

This characterization of the system state of a PS queue is quite intriguing. It is amazing that the distribution of the number in the queue depends only on the system load, and is thus independent of anything but the mean of the service distribution. This property is termed "insensitivity" and is an appealing property beyond PS (see [35, 36, 37] for other practical examples of insensitive policies and queueing networks). Theorem 3.2 has many other consequences. For instance, (3.6) follows from Theorem 3.2 almost immediately by applying Little's Law:

$$\begin{aligned} E[T]^{PS} &= \frac{1}{\lambda} E[N^{PS}] \\ &= \frac{1}{\lambda} \sum_{n=0}^{\infty} n\rho^n(1-\rho) \\ &= \frac{E[X]}{1-\rho} \end{aligned}$$

Further, with a little more work we can also derive (3.7) for $E[T(x)]^{PS}$ using a special case of Little's Law applied only to jobs of size between x and $x + \epsilon$ for $\epsilon \to 0$.

Beyond the uses of Theorem 3.2 in deriving $E[T]^{PS}$ and $E[T(x)]^{PS}$, it is also important outside of PS. In particular, Theorem 3.2 provides an extremely useful view of the total work in the system, Q. In fact, it provides an explanation for the mysterious form of $\mathcal{L}_Q(s)$ that we derived in (3.3). Using Theorem 3.2, we have that

$$Q = \sum_{i=1}^{N^{PS}} \mathcal{E}_i$$

where $\mathcal{E}_i$ are independent excesses and N^{PS} has distribution $P(N^{PS} = n) = \rho^n(1-\rho)$. As we already mentioned, this form of Q is very useful for calculating moments and other statistics of of the stationary workload.

We now understand both the system state of PS and the mean response time of PS. However, even with

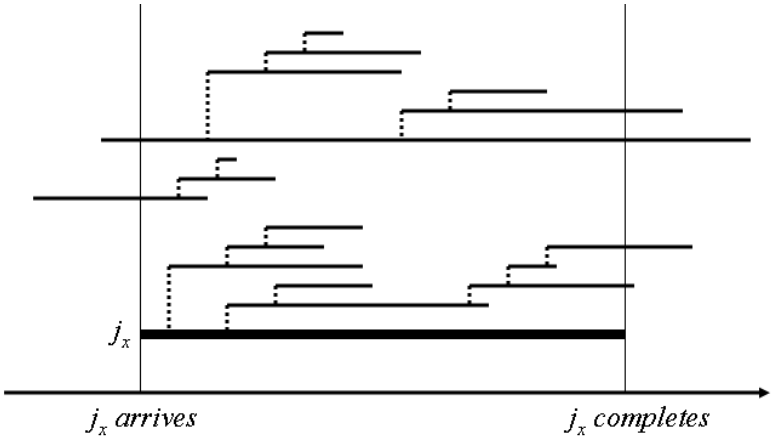


Figure 3.1: *An illustration of how to view* PS *as a branching process. In this diagram there are two jobs in the queue when the tagged job j_x arrives. The response time of j_x is the sum of the lengths of the branches between the arrival and completion instants of j_x.*

this information, it is impossible to calculate higher moments and the transform of the response time of PS. In order to calculate these, we will need yet another view of PS.

Probably the most powerful view of PS is as a branching process. This view, first introduced by Yashkov [252, 253], has led to the derivation of the transform of $T(x)^{PS}$ in the M/G1/1 queue, from which formulas for the variance and higher moments of $T(x)$ have emerged. The idea behind viewing PS as a branching process is illustrated in Figure 3.1.4. We view the response time for a tagged job j_x of size x as the sum of the lengths of branches in a random tree. Corresponding to each job, there is a branch of length equal to the size of the job. Let n_t be the number of branches at time t. Thus, when j_x arrives (at time $t = 0$) it sees $n_0 = N^{PS}$ existing branches, each having remaining length $\mathcal{E}$, and it starts a branch of length x. The process evolves as t grows by having the total arrival rate at each time t be $n_t\lambda$, split evenly among the n_t branches in the system at time t. The arrivals each form new branches attached to the branch they occurred during. One can see that this is equivalent to a PS queue by scaling the time in the branching process by n_t at each time t. Using this equivalence, it is clear that the response time of j_x is simply the sum of the lengths of the branches in the system between time 0 and time x. If we denote the sum of the lengths of the branches of a tree started by a branch of length b at height 0 between time 0 and time a as $L_a(b)$, we can use this view of PS to write $T(x)^{PS}$ as follows:

$$T(x)^{PS} \stackrel{\mathrm{d}}{=} L_x(x) + \sum_{i=1}^{N^{PS}} L_x(\mathcal{E}) \tag{3.8}$$

Though this view of PS is not as simple as the others we have discussed, it has proven to be very powerful. Let us illustrate how to use this view in the simple case of an M/D/1 queue.

Example

Our goal will be to rederive (3.7) for $E[T(x)]^{PS}$ in this simple case using the branching process.

Before we begin, let us first define some notation. Let x be the job size in the M/D/1. Further, note that the excess, $\mathcal{E}$, and age, $\mathcal{A}$, of a deterministic distribution both follow a uniform distribution. Thus, we will define U as a random variable that follows a uniform distribution on $(0, x)$.

Now, we begin by taking the expectation of (3.8):

$$E[T(x)]^{PS} \stackrel{\text{d}}{=} E[L_x(x)] + E[N^{PS}]E[L_x(\mathcal{E})] \tag{3.9}$$

Thus, in order to calculate $E[T(x)]^{PS}$ we need to derive $E[L_x(x)]$ and $E[L_x(\mathcal{E})]$.

We will derive $E[L_x(x)]$ first. In the M/D/1, $E[L_x(x)]$ is particularly easy to write since all arrivals will create new branches that extend beyond time x, which is the end of the period being studied. Thus, an arriving job at time $x - t$ will start a new subtree that will contribute $L_t(t)$ to the total tree length. Given the constant arrival rate λ, we have

$$E[L_x(x)] = \int_0^x (1 + \lambda E[L_t(t)])dt$$

Taking the derivative of both sides, gives the following differential equation,

$$\frac{dE[L_x(x)]}{dx} = 1 + \lambda E[L_x(x)]$$

which has solution

$$E[L_x(x)] = \frac{x}{\rho}\left(e^{\lambda x} - 1\right) = \frac{x}{\rho}\left(e^{\rho} - 1\right) \tag{3.10}$$

Now, we move to calculating $E[L_x(\mathcal{E})]$. Notice that in the M/D/1 it is easy to rewrite $E[L_x(\mathcal{E})]$ in terms of $E[L_x(x)]$. In particular, we have that

$$E[L_x(\mathcal{E})] = E[L_x(x)] - E[L_{\mathcal{A}}(\mathcal{A})]$$

Now, we observe that $E[L_{\mathcal{A}}(\mathcal{A})]$ is of the same form as $E[L_x(x)]$, so we can calculate it by conditioning on $\mathcal{A}$ as follows:

$$\begin{aligned}
\frac{1}{x}E[L_{\mathcal{A}}(\mathcal{A})] &= E[L_{(x-u)}(x-u)|U=u] \\
&= \int_0^x \frac{1}{\lambda}\left(e^{\lambda(x-u)} - 1\right)\frac{1}{x}du \\
&= \frac{1}{\rho^2}\left(e^{\rho} - \rho - 1\right)
\end{aligned}$$

Moving back to $E[L_x(\mathcal{E})]$, we have that

$$\begin{aligned}
E[L_x(\mathcal{E})] &= E[L_x(x)] - E[L_{\mathcal{A}}(\mathcal{A})] \\
&= \frac{x}{\rho^2}\left(\rho e^{\rho} - e^{\rho} + 1\right)
\end{aligned} \tag{3.11}$$

Finally, we are ready to put everything together and calculate $E[T(x)]^{PS}$ using (3.8), (3.10), and (3.11).

$$\begin{aligned} E[T(x)]^{PS} &= E[L_x(x)] + E[N^{PS}]E[L_x(\mathcal{E})] \\ &= \frac{x}{\rho}\left(e^{\rho}-1\right) + \left(\frac{\rho}{1-\rho}\right)\frac{x}{\rho^2}\left(\rho e^{\rho} - e^{\rho} + 1\right) \\ &= \frac{x}{\rho}\left(e^{\rho} - 1 + \frac{1}{1-\rho}\left(e^{\rho}(\rho-1)+1\right)\right) \\ &= \frac{x}{\rho}\left(\frac{\rho}{1-\rho}\right) \\ &= \frac{x}{1-\rho} \end{aligned}$$

Thus, we have arrived at the well-known result for $E[T(x)]^{PS}$.
□

Though this derivation is far more complex than our original derivation of $E[T(x)]$, and may seem like overkill for such a simple model as the M/D/1, the power of this view of PS comes in the extensibility of this argument. In particular, it is straightforward to mimic the above argument in order to derive the variance of response time in the M/D/1, and with a bit more work it can be pushed through to derive explicit formulas for the mean and variance in the M/M/1 model. Further, the branching process view of PS is at the heart of the most elegant derivations of the transform of $T(x)^{PS}$. In particular, Yashkov uses this view to derive $L_{T(x)}(s)^{PS}$ in [252, 253].

In addition to Yashkov's analysis, a number of other derivations of the transform have appeared, for example [197, 167, 256]. Probably the most useful form for the transform was provided by Zwart and Boxma in [256]. They write the transform as the following power series:

$$\mathcal{L}_{T(x)}(s)^{PS} = \left(\sum_{k=0}^{\infty}\frac{s^k}{k!}\alpha_k(x)\right)^{-1}$$

where $\alpha_0(x) = 1$, $\alpha_1(x) = x/(1-\rho)$, and for $k \geq 2$

$$\alpha_k(x) = \frac{k}{(1-\rho)^k}\int_{t=0}^{x}(x-t)^{k-1}F_Q^{(k-1)*}(t)dt$$

with $F_Q^{n*}(t)$ denoting the distribution of the n-fold convolution of the work in the system, Q. Equivalently, we could have written

$$\alpha_k(x) = \left(\frac{x}{1-\rho}\right)^k - \delta_k(x)$$

where

$$\delta_k(x) = \frac{k}{(1-\rho)^k}\int_{t=0}^{x}(x-t)^{k-1}\overline{F}_Q^{(k-1)*}(t)dt$$

The reason this form of the transform is so useful is that it is easy to use it to obtain formulas for all higher moments of $T(x)^{PS}$, which can be difficult using other forms of the transform. In particular, [256]

finds that

$$E[T(x)^k]^{PS} = -\sum_{i=1}^{k} \binom{k}{i} (-1)^i E[T(x)^{k-i}]^{PS} \alpha_i(x) \tag{3.12}$$

from which we can derive $Var[T(x)]^{PS}$:

$$Var[T(x)]^{PS} = \frac{2}{(1-\rho)^2} \int_0^x (x-t)\overline{F}_Q(t)dt \tag{3.13}$$

3.2 Priority-based policies

Though simple policies like FCFS and PS are traditionally the most common policies used to model scheduling in computer systems. More recently, there has been a growing trend towards using designs based on priority-based policies.

There are many ways that priority-based policies can assign priorities. Commonly, users are willing to pay money in order to receive high priority service and, thus, spend less time queueing. Other times, the goal may be to assign priorities in a way that minimizes some cost function (such as mean response time) of the queue. In the second scenario, it is often beneficial to give priority to small job sizes. In this section we will introduce a wide variety of priority based policies. We will talk about both non-preemptive and preemptive priority policies. Further, we will consider both policies that prioritize based on some external priority structure (such as customer payments) and policies that prioritize based on statistics of arriving jobs (such as the sizes of the jobs or the remaining sizes of the jobs).

The analysis of all priority based policies relies on using "transformations" of the service distribution and busy periods. Thus, before we talk about specific policies, it is useful to spend some time introducing the transformed service distributions and busy periods that will come up throughout the section.

3.2.1 Notation for priority-based policies

Under priority-based policies, the performance of the high priority jobs is often not impacted low priority jobs. For example, under PSJF the response time of a job with size x is unaffected by any job with size $> x$. Thus, as far as a job of size x is concerned, the service distribution may as well be cut off at x. We will see behavior similar to this under a wide variety of policies that prioritize small jobs, e.g. SRPT and FB, however the what happens to the probability mass in jobs of size $> x$ will vary depending on the scheduling policy. As a result, it will be useful to have notation to describe various ways to cut off, or truncate, the service distribution. In order to serve as a reference, we will summarize the notation for two of these here.

First, let us consider the case where all jobs with size $> x$ are simply removed from the service distribution:

$$\begin{aligned} X_x &= X1_{[X<x]} \\ m_i(x) &= E[X_x^i] = \int_0^x t^i f(t)dt \\ \rho(x) &= \lambda m_1(x) \end{aligned}$$

Here $m_i(x)$ is the i-th moment of the jobs with size $\leq x$ (ignoring all job with size $> x$ and $\rho(x)$ is the load of a system with the same arrival process and a cut-off service distribution. Notice that $\rho(x)$ can also be interpreted as the load made up by jobs with size $< x$.

Next, let us consider the case where all jobs with size $> x$ have their size reduced to x:

$$\begin{aligned} \widetilde{X_x} &= X \wedge x = \min(X, x) \\ \widetilde{m_i}(x) &= E[\widetilde{X_x}^i] = i \int_0^x t^{i-1} \overline{F}(t) dt \\ &= m_i(x) + x^i \overline{F}(x) \\ \widetilde{\rho}(x) &= \lambda \widetilde{m}_1(x) \end{aligned}$$

Again $\widetilde{m_i}(x)$ is the i-th moment of the truncated distribution and $\widetilde{\rho}(x)$ is the load of a system with the same arrival process and a truncated service distribution.

For each of these "transformed" service distributions, it will also be important to characterize how busy periods behave. These "transformed" busy periods will be fundamental to the analysis of priority based policies. For reference, Table 3.1 summarizes the various types of busy periods that we will use.

notation	service distribution	arrival rate
B, $B(y)$	X	λ
B_x, $B_x(y)$	$X_x = XI(X < x)$	λ
$\widetilde{B_x}$, $\widetilde{B_x}(y)$	$\widetilde{X_x} = \min(X, x)$	λ

Table 3.1: *A summary of the busy period variations studied in this thesis*

Clearly, the moments of B_x, $B_x(y)$, $\widetilde{B_x}$, and $\widetilde{B_x}(y)$ are all easily calculated by writing the formulas for B and $B(y)$ using the appropriate service distribution. For reference purposes, we will list them here.

$$\begin{aligned} E[B_x(y)] &= \frac{y}{1-\rho(x)} \\ Var[B_x(y)] &= \frac{\lambda y m_2(x)}{(1-\rho(x))^3} \\ E[\widetilde{B_x}(y)] &= \frac{y}{1-\widetilde{\rho}(x)} \\ Var[\widetilde{B_x}(y)] &= \frac{\lambda y \widetilde{m_2}(x)}{(1-\widetilde{\rho}(x))^3} \end{aligned}$$

3.2.2 Non-preemptive priority queues

Non-preemptive priority queues are the simplest of the priority based policies we will discuss. In non-preemptive priority queues jobs are assumed to arrive with some externally assigned priority structure. That is, arriving jobs are tagged with a certain priority level, $c = 1, 2, \ldots$, and then jobs of priority i can only move into service when the queue is empty of jobs of priority $< i$. Further, jobs within the same class are served in FCFS order. It is important to remember that we are in the non-preemptive setting, so once a job

is chosen for service, it cannot be interrupted.

The analysis of a non-preemptive priority queue is actually quite straightforward. Let W_i, N_i, X_i, λ_i, and ρ_i be the waiting time, number in queue, job sizes, arrival rate, and load of the jobs in the i-th class. We will use a tagged job approach. Consider a tagged job j from priority class $c \geq 1$. The waiting time of j consists of three pieces: (i) the remaining work in the job at the server, denoted W_0, (ii) the work from classes $1, \ldots, c$ in the queue at the arrival of j, and (iii) the work from classes $1, \ldots, c-1$ that arrives while j is in the queue. It is easy to see that the work in (i) is simply $W_0 = \mathcal{E}I(\text{busy})$. Using Little's Law, we can write (ii) as $\sum_{i=1}^{c} E[X_i]\lambda_i E[W_i]$. Further, we can write (iii) as $\sum_{i=1}^{c-1} E[X_i]\lambda_i E[W_c]$ since all arrivals during the waiting time of j from classes $1, \ldots, c-1$ receive priority over j. Thus, we have that

$$E[W_c] = E[W_0] + \sum_{i=1}^{c} E[X_i]\lambda_i E[W_i] + \sum_{i=1}^{c-1} E[X_i]\lambda_i E[W_c]$$

which gives

$$E[W_c] = \frac{E[W_0] + \sum_{i=1}^{c-1} \rho_i E[W_i]}{1 - \sum_{i=1}^{c} \rho_i}$$

This set of equations can easily be solved recursively, from which we obtain

$$E[W_c] = \frac{\lambda E[X^2]}{2(1 - \sum_{i=1}^{c-1} \rho_i)(1 - \sum_{i=1}^{c} \rho_i)} \tag{3.14}$$

where X is still the overall service distribution.

The form of this final equation is quite enlightening. We see a distinction between the effect of the jobs that are *present* in the queue at the arrival of the tagged job (the $1 - \sum_{i=1}^{c} \rho_i$ term) and the jobs that *arrive* while the tagged job is in the queue (the $1 - \sum_{i=1}^{c-1} \rho_i$ term). Intuitively, we can view the formula as stating that the expected work in the queue that finishes before the tagged job is $E[V] = \frac{\lambda E[X^2]}{2\left(1-\sum_{i=1}^{c} \rho_i\right)}$ and then the waiting time is a busy period started by V work including only arrivals from higher priority classes. This viewpoint will be central to the analysis of many of the other priority based policies in this section.

There are a number of interesting observations that can be made about (3.14). First of all notice that it is possible for high priority classes to remain stable at loads much larger than 1. Thus, prioritization provides an insulation from overload for high priority jobs. Second, it is interesting to note the improvements over $E[T]^{FCFS}$ that are possible from very simple priority schemes. For example, if we consider a 2-class system, we can see that

$$\begin{aligned} E[T] &= \frac{\lambda_1}{\lambda} E[W_1] + \frac{\lambda_2}{\lambda} E[W_2] \\ &= E[W]^{FCFS} \left(\frac{1 - \lambda_1 E[X]}{1 - \lambda_1 E[X_1]} \right) \end{aligned} \tag{3.15}$$

Examining the form of this equation, it becomes clear that whenever $E[X_1] < E[X]$ prioritization provides a smaller response time than FCFS. Further, the difference can be dramatic if $E[X_1]$ is much less than $E[X]$. This observation is a key motivation for the heuristic of "prioritizing small jobs."

Note that both higher moments and the transform of the waiting time distribution of non-preemptive

priority queues have been derived in the literature. But since the derivations are involved and they are not central to this thesis, we refer the interested reader to [61, 222] for the details.

There are many variations of the non-preemptive priority queue. Two of the most common are *non-preemptive threshold based policies* and *non-preemptive Shortest-Job-First* (SJF). We will provide an overview of each of these policies in the next two sections.

3.2.2.1 Non-preemptive threshold based policies

A non-preemptive threshold based policy is simply a non-preemptive priority queue where the priority classes are determined using job size thresholds. In particular, given thresholds $0 = t_0 < t_1 < \ldots < t_n = \infty$ an arriving job of size x is assigned priority i if $x \in [t_{i-1}, t_i)$.

The motivation for introducing threshold based policies follows immediately from our discussion in the previous section. In particular, from (3.15) we see that the mean response time of a non-preemptive priority queue is lowest when the high priority queues have the smallest possible mean service demand. Thus, if we hope to optimize $E[T]$, we need to minimize the mean service demands of the high priority class – which is exactly what happens under threshold based policies.

The analysis of the waiting time under non-preemptive threshold based policies follows immediately from our discussion of general non-preemptive priority queues. Thus, it is possible to obtain the transform and the moments of waiting time for these policies. In particular, we have that under an n class non-preemptive threshold based policy, denoted NP_n, the mean waiting time of a class i job is as follows

$$E[W_i]^{NP_n} = \frac{\lambda E[X^2]}{2(1-\rho(t_{i-1}))(1-\rho(t_i))},$$

where $\rho(t_i) = \lambda \int_0^{t_i} sf(s)ds$.

From the above, we can also calculate the overall response time for the system as follows:

$$E[T] = E[X] + \sum_{i=1}^{n} (\overline{F}(t_{i-1}) - \overline{F}(t_i))E[W_i]$$

Despite the fact that this formula is easy to write down, it is hard to completely understand. Though it is easy to see that $E[T] < \infty$ only when $E[X^2] < \infty$, even understanding simple questions such as how $E[T]$ behaves as a function of load and how $E[T]$ changes as n grows is difficult. We will spend the rest of this section investigating these questions.

If the thresholds are constant as load grows, $E[T] = \Theta(1/(1-\rho))$ as $\rho \to 1$, which is the same behavior we have already seen for FCFS, PS, and the other simple policies discussed so far. But, clearly the thresholds should vary depending on load. In fact, by choosing the thresholds for each load optimally, it is possible to provide dramatic improvements in $E[T]$ over PS and FCFS for high loads.

In particular, Bansal and Gamarnik [24] have shown that using any number $n \geq 2$ priority classes in an M/M/1 queue is enough to guarantee that $E[T]^{NP_n}$ grows much more slowly than $\Theta(1/(1-\rho))$ and thus provides huge improvements in $E[T]$ in heavy traffic.

Theorem 3.3
In an M/M/1 for any $n \geq 2$

$$E[T]^{NP_n} = \Theta\left(\frac{1}{(1-\rho)\log\left(\frac{1}{1-\rho}\right)}\right)$$

Further, non-preemptive threshold based policies can behave even better under other service distributions. In particular, we will prove a novel result that characterizes the growth rate of $E[T]$ under Pareto distributions. Before stating the theorem we need to introduce some notation. Define $g_i(\alpha)$ as follows:

$$\begin{aligned} g_{-1}(\alpha) &= 0 \\ g_0(\alpha) &= 1 \\ g_i(\alpha) &= 1 + (\alpha-1)g_{i-1}(\alpha) \text{ for } i \geq 1 \end{aligned}$$

Thus, note that

$$\frac{g_{i-1}(\alpha)}{g_i(\alpha)} = \frac{g_{i-1}(\alpha)}{1+(\alpha-1)g_{i-1}(\alpha)} = \frac{1}{\alpha - 1 + \frac{1}{g_{i-1}(\alpha)}}$$

Now we are ready to state the theorem.

Theorem 3.4
Let the service distribution be $Pareto(\alpha)$ with $\alpha > 2$ so $E[X^2] < \infty$. Consider a NP_{n+1} policy with thresholds $0 = t_0 < t_1 < \ldots < t_n < t_{n+1} = \infty$ and define t_i for $i = 1, \ldots, n$ such that $\overline{F}(t_i) = (1-\rho)^{a_i}$ with

$$\frac{a_{n-i}}{\alpha} = g_i(\alpha)\epsilon_{n+1} - g_{i-1}(\alpha)$$

where

$$\epsilon_{n+1} = \frac{g_{n-1}(\alpha)}{g_n(\alpha)} = \frac{1}{(\alpha-1) + \frac{1}{g_{n-1}(\alpha)}}$$

Then

$$E[T]^{NP_{n+1}} = \Theta\left(\frac{1}{(1-\rho)^{1-\epsilon_{n+1}}}\right)$$

Clearly, the growth rate of $E[T]$ as a function of ρ is much slower under a Pareto distribution (Theorem Theorem 3.4) than it is under an M/M/1 distribution (Theorem 3.3). Further, Theorem 3.4 provides an interesting contrast to Theorem 3.3 in terms of the impact of the number of priority classes used by the scheduler. While in the M/M/1 case, using more than 2 priority classes provided no improvement in the growth rate of $E[T]$; in the case of a Pareto service distribution, we see an improvement from each priority class added, since ϵ_n increases with n.

Proof of Theorem 3.4. We first write $E[T]^{NP_{n+1}}$ as follows:

$$E[T]^{NP_{n+1}} = E[X] + \sum_{i=1}^{n+1} \frac{\lambda E[X^2](\overline{F}(t_{i-1}) - \overline{F}(t_i))}{2(1-\rho(t_{i-1}))(1-\rho(t_i))}$$

Now, we note that by choosing $\overline{F}(t_i) = (1-\rho)^{a_i}$ for $i = 1, \ldots, n$ we have that $k/t_i = (1-\rho)^{a_i/\alpha}$

(since we are considering a Pareto distribution), which gives that

$$\begin{aligned} 1-\rho(t_i) &= 1-\rho+\rho\left(\frac{k}{t_i}\right)^{\alpha-1} \\ &= 1-\rho+\rho(1-\rho)^{(\alpha-1)a_i/\alpha} \\ &\sim (1-\rho)^{(\alpha-1)a_i/\alpha} \text{ as } \rho\to 1 \end{aligned}$$

Further, $(1-\rho(t_0))=1$ and $(1-\rho(t_{n+1}))=(1-\rho)$.

Continuing, we now see that as $\rho \to 1$

$$\begin{aligned} & E[T]^{NP_{n+1}} \\ = \; & E[X]+\sum_{i=1}^{n+1}\frac{\lambda E[X^2](\overline{F}(t_{i-1})-\overline{F}(t_i))}{2(1-\rho(t_{i-1}))(1-\rho(t_i))} \\ \sim \; & \frac{\lambda E[X^2]}{2}\left(\frac{1-(1-\rho)^{a_1}}{(1-\rho)^{(\alpha-1)a_1/\alpha}}+\left(\sum_{i=2}^{n}\frac{(1-\rho)^{a_{i-1}}-(1-\rho)^{a_i}}{(1-\rho)^{(\alpha-1)a_{i-1}/\alpha}(1-\rho)^{(\alpha-1)a_i/\alpha}}\right)+\frac{(1-\rho)^{a_n}}{(1-\rho)^{(\alpha-1)a_n/\alpha}(1-\rho)}\right) \\ \sim \; & \frac{\lambda E[X^2]}{2}\left((1-\rho)^{-(\alpha-1)a_1/\alpha}+\left(\sum_{i=2}^{n}(1-\rho)^{a_{i-1}-(\alpha-1)a_{i-1}/\alpha-(\alpha-1)a_i/\alpha}\right)+(1-\rho)^{a_n-1-(\alpha-1)a_n/\alpha}\right) \\ = \; & \frac{\lambda E[X^2]}{2}\left((1-\rho)^{-(\alpha-1)a_1/\alpha}+\left(\sum_{i=2}^{n}(1-\rho)^{a_{i-1}/\alpha-(\alpha-1)a_i/\alpha}\right)+(1-\rho)^{-1+a_n/\alpha}\right) \end{aligned}$$

Note that the best heavy traffic behavior occurs when all the exponents of $(1-\rho)$ are equal (if they are not balanced, a local improvement can be made). We will show that, when all of the exponents are equal, they are equal to $-1+\epsilon_{n+1}$, from which the theorem will follow.

Plug $\frac{a_{n-i}}{\alpha} = g_i(\alpha)\epsilon_{n+1} - g_{i-1}(\alpha)$ into each of the exponents of $(1-\rho)$ above. Note that by definition of $g_i(\alpha)$, we have $(\alpha-1)g_i(\alpha)-g_{i-1}(\alpha)=-1$ for all i. We start with the constraints for a_i, $i>1$.

$$\begin{aligned} -1+\frac{a_n}{\alpha} &= -1+g_0(\alpha)\epsilon_{n+1}-g_{-1}(\alpha) \\ &= -1+\epsilon_{n+1} \\ \frac{a_{i-1}}{\alpha}-(\alpha-1)\frac{a_i}{\alpha} &= g_{i-1}(\alpha)\epsilon_{n+1}-g_{i-2}(\alpha)-(\alpha-1)(g_i(\alpha)\epsilon_{n+1}-g_{i-1}(\alpha)) \\ &= [(\alpha-1)g_{i-1}(\alpha)-g_{i-2}(\alpha)]-[(\alpha-1)g_i(\alpha)-g_{i-1}(\alpha)]\,\epsilon_{n+1} \\ &= -1+\epsilon_{n+1} \end{aligned}$$

Next, we consider the final constraint on a_1. Here, we will need to use the fact that $\epsilon_{n+1} = g_{n-1}(\alpha)/g_n(\alpha)$.

$$\begin{aligned} -(\alpha-1)\frac{a_1}{\alpha} &= -(\alpha-1)g_{n-1}(\alpha)\epsilon_{n+1}+(\alpha-1)g_{n-2}(\alpha) \\ &= -1+\epsilon_{n+1}+[1+(\alpha-1)g_{n-2}(\alpha)]-[1+(\alpha-1)g_{n-1}(\alpha)]\,\epsilon_{n+1} \\ &= -1+\epsilon_{n+1}+g_{n-1}(\alpha)-g_n(\alpha)\epsilon_{n+1} \\ &= -1+\epsilon_{n+1} \end{aligned}$$

□

3.2.2.2 Non-preemptive Shortest-Job-First (SJF)

If our goal is truly to optimize $E[T]$ using a non-preemptive threshold based policy, we have seen that we want to have as many priority classes as possible. If we take non-preemptive threshold based policies to the extreme and add thresholds until the thresholds become arbitrarily close, the resulting policy is SJF. Under SJF, the shortest job in the queue is given non-preemptive priority. Thus, at every completion instant, the job in the queue with the smallest service time is given service, and this service is not interrupted until the job completes.

Since SJF can be viewed as the limiting case of a non-preemptive threshold based policy, it is clear that the moments and transform of the response time of SJF can be obtained by taking the appropriate limits of NP_n. Assuming the service distribution is continuous, this limit results in the following formulas for the first few moments and transform of conditional response time under SJF:

$$\begin{aligned}
E[T(x)]^{SJF} &= x + \frac{\lambda E[X^2]}{2(1-\rho(x))^2} \\
Var[T(x)]^{SJF} &= \frac{\lambda E[X^3]}{3(1-\rho(x))^3} + \frac{\lambda^2 m_2(x) E[X^2]}{(1-\rho(x))^4} - \frac{\lambda^2 E[X^2]^2}{4(1-\rho(x))^4} \\
L_{T(x)}(s)^{SJF} &= \frac{\mathcal{L}_X(s)}{s}\left((1-\rho)(s+\lambda F(x)-\lambda F(x)\mathcal{L}_{B_x}(s))\right. \\
&\quad \left. +\lambda \overline{F}(x)(1-\mathcal{L}_{X_{>x}}(s+\lambda F(x)-\lambda F(x)\mathcal{L}_{B_x}(s)))\right)
\end{aligned}$$

where $X_{>x}$ has c.d.f. $F(x)/\overline{F}(x)$.

Before moving to the overall response time of SJF, let us take a second to contrast the conditional response time of SJF with that of non-preemptive threshold based policies. Notice that the small jobs in each class of the non-preemptive threshold based policy have larger mean response times than they do under SJF while the large jobs sizes in each class have smaller response times than they do under SJF. Thus, $E[T(x)]^{SJF}$ is not uniformly better than $E[T(x)]^{NP_n}$, though SJF provides a smaller overall mean response time.

Let us now move forward and discuss the overall mean response time of SJF. Given $E[T(x)]^{SJF}$ it is easy to calculate $E[T]^{SJF}$:

$$E[T]^{SJF} = E[X] + \frac{\lambda E[X^2]}{2}\int_0^1 \frac{dF(x)}{(1-\rho(x))^2}$$

As with non-preemptive threshold based policies, despite the ease with which this formula can be written, it is difficult to understand the behavior of SJF as a function of ρ. This is a trend we will see throughout this chapter. However, very recently, a few results that characterize the growth rate of SJF as a function of ρ under certain distributions have emerged. In particular, Bansal and Gamarnik [24] have characterized the behavior of SJF under Pareto distributions and Bansal & Wierman [26] have characterized the behavior of SJF under Exponential distributions. We summarize these results in the following proposition.

Proposition 3.5
In an M/M/1,

$$E[T]^{SJF} = \Theta\left(\frac{1}{(1-\rho)\log\left(\frac{1}{1-\rho}\right)}\right)$$

Further, in an M/GI/1 where $X \sim Pareto(\alpha)$ with $\alpha > 2$ so that $E[X^2] < \infty$,

$$E[T]^{SJF} = \Theta\left(\frac{1}{(1-\rho)^{\frac{\alpha-2}{\alpha-1}}}\right)$$

It interesting to contrast Theorem 3.5 with Theorems 3.3 and 3.4, which characterize the growth rate of $E[T]$ under non-preemptive threshold based policies. In the M/M/1 setting, we see that $E[T]^{SJF} = \Theta(E[T]^{NP_n})$, which is quite surprising since it says using two priority classes provides the same benefits as using an infinite number of priority classes. However, the case of Pareto service distributions provides a different picture. Under Pareto distributions we have that

$$E[T]^{SJF} = \Theta\left(\frac{1}{(1-\rho)^{\frac{\alpha-2}{\alpha-1}}}\right) = \Theta\left(\frac{1}{(1-\rho)^{1-\frac{1}{\alpha-1}}}\right)$$

and

$$\begin{aligned} E[T]^{NP_{n+1}} &= \Theta\left(\frac{1}{(1-\rho)^{1-\frac{g_{n-1}(\alpha)}{g_n(\alpha)}}}\right) = \Theta\left(\frac{1}{(1-\rho)^{-1+\frac{1}{\alpha-1+1/g_{n-1}(\alpha)}}}\right) \\ &\to \Theta\left(\frac{1}{(1-\rho)^{1+\frac{1}{\alpha-1}}}\right) \quad \text{as } n \to \infty \end{aligned}$$

Thus, $E[T]^{NP_n}$ approaches the growth rate of $E[T]^{SJF}$ as $n \to \infty$, but NP_n does not achieve the growth rate of SJF for any finite n.

3.2.3 Preemptive priority queues

Preemptive priority queues are very similar to the non-preemptive priority queues we just discussed. Again, jobs are assumed to arrive tagged with a priority class $c = 1, 2, \ldots$, and jobs within each class are served in FCFS order. The difference is that a job of priority c can only receive service when there are no jobs with priority $< c$ in the system, i.e. a job of priority c in service is preempted whenever an arrival of priority $< c$ occurs.

The analysis of preemptive priority queues can be performed in a manner parallel to the analysis of non-preemptive priority queues (see [120, 222] for example). However, instead of mimicking the analysis of the non-preemptive priority queues here, we will present a more intuitive analysis that is only possible in the preemptive case. To accomplish this, let us break up the response time of a tagged customer with priority c into the residence time and the waiting time of the job.

To analyze the residence time of a class c job, denoted R_c, notice that all arrivals with priority $< c$

receive preemptive priority over the tagged job. Thus, the residence time of the tagged job is simply a busy period started by the tagged job including all arrivals of jobs in classes $1 \ldots, c-1$, which we will denote $B_{\leq c-1}(X_c)$.

Similarly, we can view the waiting time as a busy period including all arrivals from classes $1, \ldots, c-1$ started by the work in the system from classes $1, \ldots, c$, denoted $B_{\leq c-1}(Q_{\leq c})$. All that remains is to determine the distribution of $Q_{\leq c}$. This can be accomplished by viewing $Q_{\leq c}$ as the workload in a transformed system where only arrivals from classes $1, \ldots, c$ are included. Noting that this transformed system is still work conserving (by definition of a preemptive priority queue), we can see that $Q_{\leq c}$ is the stationary workload in a FCFS queue with arrival rate $\lambda_{\leq c} = \sum_{i=1}^{c} \lambda_i$ and service distribution $X_{\leq c} = X_i$ with probability $\lambda_i/\lambda_{\leq c}$ for $i \leq c$. Thus, we can write $E[T_c]$ as follows:

$$E[T_c] = E[B_{\leq c-1}(X_c) + B_{\leq c-1}(Q_{\leq c})] \tag{3.16}$$

$$= \frac{E[X_c]}{1 - \sum_{i=1}^{c-1} \rho_i} + \frac{\sum_{i=1}^{c} \lambda_i E[X_i^2]}{2(1 - \sum_{i=1}^{c-1} \rho_i)(1 - \sum_{i=1}^{c} \rho_i)} \tag{3.17}$$

Similarly, it is easy to calculate higher moments of T_c and the transform T_c using this formulation.

Notice the contrasts between the form of (3.17) for $E[T_c]$ in preemptive priority queues and (3.14) for $E[W_c]$ in non-preemptive priority queues. Clearly, the waiting time for class c jobs is larger in the non-preemptive case because it includes the second moment of the full service distribution. However, under non-preemptive priority queues a job cannot be interrupted once it begins service. Thus, the residence time of class c jobs in the non-preemptive case is much lower than the residence time in the preemptive case.

As in the non-preemptive case, there are two common variations of the preemptive priority queue: preemptive threshold based policies and Preemptive-Shortest-Job-First (PSJF). In the next two sections we will provide an overview of each of these policies.

3.2.3.1 Preemptive threshold based policies

Paralleling the definition of non-preemptive threshold based policies, a preemptive threshold based policy P_n is simply a preemptive priority queue where the n priority classes are determined using job size cutoffs. That is, given thresholds $t_0 < t_1 < \ldots < t_n = \infty$ an arriving job of size x is assigned priority c if $x \in [t_{c-1}, t_c)$. Thus, applying the results from the previous section, it is easy to see that

$$E[T_c]^{P_n} = \frac{E[X_c]}{1 - \rho(t_{c-1})} + \frac{\lambda m_2(t_c)}{2(1 - \rho(t_{c-1}))(1 - \rho(t_c))}$$

Similarly, the transform and higher moments of response time follow immediately from the corresponding results for preemptive priority queues.

It is interesting at this point to contrast the behavior of preemptive threshold based policies with the behavior of the non-preemptive threshold based policies we previously introduced. In particular, it is natural to think that response times under preemptive threshold based policies should be lower than response times under non-preemptive threshold based policies. However, we can see from comparing the formulas for $E[T_c]$ in each case that the waiting time is larger in the non-preemptive case, but the residence time is larger in the preemptive case. To see how these two terms trade off, let us consider the special case of 2 class disciplines where the thresholds are the same under both the preemptive and the non-preemptive policies. Then, the increase in $E[R]$ for class 2 jobs in moving from the non-preemptive policy to the preemptive

policy is $\frac{\rho_1 E[X_2]}{1-\rho_1}$. In contrast, the decrease in $E[W]$ experienced by class 1 jobs is $\frac{\lambda_2 E[X_2^2]}{2(1-\rho_1)}$. Thus, the preemptive threshold based policy has smaller $E[T]$ only when

$$\frac{\lambda_2 \rho_1 E[X_2]}{1-\rho_1} - \frac{\lambda_1 \lambda_2 E[X_2^2]}{2(1-\rho_1)} < 0$$

which gives that

$$E[T]^{P_n} < E[T]^{NP_n} \quad \Leftrightarrow \quad E[X_1] < \frac{E[X_2^2]}{2E[X_2]}$$

This final condition is quite illustrative. Recall that the RHS of the equation is simply the mean excess of a class 2 job. Thus, preemptive threshold based policies are better only when the expected remaining size of a class 2 job that is being preempted is larger than the expected size of the class 1 job that is preempting it.

The above comparison provides a first step towards understanding the behavior of the mean response time under preemptive threshold based policies. In particular, in the M/M/1 we have that $E[T]^{P_2} = E[T]^{NP_2}$ which gives

$$E[T]^{P_2} = \Theta\left(\frac{1}{(1-\rho)\log\left(\frac{1}{1-\rho}\right)}\right)$$

Further, we can prove a novel result that shows that $E[T]^{P_n} = \Theta(E[T]^{NP_n})$ in general.

Theorem 3.6
In an M/GI/1 queue, $E[T]^{P_n} = \Theta(E[T]^{NP_n})$ as $\rho \to 1$.

Note that Theorem 3.6 combined with Theorems 3.3 and 3.4 immediately gives us the growth rate of $E[T]^{P_n}$ with ρ under the Exponential and Pareto service distributions.

Proof. Let us begin by observing that both preemptive and non-preemptive threshold based policies have $E[T] < \infty$ only when $E[X^2] < \infty$ and $\rho < 1$ since in both cases the waiting time of the nth class has an $E[X^2]$ term. Thus, we need only consider service distributions where $E[X^2] < \infty$.

We continue by bounding $E[T_i]^{P_n}$ in terms of $E[T_i]^{NP_n}$.

$$\begin{aligned}
E[T_i]^{P_n} &= \frac{E[X_i]}{1-\rho(t_{i-1})} + \frac{\lambda m_2(t_i)}{2(1-\rho(t_{i-1}))(1-\rho(t_1))} \\
&= E[X_i] + \frac{\lambda m_2(t_i) + E[X_i]\rho(t_{i-1})(1-\rho(t_1))}{2(1-\rho(t_{i-1}))(1-\rho(t_i))} \\
&= E[X_i] + \left(\frac{m_2(t_i) + E[X_i] m_1(t_{i-1})(1-\rho(c_i))}{E[X^2]}\right) E[W_i]^{NP_n} \\
&\leq E[X_i] + 2E[W_i]^{NP_n} \\
&\leq 2E[T_i]^{NP_n}
\end{aligned}$$

Similarly, it is easy to see that

$$E[T_i]^{NP_n} \leq \frac{E[X^2]}{m_2(t_i)} E[T_i]^{P_n}$$

which gives that

$$\frac{m_2(t_1)}{E[X^2]}E[T]^{NP_n} \leq E[T]^{P_n} \leq 2E[T]^{NP_n}$$

Thus, as long as $t_1 > \epsilon$ as $\rho \to 1$, $E[T]^{NP_n}$ and $E[T]^{P_n}$ are always within a constant factor. But, if $t_1 \to 0$, that would mean that $\rho(t_1) \to 0$ since the increase in λ is bounded. Thus, the limiting policy would be an $n-1$ class policy, which is a contradiction since $E[T]^{P_n} \leq E[T]^{P_{n-1}}$ for the optimal choices of thresholds. □

3.2.3.2 Preemptive-Shortest-Job-First (PSJF)

As in the non-preemptive case, increasing the number of priority classes in preemptive threshold based policies decreases $E[T]$. If we take preemptive threshold based policies to the extreme and add thresholds until they become arbitrarily close, the resulting policy is PSJF. Under PSJF the job in the system with the smallest original size is always receiving service. Thus, a job at the server is always interrupted when a smaller job arrives. As in the non-preemptive case, taking the appropriate limits of P_n yields $E[T(x)]^{PSJF}$ when the service distribution is continuous:

$$E[T(x)]^{PSJF} = \frac{x}{1-\rho(x)} + \frac{\lambda m_2(x)}{2(1-\rho(x))^2}$$

The first term in this equation is the residence time of PSJF, $R(x)^{PSJF}$, and the second term is the waiting time of PSJF, $W(x)^{PSJF}$.

Though it is easy to obtain higher moments and the transform of $T(x)^{PSJF}$ by taking limits of threshold based policies, it is perhaps more illustrative to derive the statistics of $T(x)^{PSJF}$ directly. To this end, let us first consider $R(x)^{PSJF}$. Once a tagged job of size x, j_x, begins to receive service the only jobs that can receive higher priority than j_x are new arrivals with size $< x$. Thus, j_x will complete at the end of a busy period started by x work including only arrivals of size $< x$, denoted $B_x(x)$. Further, we can view $W(x)^{PSJF}$ as a busy period started by the work in the system at the arrival of j_x having original size $< x$ including only new arrivals with size $< x$. Thus, $W(x)^{PSJF} = B_x(Q_x^{PSJF})$ where Q_x^{PSJF} is the stationary workload of a system where only arrivals size $< x$ are considered. Combining all these statements and using the linearity of busy periods, we obtain

$$T(x)^{PSJF} \stackrel{\mathrm{d}}{=} B_x(x + Q_x^{PSJF}) \stackrel{\mathrm{d}}{=} B_x(x) + B_x(Q_x^{PSJF})$$

from which the moments and transform of PSJF follow directly. Let us just state the $Var[T(x)]$ and the transform, since these will be of use later in the thesis.

$$\begin{aligned} Var[T(x)]^{PSJF} &= \frac{\lambda x m_2(x)}{(1-\rho(x))^3} + \frac{\lambda m_3(x)}{3(1-\rho(x))^3} + \frac{3}{4}\left(\frac{\lambda m_2(x)}{(1-\rho(x))^2}\right)^2 \\ \mathcal{L}_{T(x)}(s)^{PSJF} &= \frac{1}{s}(1-\rho(x))(s+\lambda F(x) - \lambda F(x)\mathcal{L}_{B_x}(s))e^{-x(s+\lambda F(x)-\lambda F(x)\mathcal{L}_{B_x}(s))} \end{aligned}$$

So far, we have only discussed the conditional response time under PSJF, $T(x)^{PSJF}$, but typically the metric of interest is actually the overall response time, specifically $E[T]^{PSJF}$. Clearly $E[T]^{PSJF}$ can be

computed by deconditioning $E[T(x)]$ as follows:

$$E[T]^{PSJF} = \int_0^1 E[T(x)]dF(x) = \int_0^\infty \left(\frac{x}{1-\rho(x)} + \frac{\lambda m_2(x)}{2(1-\rho(x))^2}\right) f(x)dx$$

However, this form is difficult to work with. In particular, $E[W]^{PSJF}$ typically needs to be calculated numerically, and further the above equation provides little intuition about the behavior of $E[T]^{PSJF}$ as $\rho \to 1$ or as job size variability grows. To provide such intuition, Wierman, Harchol-Balter, and Osogami prove the following simple bounds on $E[T]^{PSJF}$ in [241].

Theorem 3.7
Consider an M/GI/1 PSJF *queue. Let K satisfy $\lambda m_2(x) \leq Kx\rho(x)$ and X_1 and X_2 be i.i.d. service demands. Then*

$$E[X]\left(\frac{\rho E[(X_1 \wedge X_2)^2]}{4E[X]^2} + \frac{1}{\rho}\log\left(\frac{1}{1-\rho}\right)\right) \leq E[T]^{PSJF} \leq E[X]\left(\frac{K/2}{1-\rho} + \left(\frac{1-K/2}{\rho}\right)\log\left(\frac{1}{1-\rho}\right)\right)$$

Notice that Theorem 3.7 is stated in terms of a parameter K, where K is such that $\lambda m_2(x) \leq Kx\rho(x)$. Clearly $K \leq 1$ for all service distributions, but it is not immediately clear how K behaves under specific distributions. In Theorem 3.8 we illustrate how this constant may be set under common distributions. For example, we show that the constant K may be set at $\frac{2}{3}$ when the service distribution is decreasing. Further, in more generality, it defines K in a way that is highly tied to the tail properties of $f(x)$.

Theorem 3.8
Let i be a positive integer. Define j such that $x^j f(x)$ is decreasing and $j < i+1$. Then,

$$m_{i+1}(x) \leq \left(\frac{i-j+1}{i-j+2}\right) x m_i(x)$$

Since the proof of Theorem 3.8 is tangential to the current discussion, we defer it to the end of the section.

Proof of Theorem 3.7. First, we calculate $E[R]^{PSJF}$ exactly. Note that $\frac{d}{dx}\rho(x) = \lambda x f(x)$.

$$\begin{aligned} E[R]^{PSJF} &= \frac{1}{\lambda}\int_0^\infty \frac{\lambda x f(x)}{1-\rho(x)}dx \\ &= -\frac{1}{\lambda}\log(1-\rho) \end{aligned}$$

Next, we can calculate an upper bound on the waiting time of PSJF:

$$\begin{aligned} E[W]^{PSJF} &\leq \frac{K}{2\lambda}\int_0^\infty \frac{\lambda x f(x)\rho(x)}{(1-\rho(x))^2}dx \\ &= \frac{K}{2\lambda}\left(\frac{\rho}{1-\rho} + \log(1-\rho)\right) \end{aligned}$$

Finally, to prove the lower bound on waiting time, recall that the p.d.f. of $X_1 \wedge X_2$ is $f_{min}(x) =$

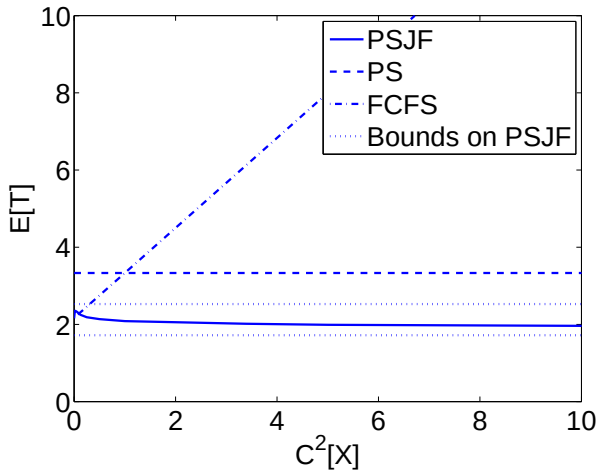


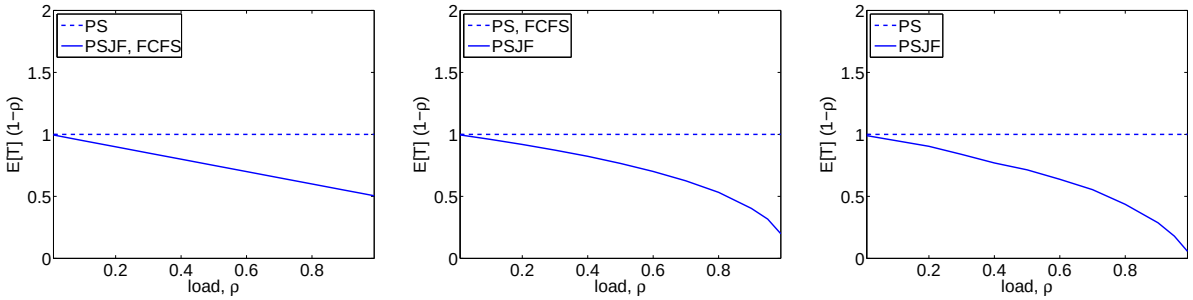
Figure 3.2: *This figure shows the mean response time of* PSJF, PS, *and* FCFS *as a function of the variability of the service distribution ($C^2[X]$). The load is 0.7 and the service distribution is a Weibull with mean 1 in each case. The bounds shown are in Theorem 3.7.*

$2f(x)\overline{F}(x)$. Thus

$$\begin{aligned} E[W]^{PSJF} &\geq \frac{\lambda}{2}\int_0^\infty f(x)\int_0^x t^2 f(t)dtdx \\ &= \frac{\lambda}{4}\int_0^\infty 2t^2 f(t)\overline{F}(t)dt \\ &= \frac{\lambda}{4}E[(X_1 \wedge X_2)^2] \end{aligned}$$

□

Though the bounds in Theorem 3.7 do not completely characterize the behavior of $E[T]^{PSJF}$, they already provide some useful information. First and foremost, notice that $E[T]^{PSJF} < \infty$ even when $E[X^2] = \infty$. This is a huge contrast with the behavior of the other preemptive and non-preemptive threshold based policies we have studied, under which $E[T] = \infty$ if $E[X^2] = \infty$. Further, since the bounds on $E[T]^{PSJF}$ in Theorem 3.7 are completely independent of $E[X^2]$, they indicate that PSJF is "nearly" insensitive to variability in the service distribution. In fact, the bounds characterize the degree to which variability can affect $E[T]^{PSJF}$. In addition, the bounds are as tight as possible without making use of the variability in the service distribution: we see that the upper bound becomes tight when the service distribution has low variability and the lower bound becomes tight when the service distribution is highly variable (Theorem 3.9). It is interesting to note that variability tends to improve mean response time under PSJF. This is a result of the fact that having a big distinction between large and small jobs (as is true when variability is high) is what allows the heuristic of "prioritizing small jobs" to be effective. This is in stark contrast to



(a) Deterministic (b) Exponential (c) Pareto ($\alpha = 1.2$)

Figure 3.3: *This figure illustrates the behavior of mean response time under* PSJF, FCFS, *and* PS *as a function of load.*[1] *The mean response time is scaled by* $(1-\rho)$ *in order to highlight differences between the policies. In (a) job sizes are deterministic, in (b) job sizes are exponential, and in (c) job sizes are Pareto with* $\alpha = 1.2$ *(thus they have* $E[X^2] = \infty$*). In all cases,* $E[X] = 1$*. Note that* FCFS *is not included in (c) because* $E[T]^{FCFS} = \infty$ *in this case.*

FCFS, for which variability is extremely detrimental.

Beyond the new insights into the effect of variability on $E[T]^{PSJF}$, Theorem 3.7 provides the beginnings of an understanding of the behavior of $E[T]^{PSJF}$ as a function of load. In particular, the upper bound is $\Theta(1/(1-\rho))$ as $\rho \to 1$, which matches the growth rate of PS and FCFS, and the lower bound grows like $\Theta(\log(1/(1-\rho))$, which is a huge improvement over PS and FCFS. It is not immediately clear that the the lower bound is ever achieved, but Figure 3.2 hints that it is, and it has recently been proven by Bansal and Gamarnik [24] that the growth rate of PSJF under a Pareto service distribution matches that in the lower bound. Further, Wierman et. al. also characterize the growth rate of PSJF under an Exponential distribution [26].

Theorem 3.9

In an M/M/1 queue

$$E[T]^{PSJF} = \Theta\left(\frac{1}{(1-\rho)\log\left(\frac{1}{1-\rho}\right)}\right)$$

Further in an M/GI/1 queue with $X \sim Pareto(\alpha)$ *where* $\alpha > 1$ *so that* $E[X] < \infty$*,*

$$E[T]^{PSJF} = \begin{cases} \Theta\left(\log\left(\frac{1}{1-\rho}\right)\right), & \text{if } \alpha < 2 \\ \Theta\left(\log^2\left(\frac{1}{1-\rho}\right)\right), & \text{if } \alpha = 2 \\ \Theta\left((1-\rho)^{-\frac{\alpha-2}{\alpha-1}}\right), & \text{if } \alpha > 2. \end{cases}$$

Contrasting Theorem 3.9 with the parallel results for SJF, preemptive threshold based policies, and non-preemptive threshold based policies is quite interesting. The first thing to notice is that the growth rate of all of these policies is the same in the M/M/1 queue. Further, under a Pareto distribution, as long as $E[X^2] < \infty$ the growth rate of PSJF matches that of SJF, which is the limiting case of both P_n and NP_n

as $n \to \infty$. However, the case when the service distribution is Pareto and $E[X^2] = \infty$ is when the growth rate of $E[T]^{PSJF}$ is a huge improvement over the other priority based policies we have discussed so far. Thus, the best case for PSJF corresponds to the worst case for SJF and threshold based policies.

We end the section by providing a proof of Theorem 3.8.

Proof of Theorem 3.8.

First, we observe the following equality:

$$\begin{aligned}
\int_{t=0}^{x} m_i(t)dt &= \int_{t=0}^{x}\int_{s=0}^{t} s^i f(s)dsdt \\
&= \int_{s=0}^{x} s^i f(s)\int_{t=s}^{x} dtds \\
&= \int_{s=0}^{x} (x-s)s^i f(s)ds \\
&= xm_i(x) - m_{i+1}(x) \qquad (3.18)
\end{aligned}$$

We will now use this relation to bound $m_{i+1}(x)$ in terms of $m_i(x)$ by first bounding $\int_0^x m_i(t)dt$. Remember, by assumption we know that $s^j f(s)$ is decreasing for some j such that $j < i+1$.

$$\begin{aligned}
\int_{t=0}^{x} m_i(t)dt &= \int_{t=0}^{x}\int_{s=0}^{t} s^i f(s)dsdt \\
&\geq \int_{t=0}^{x} t^j f(t)\int_{s=0}^{t} s^{i-j}dsdt \\
&= \frac{1}{i-j+1}\int_{t=0}^{x} t^j f(t)t^{i-j+1}dt \\
&= \frac{1}{i-j+1}m_{i+1}(x) \qquad (3.19)
\end{aligned}$$

In this chain of equalities, the inequality follows directly from the assumption that $s^j f(s)$ is decreasing.

Finally, combining Equation 3.18 and Equation 3.19, we can complete the proof.

$$\begin{aligned}
xm_i(x) - m_{i+1}(x) &\geq \frac{1}{i-j+1}m_{i+1}(x) \\
\left(\frac{i-j+1}{i-j+2}\right)xm_i(x) &\geq m_{i+1}(x)
\end{aligned}$$

□

3.2.4 Shortest-Remaining-Processing-Time-First (SRPT)

We now move to probably the most well known priority based policy: SRPT. Under SRPT, at every instance, the job with the smallest remaining service time is scheduled. So, SRPT differs from the priority

based policies we have discussed so far in that the priority of a job actually increases while the job is in the system, i.e. as the remaining size of the job decreases. In this way SRPT greedily tries to minimize the number in system by always working on the job that can be finished the quickest. In the preempt-resume setting, this greedy approach is good enough to minimize the number in the system, and thus $E[T]$, regardless of the arrival and service processes because any scheduling decision can be reversed without penalty if a more attractive (smaller) candidate arrives [201]. Because of this optimality, SRPT has received a large amount of attention in the literature.

As we have for previous priority based policies, we will begin our discussion of SRPT by analyzing the conditional response time and then we will exploit this analysis in order to study the overall mean response time of SRPT.

3.2.4.1 Deriving the conditional response time of SRPT

Beginning as early as 1966, Schrage and Miller had already analyzed the moments and the transform of $T(x)$ under SRPT [202]. The resulting form of $T(x)$ is quite complex, and best understood by breaking it into pieces that correspond to the residence time and waiting time.

We start with the residence time. Recall that the residence time of a job is the time from when it first receives service until it completes. Once a tagged job j begins to receive service, it has higher priority than all other jobs in the system, and thus, only new arrivals with smaller original size than the remaining size of j can preempt j. Further, once one such job preempts j when j has remaining size t, the job starts a period where all arriving jobs with size $< t$ receive higher priority than j. This is clearly a busy period, B_t. So, the time it takes j to move from having remaining size t to having remaining size $t - dt$ is simply $B_t(dt)$. Integrating over t, we can obtain the moments and transform of $R(x)^{SRPT}$

$$\begin{aligned} E[R(x)]^{SRPT} &= \int_0^x E[B_t(dt)] = \int_0^x \frac{dt}{1-\rho(t)} \\ Var[R(x)]^{SRPT} &= \int_0^x Var[B_t(dt)] = \int_0^x \frac{\lambda m_2(t)}{(1-\rho(t))^3} dt \\ \mathcal{L}_{R(x)}(s)^{SRPT} &= e^{-\int_0^x \left(s+\lambda F(y)-\lambda F(y)\mathcal{L}_{B_y}(s)\right)dy} \end{aligned}$$

Moving to the analysis of $W(x)^{SRPT}$, there are two approaches that can be used for the analysis.

The first approach, which is what Schrage and Miller used in the original analysis [202], is to view SRPT as a particular case of a non-preemptive threshold based policy. In particular, $W(x)^{SRPT}$ is equivalent to the waiting time of class 2 jobs in the following 3-class non-preemptive priority queue. Class 1 jobs are those with size $< x$. Class 2 jobs are those of size x. Class 3 jobs are those of size $> x$, and only arrive at the instant they achieve remaining size x in the SRPT system. Thus, the arrivals of class 1 and 2 jobs are Poisson, but class 3 jobs do not have a Poisson arrival process. With a little work, it is easy to show that the waiting time of class 2 jobs in the non-preemptive priority queue is equivalent to $W(x)^{SRPT}$, and then results for SRPT follow from those in Section 3.2.2.

However, instead of working through the details of this first approach, we will focus on a second approach for analyzing $T(x)^{SRPT}$ that provides more intuition for the final results. We begin by noting that $W(x)^{SRPT}$ can be viewed as a busy period as follows. First, denote the work with remaining size $< x$ seen by a tagged job upon arrival by Q_x^{SRPT}. Then, notice that the tagged job's waiting time is exactly the time until the system becomes idle of jobs with remaining size $< x$. Noting that a later arrival contributes to the

waiting time of the tagged job only when the size of the arrival is $< x$ gives: $W(x) \stackrel{\mathrm{d}}{=} B_x(Q_x^{SRPT})$. Thus, determining the moments and transform of $W(x)^{SRPT}$ reduces to the problem of understanding Q_x^{SRPT}. Deriving the moments of Q_x^{SRPT} is typically not too difficult because Q_x^{SRPT} is simply the workload of a work conserving queue in a transformed system, Q_x^T, where jobs arrive the instant their remaining size is $< x$. Thus, in the transformed system jobs with original size $< x$ arrive according to a Poisson process with rate $\lambda F(x)$ and jobs with original size $\geq x$ arrive to an idle server when they obtain remaining size $< x$ in the original queue. As an example, let us derive $E[Q_x^{SRPT}]$. First, notice that Q^T is simply the stationary workload of a work conserving queue, so we can view the scheduling policy as FCFS. Next notice that the load in the transformed system is $\widetilde{\rho}(x)$ and the mean of the excess of the job at the server is $E[\widetilde{X_x}^2]/(2E[\widetilde{X_x}])$. Thus, we have that

$$\begin{aligned} E[Q_x^T] &= \rho_x \frac{E[\widetilde{X_x}^2]}{2E[\widetilde{X_x}]} + E[N_q]E[X_x] \\ &= \frac{\lambda}{2}\widetilde{m_2}(x) + E[Q_x^T]\rho(x) \end{aligned}$$

which results in

$$E[Q_x^T] = \frac{\lambda \widetilde{m_2}(x)}{2(1-\rho(x))}$$

Thus, we have that

$$\begin{aligned} E[W(x)]^{SRPT} &= E[B(Q_x^T)] \\ &= \frac{\lambda \widetilde{m}_2(x)}{2(1-\rho(x))^2} \end{aligned}$$

Using either of these techniques, it is possible to derive higher moments and the transform of the $W(x)^{SRPT}$. We summarize these formulas below.

$$\begin{aligned} E[W(x)^2]^{SRPT} &= \frac{\lambda \widetilde{m}_3(x)}{3(1-\rho(x))^3} + \frac{\lambda m_2(x)\lambda \widetilde{m}_2(x)}{(1-\rho(x))^4} \\ \mathcal{L}_{W(x)}(s)^{SRPT} &= \frac{1}{s}(1-\widetilde{\rho}(x))\left(s + \lambda F(x) - \lambda F(x)\mathcal{L}_{B_x}(s)\right) + \lambda \overline{F}(x)\left(1 - e^{-x(s+\lambda F(x)-\lambda F(x)\mathcal{L}_{B_x}(s))}\right) \end{aligned}$$

Combining the results for the waiting time and the residence time of SRPT gives the following formulas for the mean, variance, and transform of conditional response time under SRPT:

$$\begin{aligned} E[T(x)]^{SRPT} &= \int_0^x \frac{dt}{1-\rho(t)} + \frac{\lambda \widetilde{m_2}(x)}{2(1-\rho(x))^2} \\ Var[T(x)]^{SRPT} &= \int_0^x \frac{\lambda m_2(t)}{(1-\rho(t))^3}dt + \frac{\lambda \widetilde{m_3}(x)}{3(1-\rho(x))^3} + \frac{\lambda m_2(x)\lambda \widetilde{m_2}(x)}{(1-\rho(x))^4} - \frac{1}{4}\left(\frac{\lambda \widetilde{m_2}(x)}{(1-\rho(x))^2}\right)^2 \\ &= \int_0^x \frac{\lambda m_2(t)}{(1-\rho(t))^3}dt + \frac{\lambda \widetilde{m_3}(x)}{3(1-\rho(x))^3} + \frac{3}{4}\left(\frac{\lambda \widetilde{m_2}(x)}{(1-\rho(x))^2}\right)^2 - \frac{\lambda^2 x^2 \widetilde{m_2}(x)\overline{F}(x)}{(1-\rho(x))^4} \end{aligned}$$

At this point, it is illustrative to compare the response time of SRPT with those of PSJF and SJF. Intuitively, SRPT provides a middle ground between PSJF and SJF in the sense that SRPT allows small arrivals to preempt a large job in service only if the remaining size of the large job is still large. For this reason, SRPT is sometimes termed a *semi-preemptive* policy [120]. The conditional response time of SRPT provides a formalization of this intuition. In particular, if we compare the residence times of SRPT, PSJF, and SJF we find that

$$R(x)^{SJF} \leq_{st} R(x)^{SRPT} \leq_{st} R(x)^{PSJF}$$

Further, when we compare the waiting times of these three policies we find that

$$W(x)^{PSJF} \leq_{st} W(x)^{SRPT} \leq_{st} W(x)^{SJF}$$

3.2.4.2 The overall mean response time of SRPT

So far, we have only discussed the conditional response time of SRPT. Though the conditional response time is an important measure, typically the metric of interest for applications is the overall response time. As we have in the past, we can easily write a formula for the mean response time of SRPT using the results for $T(x)^{SRPT}$. In particular, we have that

$$\begin{aligned} E[T]^{SRPT} &= \int_0^1 E[T(x)]^{SRPT} dF(x) \\ &= \int_0^\infty \left(\int_0^x \frac{dt}{1-\rho(t)} + \frac{\lambda \widetilde{m_2}(x)}{2(1-\rho(x))^2} \right) f(x) dx \end{aligned}$$

Though this formula is easy to write, and can be evaluated numerically, it provides little information about the behavior of $E[T]^{SRPT}$. Further, the numerical calculations are quite time-consuming – in many situations simulating the policy is faster than evaluating the formulas numerically in Mathematica – and are numerically imprecise at high loads. As a result, it is important to provide simple bounds on the mean response time of SRPT in order to characterize its behavior.

Bounding the mean response time of SRPT

In [241], Wierman, Harchol-Balter, and Osogami prove the first such bounds.

Theorem 3.10

Consider an M/GI/1 SRPT *queue. Let K satisfy $\lambda m_2(x) \leq Kx\rho(x)$. Then*

$$\frac{E[X]}{\rho} \log\left(\frac{1}{1-\rho}\right) \leq E[T]^{SRPT} \leq E[X] \left(\frac{K(1-\rho/2)}{1-\rho} + \left(\frac{1-K}{\rho}\right) \log\left(\frac{1}{1-\rho}\right) \right)$$

Like Theorem 3.7 for PSJF, Theorem 3.10 is stated in terms of a K such that $\lambda m_2(x) \leq Kx\rho(x)$. To understand K, note that $K \leq 1$ for all distributions and $K \leq 2/3$ for decreasing service distributions. Further, Theorem 3.8 provides a characterization of K in terms of the tail behavior of the service distribution.

There are a number of interesting observations that follow from Theorem 3.10. Though the bounds in Theorem 3.10 do not completely characterize the behavior of $E[T]^{SRPT}$, they already provide some useful information. First and foremost, notice that $E[T]^{SRPT} < \infty$ even when $E[X^2] = \infty$. Further, since the bounds on $E[T]^{SRPT}$ in Theorem 3.7 are completely independent of $E[X^2]$, they indicate that SRPT is

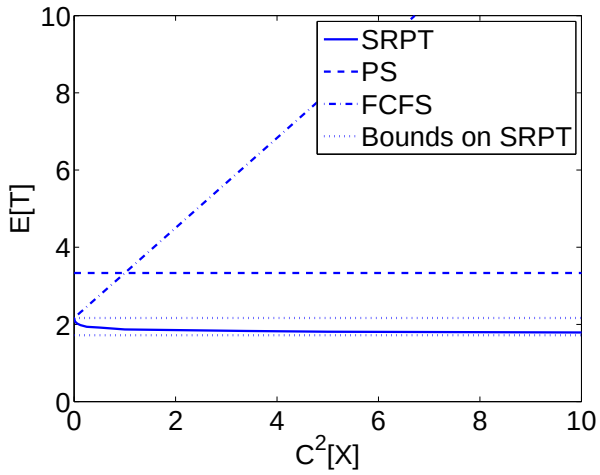


Figure 3.4: *This figure shows the mean response time of* SRPT, PS, *and* FCFS *as a function of the variability of the service distribution ($C^2[X]$). The load is 0.7 and the service distribution is a Weibull with mean 1 in each case. The bounds shown are in Theorem 3.10.*

"nearly" insensitive to variability in the service distribution, similarly to what we saw for PSJF. In addition, as illustrated in Figure 3.4, the bounds are as tight as possible without making use of the variability in the service distribution. To see that the upper bound is tight, consider a deterministic service distributions. Then,

$$E[T]^{SRPT} = E[T]^{FCFS} = \frac{1-\rho/2}{1-\rho}E[X]$$

which is the same as the upper bound with $K = 1$. Further, the lower bound becomes tight when the service distribution is highly variable and load is high (as illustrated in Figure 3.4).

The proof of Theorem 3.10 is a bit involved, so we will break up the proof into a number of lemmas that will be of use later in the thesis.

In order to bound $E[T]^{SRPT}$, we start by characterizing the difference in the residence times of SRPT and PSJF. It turns out that the difference in the residence times of SRPT and PSJF is very related to the difference in the waiting times of SRPT and PSJF. In particular, define

$$E[W_2] = E[W]^{SRPT} - E[W]^{PSJF} = \int_0^\infty \frac{\lambda x^2 f(x)\overline{F}(x)}{2(1-\rho(x))^2}dx$$

Lemma 3.11
In an M/GI/1 queue,

$$2E[W_2] \quad = \quad E[R]^{PSJF} - E[R]^{SRPT}$$

Proof. We can prove this result by repeated interchanging of the integrals.

$$\begin{aligned}
2E[W_2] &= \int_0^\infty \frac{\lambda x^2 f(x)\overline{F}(x)}{(1-\rho(x))^2}dx \\
&= \int_0^\infty f(t)\int_0^t \frac{x\rho'(x)}{(1-\rho(x))^2}dxdt \\
&= \frac{1}{\lambda}\int_0^\infty \frac{\rho'(t)}{1-\rho(t)} - \int_0^\infty f(t)\int_0^t \frac{1}{1-\rho(x)}dxdt \\
&= -\frac{1}{\lambda}\log(1-\rho) - \int_0^\infty \frac{\overline{F}(x)}{1-\rho(x)}dx \\
&= E[R]^{PSJF} - E[R]^{SRPT}
\end{aligned}$$

□

Next, we relate $E[W_2]$ with the waiting time of PSJF.

Lemma 3.12
In an M/GI/1 queue

$$\begin{aligned}
E[R]^{SRPT} + 2E[W]^{PSJF} &\geq E[R]^{PSJF} \quad \textit{and thus} \\
E[W]^{PSJF} &\geq E[W_2]
\end{aligned}$$

Proof. We start by proving a conditional version of the lemma.

$$\begin{aligned}
& E[R(x)]^{SRPT} + 2E[W(x)]^{PSJF} \\
= \; & \frac{x}{1-\rho(x)} - \int_0^x \frac{\rho(x)-\rho(t)}{(1-\rho(x))(1-\rho(t))}dt + \frac{\lambda m_2(x)}{(1-\rho(x))^2} \\
\geq \; & \frac{x}{1-\rho(x)} - \int_0^x \frac{\rho(x)-\rho(t)}{(1-\rho(x))^2}dt + \frac{\lambda m_2(x)}{(1-\rho(x))^2} \\
= \; & \frac{x}{1-\rho(x)} - \frac{x\rho(x) - x\rho(x) + \lambda m_2(x)}{(1-\rho(x))^2} + \frac{\lambda m_2(x)}{(1-\rho(x))^2} \\
= \; & E[R(x)]^{PSJF}
\end{aligned}$$

Now, we decondition.

$$\begin{aligned}
E[R]^{SRPT} + 2E[W]^{PSJF} &\geq \int_0^\infty E[R(x)]^{PSJF} f(x)dx \\
&= E[R]^{PSJF}
\end{aligned}$$

Further, combining the above with Lemma 3.11, we have:

$$E[W]^{PSJF} \geq \frac{1}{2}\left(E[R]^{PSJF} - E[R]^{SRPT}\right) = E[W_2]$$

□

Finally, we upper bound the sum of the residence time of SRPT and the waiting time of PSJF.

Lemma 3.13
Let K satisfy $\lambda m_2(x) \leq Kx\rho(x)$.

$$E[R]^{SRPT} + 2E[W]^{PSJF} \leq \frac{1}{\lambda}\left(\frac{K\rho^2}{1-\rho} + 2K\rho + (2K-1)\log(1-\rho)\right)$$

Proof. We start by proving a conditional version of the theorem.

$$\begin{aligned}
& E[R(x)]^{SRPT} + 2E[W(x)]^{PSJF} \\
= & \int_0^x \frac{dt}{1-\rho(t)} + \frac{\lambda m_2(x)}{(1-\rho(x))^2} \\
= & \frac{x}{1-\rho(x)} - \int_0^x \frac{\rho(x)-\rho(t)}{(1-\rho(x))(1-\rho(t))}dt + \frac{\lambda m_2(x)}{(1-\rho(x))^2} \\
\leq & \frac{x}{1-\rho(x)} - \int_0^x \frac{\rho(x)-\rho(t)}{(1-\rho(x))}dt + \frac{\lambda m_2(x)}{(1-\rho(x))^2} \\
= & \frac{x}{1-\rho(x)} - \frac{x\rho(x) - x\rho(x) + \lambda m_2(x)}{(1-\rho(x))} + \frac{\lambda m_2(x)}{(1-\rho(x))^2} \\
= & E[R(x)]^{PSJF} + \frac{\lambda m_2(x)\rho(x)}{(1-\rho(x))^2}
\end{aligned}$$

Now, we can decondition as follows:

$$\begin{aligned}
E[R]^{SRPT} + & \int_0^\infty \frac{\lambda m_2(x)}{(1-\rho(x))^2} f(x)dx \\
\leq & \int_0^\infty \left(\frac{x}{1-\rho(x)} + \frac{\lambda m_2(x)\rho(x)}{(1-\rho(x))^2}\right) f(x)dx \\
\leq & -\frac{1}{\lambda}\log(1-\rho) + \frac{K}{\lambda}\int_0^\infty \frac{\lambda x f(x)\rho(x)^2}{(1-\rho(x))^2}dx \\
= & -\frac{1}{\lambda}\log(1-\rho) + \frac{K}{\lambda}\left(\frac{\rho^2}{1-\rho} + 2\log(1-\rho) + 2\rho\right) \\
= & \frac{1}{\lambda}\left(\frac{K\rho^2}{1-\rho} + 2K\rho + (2K-1)\log(1-\rho)\right)
\end{aligned}$$

□

Combining the above lemmas, we can now easily prove Theorem 3.10.

Proof of Theorem 3.10. We first prove the upper bound using Lemmas 3.11 and 3.13:

$$\begin{aligned} E[T]^{SRPT} &= -\frac{1}{2\lambda}\log(1-\rho) - \frac{1}{2}E[R]^{SRPT} \\ &\quad +E[W]^{PSJF} + E[R]^{SRPT} \\ &\leq -\frac{1}{2\lambda}\log(1-\rho) \\ &\quad +\frac{1}{2\lambda}\left(\frac{K\rho^2}{(1-\rho)} + 2K\rho + (2K-1)\log(1-\rho)\right) \\ &= \left(K - \frac{K\rho}{2} + (K-1)\left(\frac{1-\rho}{\rho}\right)\log(1-\rho)\right)E[T]^{PS} \end{aligned}$$

Next, we prove the lower bound using Lemma 3.12:

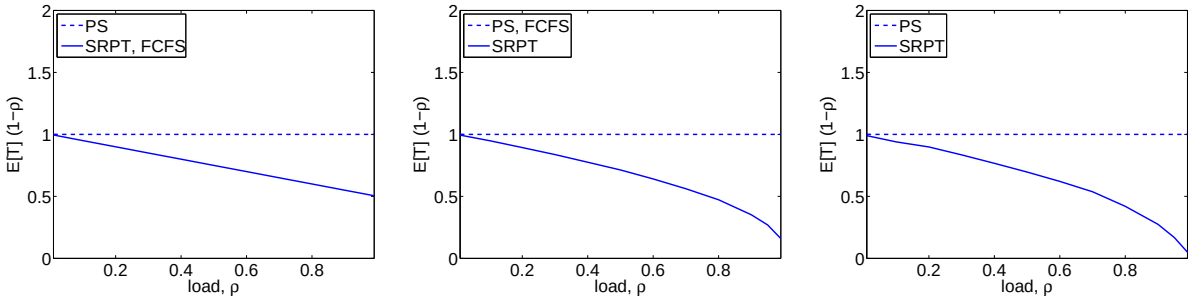
$$\begin{aligned} E[T]^{SRPT} &= -\frac{1}{2\lambda}\log(1-\rho) - \frac{1}{2}E[R]^{SRPT} + E[W]^{PSJF} + E[R]^{SRPT} \\ &\geq -\frac{1}{2\lambda}\log(1-\rho) - \frac{1}{2\lambda}\log(1-\rho) \end{aligned}$$

□

The growth rate of the mean response time of SRPT

The bounds on $E[T]^{SRPT}$ in Theorem 3.10 provide a number of useful insights into the behavior of the mean response time of SRPT; however they also leave a number of questions unanswered. In particular, the growth rate of $E[T]^{SRPT}$ as a function of load under any specific service distribution cannot be inferred from Theorem 3.10. This is unfortunate because understanding the growth rate of $E[T]^{SRPT}$ is fundamental to benchmarking the performance of other policies. For example, we have seen that a number of policies including PSJF and SJF have $E[T] = \Theta\left(\frac{1}{(1-\rho)\log\left(\frac{1}{1-\rho}\right)}\right)$ in the M/M/1. It is natural to wonder how this growth rate compares to that of $E[T]^{SRPT}$. Further, we have results characterizing the growth rate of PSJF and SJF under Pareto service distributions, how do these growth rates compare with optimal?

Bansal recently proved the first result characterizing the growth rate of SRPT under a specific service distribution. In particular, Bansal derived the growth rate of SRPT in the M/M/1 queue [23]. Soon after, Bansal and Gamarnik derived the growth rate of $E[T]^{SRPT}$ under a Pareto service distribution [24]. Not surprisingly, in both cases $E[T]^{SRPT}$ has far better behavior as $\rho \to 1$ than simple policies like PS and FCFS. Summarizing these two results, we have the following.



(a) Deterministic (b) Exponential (c) Pareto ($\alpha = 1.2$)

Figure 3.5: *This figure illustrates the behavior of mean response time under* SRPT, FCFS, *and* PS *as a function of load.*[2] *The mean response time is scaled by* $(1-\rho)$ *in order to highlight differences between the policies. In (a) job sizes are deterministic, in (b) job sizes are exponential, and in (c) job sizes are Pareto with* $\alpha = 1.2$ *(thus they have* $E[X^2] = \infty$*). In all cases,* $E[X] = 1$*. Note that* FCFS *is not included in (c) because* $E[T]^{FCFS} = \infty$ *in this case.*

Theorem 3.14
In an M/M/1 queue

$$E[T]^{SRPT} = \Theta\left(\frac{1}{(1-\rho)\log\left(\frac{1}{1-\rho}\right)}\right)$$

Further in an M/GI/1 queue with $X \sim Pareto(\alpha)$ *where* $\alpha > 1$ *so that* $E[X] < \infty$*,*

$$E[T]^{PSJF} = \begin{cases} \Theta\left(\log\left(\frac{1}{1-\rho}\right)\right), & \text{if } \alpha < 2 \\ \Theta\left(\log^2\left(\frac{1}{1-\rho}\right)\right), & \text{if } \alpha = 2 \\ \Theta\left((1-\rho)^{-\frac{\alpha-2}{\alpha-1}}\right), & \text{if } \alpha > 2. \end{cases}$$

Interestingly, a consequence of Theorem 3.14 is that SRPT, PSJF, SJF, and threshold based policies all have the same growth rate for mean response time in the M/M/1 setting. Further, under a Pareto service distribution, SRPT and PSJF have equivalent growth rates for $E[T]$. In fact, the parallel behavior of the growth rates of SRPT and PSJF observed under these two service distributions holds much more generally: Wierman, Harchol-Balter, and Osogami have proven that:

Theorem 3.15
In an M/GI/1 queue, $E[T]^{SRPT} \leq E[T]^{PSJF} \leq \frac{3}{2}E[T]^{SRPT}$*.*

Proof. First, let use show that $E[W_2]$, which we have defined as $E[W]^{SRPT} - E[W]^{PSJF}$, is also equal

to $E[T]^{SRPT} - E[T]^{PSJF}$:

$$\begin{aligned} E[T]^{SRPT} &= E[R]^{SRPT} + E[W]^{PSJF} + E[W_2] \\ &= \frac{1}{2}E[R]^{PSJF} + \frac{1}{2}E[R]^{SRPT} + E[W]^{PSJF} \\ &= E[T]^{PSJF} - \frac{1}{2}E[R]^{PSJF} + \frac{1}{2}E[R]^{SRPT} \\ &= E[T]^{PSJF} - E[W_2] \end{aligned}$$

Combining the above, with Lemma 3.12 we can complete the proof.

$$\begin{aligned} E[T]^{PSJF} &= E[T]^{SRPT} + E[W_2] \\ &= E[T]^{SRPT}\left(1 + \frac{\frac{1}{2}E[W_2] + \frac{1}{2}E[W_2]}{E[T]^{SRPT}}\right) \\ &\leq E[T]^{SRPT}\left(1 + \frac{E[W]^{PSJF} + E[W_2]}{2E[T]^{SRPT}}\right) \\ &\leq \frac{3}{2}E[T]^{SRPT} \end{aligned}$$

□

Theorem 3.15 is important for many reasons. First and foremost, it is of practical importance that PSJF, which is easier to implement than SRPT since it uses only static properties of jobs, always provides near optimal mean response times. However, almost as importantly, another consequence of Theorem 3.15 is that when trying to characterize the behavior of $E[T]^{SRPT}$ it suffices to study the behavior of $E[T]^{PSJF}$, which is a much simpler task.

Using this observation, we can prove the following result summarizing the effect of an upper bound on the service distribution on the growth rate of $E[T]^{SRPT}$.

Theorem 3.16

Consider an M/GI/1 queue. If the service distribution is bounded, then

$$E[T]^{SRPT} = \Theta\left(\frac{1}{1-\rho}\right).$$

However, if the service distribution is unbounded, then

$$E[T]^{SRPT} = o\left(\frac{1}{1-\rho}\right).$$

Notice the huge impact an upper bound on the service distribution has on the behavior of $E[T]$ as $\rho \to 1$. If there is no upper bound, than, in heavy traffic, SRPT provides huge gains over standard policies like PS and FCFS that have $E[T] = \Theta(1/(1-\rho))$. However, when there is an upper bound on the service

distribution, no scheduling policy can have a heavy traffic growth rate better than $\Theta\left(\frac{1}{1-\rho}\right)$, which can be achieved under even the simplest policies.

Proof of Theorem 3.16. We will start by proving the result in the case of a bounded service distribution. Clearly, the theorem holds for the deterministic service distribution (where SRPT is equivalent to FCFS), so we let X be non-deterministic. Let x_U be the upper bound of the service distribution and $y > 0$ be a point such that $\rho(y) < 1 - \epsilon$ for some $\epsilon > 0$. Then

$$\begin{aligned}
E[T]^{SRPT} &\geq \int_0^{x_u} \frac{\lambda \widetilde{m_2}(x) f(x)}{2(1-\rho(x))^2} dx \\
&\geq \frac{\widetilde{m_2}(y)}{2} \int_y^{x_u} \frac{\lambda x f(x)}{(1-\rho(x))^2} \frac{1}{x} dx \\
&\geq \frac{\widetilde{m_2}(y)}{2x_u} \int_y^{x_u} \frac{\rho'(x)}{(1-\rho(x))^2} dx \\
&= \frac{\widetilde{m_2}(y)}{2x_u} \left(\frac{1}{1-\rho} - \frac{1}{1-\rho(y)} \right) \\
&\sim \frac{1}{1-\rho} \text{ as } \rho \to 1
\end{aligned}$$

To prove the upper bound, we note that

$$\begin{aligned}
E[T]^{SRPT} &\leq E[R]^{SRPT} + \int_0^{x_u} \frac{\lambda \widetilde{m_2}(x) f(x)}{2(1-\rho(x))^2} dx \\
&\leq E[R]^{PSJF} + \frac{x_u^2}{2} \int_0^{y} \frac{\lambda f(x)}{(1-\rho(x))^2} dx + \frac{x_u^2]}{2} \int_y^{x_u} \frac{\lambda x f(x)}{(1-\rho(x))^2} \frac{1}{x} dx \\
&\leq E[R]^{PSJF} + \frac{\lambda x_u^2}{2(1-\rho(y))^2} + \frac{x_u^2}{2y} \int_0^{x_u} \frac{\rho'(x)}{(1-\rho(x))^2} dx \\
&= \frac{1}{\lambda} \log\left(\frac{1}{1-\rho} \right) + \frac{x_u^2}{2y} \left(\frac{1}{1-\rho} - \frac{1}{1-\rho(y)} + \frac{\lambda y}{(1-\rho(y))^2} \right) \\
&\sim \frac{x_u^2}{2y} \frac{1}{1-\rho} \text{ as } \rho \to 1
\end{aligned}$$

Thus, $E[T]^{SRPT} = \Theta\left(\frac{1}{1-\rho}\right)$.

Let us now move to the proof of the result in the case of an unbounded service distribution. By Theorem 3.15, we can equivalently study PSJF. Further, since $E[R]^{PSJF} = \Theta(\log(1/(1-\rho)))$, we need only study the behavior of the waiting time under PSJF.

$$\begin{aligned}
E[W]^{PSJF} &= \int_0^{\infty} \frac{\lambda m_2(x) f(x)}{2(1-\rho(x))^2} dx \\
&= \frac{1}{2} \int_0^{\infty} \frac{\rho'(x)}{(1-\rho(x))^2} \frac{m_2(x)}{x} dx
\end{aligned}$$

For any service distribution with a finite mean, we have that for $x \to \infty$,

$$m_2(x) = \int_0^x t^2 f(t)dt = o(x).$$

Thus, for every $\epsilon > 0$, there is a $y(\epsilon)$ such that for all $x > y(\epsilon)$ $m_2(x)/x < \epsilon$. Further, note that

$$\int_0^{y(\epsilon)} \frac{\rho'(x)}{(1-\rho(x))^2}\frac{m_2(x)}{x} \leq \frac{\lambda m_2(y(\epsilon))}{(1-\rho(y(\epsilon)))}F(y(\epsilon))$$

which is simply $\Theta(1)$ as $\rho \to 1$. Thus,

$$\begin{aligned} E[W]^{PSJF} &\sim \frac{1}{2}\int_{y(\epsilon)}^{\infty} \frac{\rho'(x)}{(1-\rho(x))^2}\frac{m_2(x)}{x}dx \\ &\leq \frac{1}{2}\epsilon\int_{y(\epsilon)}^{\infty} \frac{\rho'(x)}{(1-\rho(x))^2} \\ &\leq \frac{1}{2}\frac{\epsilon}{1-\rho} \end{aligned} \tag{3.20}$$

Since $\epsilon > 0$ was arbitrary, this completes the proof of the unbounded case.

□

3.2.4.3 Competitive analysis in the M/GI/1

Now that we have characterized the optimal mean response time, it is interesting to go back and understand "which scheduling polices provide near optimal mean response times for all loads and service distributions?" In order to address this question, we will borrow the notion of *competitive analysis* from the worst case style of scheduling analysis, and adjust it to fit our stochastic setting. In particular, we introduce the following definition:

Definition 3.1 *A scheduling policy P is c-**competitive** wrt $\mathcal{D}$ if there exists a constant c such that*

$$E[T]^P \leq cE[T]^{SRPT} \quad \forall \rho < 1, \text{ and } X \in \mathcal{D}. \tag{3.21}$$

*When no service distribution is specified, it is assumed that $\mathcal{D}$ is the set of all distributions with finite mean. Further, if there exists some c such that P is c-competitive, then we say simply that P is **competitive**.*

The important point in the above definition is that the constant c does not depend on the load ρ or the service distribution X. Thus, if an algorithm is c-competitive, it means that the response time is no more than c times worse under any load or service distribution. Note that because c must hold across all loads, it must hold as $\rho \to 1$. Further, the interesting case is when the load is very high, since under low loads all policies are within a constant factor. In fact, it is easy to see that a policy P is c-competitive if and only if $E[T]^P = \Theta(E[T]^{SRPT})$ as $\rho \to 1$ under all service distributions.

Guaranteeing that a policy P is competitive is an extremely strong guarantee about the performance of P. In fact, in many ways it seems like an unreachable goal. However, we have already seen one policy that is competitive: PSJF. In particular, Theorem 3.15 proves that $E[T]^{PSJF} \leq \frac{3}{2}E[T]^{SRPT}$ regardless of the load and service distribution; thus PSJF is $3/2$-competitive.

However, outside of PSJF we have not discussed any other competitive policies in this chapter! Even the preemptive threshold based policies from which PSJF is obtained as a limiting case are not competitive when $E[X^2] = \infty$. Further, it is immediate to see that no non-preemptive policies are competitive when $E[X^2] = \infty$, $E[T]^{SRPT} < \infty$ while all non-preemptive policies have $E[T] = \infty$. In addition, no blind policy is competitive since all blind policies have $E[T] = E[X]/(1-\rho)$ in the M/M/1 while $E[T]^{SRPT} = o(1/(1-\rho))$ in this setting.

Despite the fact that non-preemptive and blind policies are not competitive across all service distributions, they can be competitive for certain classes of service distributions. For instance, if the service distribution is bounded, Theorem 3.16 tells us that $E[T]^{SRPT} = \Theta(1/(1-\rho))$, which means that FCFS and PS, in addition to many other non-preemptive and blind policies, are competitive under bounded distributions. Even outside of bounded distributions, some non-preemptive and blind policies can be competitive. In particular, it turns out that SJF is competitive as long as $E[X^2] < \infty$. We will also see that FB, the blind policy we will discuss in the next chapter, can be competitive in some settings.

Theorem 3.17

Consider an M/GI/1 SJF *queue. Let X_1 and X_2 be i.i.d. service demands. If $E[X^2] < \infty$, then* SJF *is $\frac{3E[X^2]}{E[\min(X,X_1)^2]}$-competitive.*

Notice that, though SJF is competitive under service distributions with finite variance, it pays a price in performance for not using preemption. This price in performance can be seen in the fact that the competitive ratio of SJF is much larger than that of its preemptive counterpart PSJF.

Proof. We start by comparing the waiting time of SJF and PSJF.

$$\begin{aligned} E[W]^{SJF} &= \int_0^1 \frac{\lambda E[X^2]}{2(1-\rho(x))^2} dF(x) \\ &= E[W]^{PSJF} + \int_0^1 \frac{\lambda \int_x^\infty t^2 f(t)dt}{2(1-\rho(x))^2} dF(x) \end{aligned}$$

We will now apply the Chebyshev integral inequality. For details on the inequality, see [237]. Noting that $\int_x^\infty t^2 f(t)dt$ is monotonically decreasing and $\frac{\lambda}{2(1-\rho(x))^2}$ is monotonically increasing, the Chebyshev integral inequality gives

$$\int_0^1 \frac{\lambda \int_x^\infty t^2 f(t)dt}{2(1-\rho(x))^2} dF(x) \leq \int_0^1 \int_x^\infty t^2 f(t)dt dF(x) \int_0^1 \frac{\lambda}{2(1-\rho(x))^2} dF(x)$$

Thus,

$$\begin{aligned}
E[W]^{SJF} &\leq \int_0^1 \int_x^\infty t^2 f(t) dt dF(x) \int_0^1 \frac{\lambda}{2(1-\rho(x))^2} dF(x) \\
&\leq E[W]^{PSJF} + \frac{1}{E[X^2]} \int_0^\infty t^2 f(t) F(t) dt \int_0^1 \frac{\lambda E[X^2]}{2(1-\rho(x))^2} dF(x) \\
&\leq E[W]^{PSJF} + \frac{1}{E[X^2]} \left(\int_0^\infty t^2 f(t) dt - \frac{1}{2} \int_0^\infty 2t^2 f(t) \overline{F}(t) dt \right) E[W]^{SJF} \\
&\leq E[W]^{PSJF} + \left(1 - \frac{E[\min(X, X_1)^2]}{2E[X^2]} \right) E[W]^{SJF}
\end{aligned}$$

from which it follows that

$$E[W]^{SJF} \leq \frac{2E[X^2]}{E[\min(X, X_1)^2]} E[W]^{PSJF}$$

Finally, noting that $E[R]^{SJF} \leq E[R]^{PSJF}$ and $E[T]^{PSJF} \leq (3/2)E[T]^{SRPT}$, we have

$$\begin{aligned}
E[T]^{SJF} &\leq \frac{2E[X^2]}{E[\min(X, X_1)^2]} E[T]^{PSJF} \\
&\leq \frac{3E[X^2]}{E[\min(X, X_1)^2]} E[T]^{SRPT}
\end{aligned}$$

□

3.2.5 Foreground-Background scheduling (FB)

Throughout this chapter we have seen examples of policies that provide small response times by discriminating in favor of small jobs. In all of the policies we have studied, this discrimination is accomplished by using job size or remaining size information to prioritize. However, in many applications job size information is not available, thus some other job statistic must be used as a substitute for job size when scheduling. One statistic that can help provide information about the remaining size of jobs is the *age* (a.k.a. the attained service) of a job. To see this, consider the case when the service distribution has a decreasing failure rate (DFR). In this setting, the larger the attained service the longer the remaining size of the job. FB is designed to perform well in exactly this setting.

FB is not as common a policy as many of the disciplines we have discussed in this chapter and as a result the literature on FB in different communities has developed somewhat independently. The policy itself has been referred to by a number of different names, including Foreground-Background Processor-Sharing (FBPS), Least-Attained-Service-Time first (LAS , or LAST), and Shortest-Elapsed-Time first (SET). Currently though, it seems that FB is the commonly excepted name, though LAS is still preferred in some Computer Science communities.

This cacophony of names already provides a fairly complete description of the workings of FB . Under

FB the server is shared evenly at all times among the cohort of jobs with the smallest age. Thus, when a new job arrives it immediately receives service, and continues to reside at the server until its attained service matches that of the cohort of jobs it preempted, at which point the server is shared evenly among the cohort and the new job. In this way, when the service distribution has a DFR, FB is sharing the server among the jobs with the smallest expected remaining size. Thus, in some sense, FB is behaving like a "poor man's SRPT."

Though FB behaves like SRPT when the service distribution is DFR, this clearly will not always be the case. For example, if the service distribution is IFR, then the jobs with the smallest remaining size are likely to be the jobs with the largest age. Thus, FB is doing completely the opposite thing, and using a policy that favors jobs with the largest age (i.e. FCFS) is a much better idea.

The intuition behind FB that we just described can actually be formalized in a very natural way. In particular, Righter and Shantikumar [188, 189] have proven that FB optimizes the queue length process among blind policies when the service distribution has a DFR, but has the largest queue length among blind policies when the service distribution has an IFR. Let $N(t)$ be the queue length (in jobs) at time t.

Proposition 3.18
In the GI/GI/1, let P be a blind policy and the service distribution be DFR, then for all $t \geq 0$,

$$N(t)^{FB} \leq_{st} N(t)^{P} \leq_{st} N(t)^{FCFS}$$

Further, if the service distribution is IFR the inequalities are reversed.

This proposition already indicates that FB has very interesting behavior with respect to $E[T]$. For instance, using Little's Law, it follows that $E[T]^{FB} \leq E[T]^{PS}$ for DFR distributions and $E[T]^{FB} \geq E[T]^{PS}$ for IFR distributions. Further, it is easy to see that all blind policies have equivalent $N(t)$ processes in the M/M/1, thus $E[T]^{FB} = E[T]^{PS}$ in this case.

To understand more about the behavior of mean response time under FB, we need to first derive the behavior of the conditional response time, $T(x)^{FB}$. After deriving the behavior of $T(x)^{FB}$ we will return to a discussion of the behavior of $E[T]^{FB}$.

3.2.5.1 The conditional response time of FB

Let us begin our analysis of $T(x)^{FB}$ by considering the experience of a tagged arrival, j_x, of size x. Notice that no job with age $\geq x$ will ever receive service while there is a job with age $< x$ in the system. Thus, we can transform the service distribution from X to $\widetilde{X}_x = X \wedge x$ without affecting the $T(x)$. Further, notice that the transformed system is still work conserving. Finally, notice that j_x finishes exactly when this transformed system goes idle. Thus, if we define Q_x^{FB} as the steady state work in transformed system and $\widetilde{B}_x(y)$ as the length of a busy period started by y work where arrivals occur at rate λ and with size $\widetilde{X}_x$, we have

$$T(x)^{FB} \stackrel{\mathrm{d}}{=} \widetilde{B}_x(x + Q_x^{FB}) \tag{3.22}$$

From this equation it is easy to obtain the moments and the transform of $T(x)^{FB}$ in the M/GI/1 setting.

$$E[T(x)]^{FB} = \frac{x}{1-\widetilde{\rho}(x)} + \frac{\lambda\widetilde{m_2}(x)}{2(1-\widetilde{\rho}(x))^2} \tag{3.23}$$

$$Var[T(x)]^{FB} = \frac{\lambda x\widetilde{m_2}(x)}{(1-\widetilde{\rho}(x))^3} + \frac{\lambda\widetilde{m_3}(x)}{3(1-\widetilde{\rho}(x))^3} + \frac{3}{4}\left(\frac{\lambda\widetilde{m_2}(x)}{(1-\widetilde{\rho}(x))^2}\right)^2 \tag{3.24}$$

$$\mathcal{L}_{T(x)}(s)^{FB} = \frac{(1-\widetilde{\rho}(x))(s+\lambda-\lambda\mathcal{L}_{\widetilde{B_x}}(s))}{s} e^{-x(s+\lambda-\lambda\mathcal{L}_{\widetilde{B_x}}(s))} \tag{3.25}$$

where $\widetilde{\rho}(x) = \lambda\widetilde{m_1}(x)$ and $\widetilde{m_i}(x) = i\int_0^x t^{i-1}\overline{F}(t)dt = m_i(x) + x^i\overline{F}(x)$.

Since a job immediately receives service under FB, the residence time of FB is equal to the response time of FB. However, the form $T(x)$ under FB parallels that of SRPT so closely, that it will be useful for analytic purposes to break the formulas for the moments of FB into pseudo residence time and waiting time pieces. Thus, we define

$$\begin{aligned} R(x)^{FB} &\stackrel{\mathrm{d}}{=} \widetilde{B_x}(x) \\ E[R(x)]^{FB} &= \frac{x}{1-\widetilde{\rho}(x)} \\ Var[R(x)]^{FB} &= \frac{\lambda x\widetilde{m_2}(x)}{(1-\widetilde{\rho}(x))^3} \end{aligned}$$

and

$$\begin{aligned} W(x)^{FB} &\stackrel{\mathrm{d}}{=} \widetilde{B_x}(Q_x^{FB}) \\ E[W(x)]^{FB} &= \frac{\lambda\widetilde{m_2}(x)}{2(1-\widetilde{\rho}(x))^2} \\ Var[W(x)]^{FB} &= \frac{\lambda\widetilde{m_3}(x)}{3(1-\widetilde{\rho}(x))^3} + \frac{3}{4}\left(\frac{\lambda\widetilde{m_2}(x)}{(1-\widetilde{\rho}(x))^2}\right)^2 \end{aligned}$$

Clearly, the formulas for the moments of FB appear to be quite similar to the corresponding results for PSJF and SRPT that we derived in previous sections. This is an indication that FB truly is discriminating in favor of small jobs, despite the fact that it is scheduling without any information about service demands. To illustrate the parallels between FB and policies that prioritize small jobs, notice that by replacing $m_i(x)$ in the formulas for PSJF with $\widetilde{m}(x)$ we obtain the corresponding formulas for FB. Thus, FB is equivalent to running PSJF on a transformed service distribution.

However, though FB mimics the behavior of PSJF, FB clearly pays a price for not using job size information to prioritize. Since $m_i(x) \leq \widetilde{m}(x)$, we can see that the mean and variance of $T(x)$ are smaller under PSJF than under FB. Similarly, it is easy to bound the first few moments of $T(x)^{SRPT}$ by those of $T(x)^{FB}$.

Theorem 3.19
In an M/GI/1 queue for $P \in \{SRPT, PSJF\}$, $E[T(x)]^P \leq E[T(x)]^{FB}$ and $Var[T(x)]^P \leq Var[T(x)]^{FB}$.

Further, we can bound the distributional behavior of $T(x)^{PSJF}$ by that of $T(x)^{FB}$. Comparing (3.22)

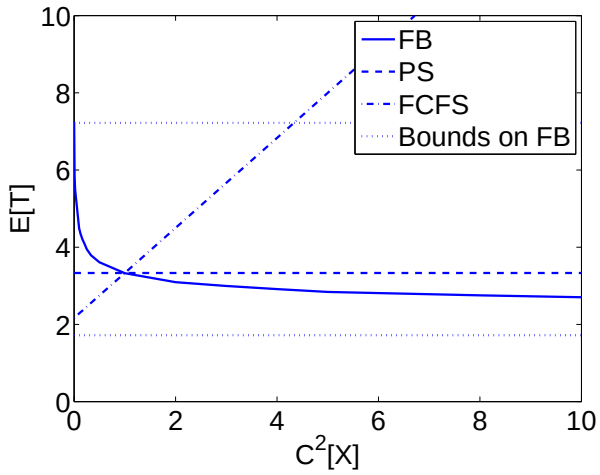


Figure 3.6: *This figure shows the mean response time of* FB, PS, *and* FCFS *as a function of the variability of the service distribution ($C^2[X]$). The load is 0.7 and the service distribution is a Weibull with mean 1 in each case.*

with (3.18) immediately gives that in an M/GI/1 $T(x)^{PSJF} \leq_{st} T(x)^{FB}$. In addition, with a lot more work Nuyens, Wierman, and Zwart [161] recently proved the following stochastic bound comparing both SRPT and PSJF to FB.

Theorem 3.20
In an M/GI/1 queue,

$$R(x)^{PSJF} + W(x)^{SRPT} \leq_{st} T(x)^{FB}$$

Notice that it follows from Theorem 3.20 that both $T(x)^{PSJF} \leq_{st} T(x)^{FB}$ and $T(x)^{SRPT} \leq_{st} T(x)^{FB}$ since we have already observed that $R(x)^{SRPT} \leq_{st} R(x)^{PSJF}$ and $W(x)^{PSJF} \leq_{st} W(x)^{SRPT}$.

3.2.5.2 The overall response time of FB

Now that we understand $T(x)^{FB}$, we can use that information to analyze $E[T]^{FB}$. Like we have done in the past, we can calculate $E[T]^{FB}$ as follows:

$$E[T]^{FB} = \int_0^1 E[T(x)]dF(x)$$

However, as in the cases of PSJF and SRPT, the complicated form of $E[T(x)]$ makes it difficult to understand the behavior of $E[T]$, so more work is necessary.

We have already seen some bounds on $E[T]^{FB}$. In particular, from Proposition 3.18 we know that for all blind policies P, $E[T]^{FB} \leq E[T]^{P}$ when the service distribution has a DFR, and that the reverse holds when the service distribution has an IFR.

In addition, it follows from Theorem 3.20 that

$$E[T]^{SRPT} \leq E[T]^{PSJF} \leq E[T]^{FB}.$$

It is not really surprising that using FB results in larger response times than using either SRPT or PSJF since FB does not use job size information while both SRPT and PSJF do. However, what is surprising is that, although the mean response time of FB is larger than those of PSJF or SRPT, in some cases $E[T]^{FB}$ still has the same growth rate as PSJF and SRPT. In particular, Bansal and Gamarnik prove that $E[T]^{FB} = \Theta(E[T]^{SRPT})$ when the service distribution is Pareto [24].

Theorem 3.21
In an M/GI/1 queue with $X \sim Pareto(\alpha)$ where $\alpha > 1$ so that $E[X] < \infty$,

$$E[T]^{FB} = \begin{cases} \Theta\left(\log\left(\frac{1}{1-\rho}\right)\right), & \text{if } \alpha < 2 \\ \Theta\left(\log^2\left(\frac{1}{1-\rho}\right)\right), & \text{if } \alpha = 2 \\ \Theta\left((1-\rho)^{-\frac{\alpha-2}{\alpha-1}}\right), & \text{if } \alpha > 2. \end{cases}$$

Though the growth rate of $E[T]^{FB}$ is optimal in the case of a Pareto service distribution, it is clear that the growth rate of FB is not always so good. For instance, in the M/M/1,

$$E[T]^{M/M/1/FB} = \Theta\left(\frac{1}{1-\rho}\right).$$

Even worse, in the M/D/1, FB finishes every job at the end of the busy period into which it arrives. Thus,

$$E[T]^{M/D/1/FB} = \theta\left(\frac{1}{(1-\rho)^2}\right),$$

which is as bad as possible under any work conserving policy. Even distributions that are not deterministic can cause the heavy traffic growth rate of FB to be quite bad. In particular, Nuyens has illustrated that $E[T]^{FB}$ has a faster growth rate than PS for a large class of distributions that generalize uniform distributions [159]. We strengthen the result of Nuyens here.

Theorem 3.22
Consider an M/GI/1 with a service distribution of the form $\overline{F}(x) \sim \alpha(x_U - x)^\beta$ for some $\alpha, \beta > 0$ and $x_U < \infty$ as $x \to x_U$. Then as $\rho \to 1$,

$$E[T]^{FB} = \Theta\left(\frac{1}{(1-\rho)^{1+1/(\beta+1)}}\right)$$

Note that as $\beta \to 0$, we again get behavior matching the worst possible behavior under any work-conserving policy. Further, as $\beta \to \infty$ the growth rate converges to $1/(1-\rho)$, which is the same growth rate of PS.

Proof. First, note that $E[R(x)]^{FB} \leq \frac{x}{1-\rho}$, so it will not dominate the behavior as $\rho \to 1$ in this setting

and we can focus on the waiting time. Letting $\mu(x) = f(x)/\overline{F}(x)$ be the failure rate, we have (as $\rho \to 1$),

$$\begin{aligned} E[W]^{FB} &= \int_0^{x_U} \frac{\lambda \widetilde{m}_2(x) f(x)}{2(1-\widetilde{\rho}(x))^2} dx \\ &= \Theta\left(\int_y^{x_U} \frac{\widetilde{\rho}'(x)}{(1-\widetilde{\rho}(x))^2} \mu(x) dx \right) \text{ for arbitrary constant } y \end{aligned}$$

where the second step follows from (i) noting that the contribution of sizes from 0 to y is asymptotically negligible as $\rho \to 1$ since the load these jobs experience is bounded by $\rho(y)$, and (ii) bounding $\widetilde{m}_2(x)$ between $\widetilde{m}_2(y)$ and $E[X^2]$.

Thus, what remains is to understand the behavior of $\mu(x)$. Using the behavior of $\overline{F}(x)$ in the statement of the theorem, we have that as $x \to x_U$,

$$\mu(x) \sim \frac{\alpha\beta(x_U - x)^{\beta-1}}{\alpha(x_U - x)^{\beta}} \sim \frac{\beta}{(x_U - x)}$$

Further, as $\rho \to 1$,

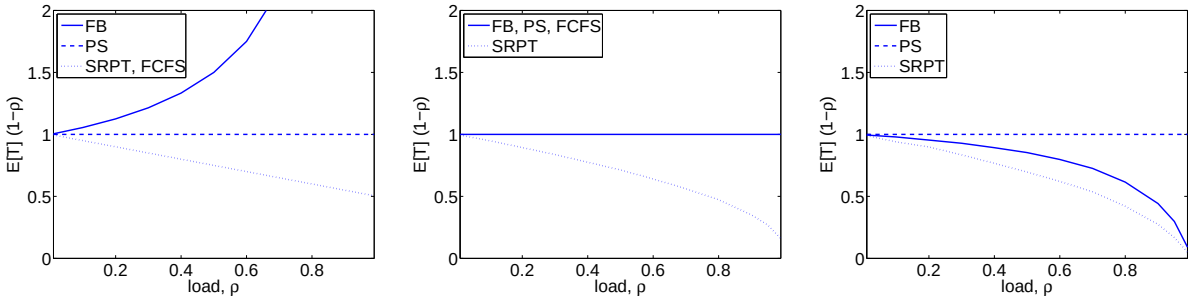
$$1 - \widetilde{\rho}(x) \sim \rho - \widetilde{\rho}(x) = \lambda \int_x^{x_U} \alpha(x_U - t)^{\beta} dt \sim \lambda\alpha(x_U - t)^{\beta+1}\beta$$

Thus, choosing y large enough and then letting $\rho \to 1$, we have

$$\begin{aligned} E[W]^{FB} &= \Theta\left(\int_y^{x_U} \frac{\widetilde{\rho}'(x)}{(1-\widetilde{\rho}(x))^2} \mu(x) dx \right) \\ &= \Theta\left(\int_y^{x_U} \frac{\widetilde{\rho}'(x)}{(1-\widetilde{\rho}(x))^{2+1/(\beta+1)}} dx \right) \\ &= \Theta\left(\frac{1}{(1-\rho)^{1+1/(\beta+1)}} \right) \end{aligned}$$

where the interchange of the integral and the limit in the second step is justified using the bounded convergence theorem.
□

The results we have seen so far illustrate a trend that is common in results about FB: the heavy traffic growth rate of FB is better than that of PS and FCFS when the service distribution is "highly variable", and worse when the service distribution is "lightly variable." However, the results we have seen so far only consider a few specific classes of distributions, so it has not been determined what *properties* of the service distribution lead to good/bad heavy-traffic growth rates under FB. As a step towards answering this question, Nuyens and Wierman [160] prove the following result. The theorem shows that a key determining factor as to whether the heavy traffic growth rate of FB is better or worse that PS and FCFS is whether or not there is an upper bound on the service distribution.



(a) Deterministic (b) Exponential (c) Pareto ($\alpha = 1.2$)

Figure 3.7: *This figure illustrates the behavior of mean response time under* FB *,* SRPT*,* FCFS*, and* PS *as a function of load.*[3] *The mean response time is scaled by* $(1-\rho)$ *in order to highlight differences between the policies. In (a) job sizes are deterministic, in (b) job sizes are exponential, and in (c) job sizes are Pareto with* $\alpha = 1.2$ *(thus they have* $E[X^2] = \infty$*). In all cases,* $E[X] = 1$*. Note that* FCFS *is not included in (c) because* $E[T]^{FCFS} = \infty$ *in this case.*

Theorem 3.23

Consider an M/GI/1 queue with a continuous service distribution with failure rate $\mu(x)$*.*

(i) If the service distribution is bounded, then $E[T]^{FB} = \Omega(1/(1-\rho))$ *as* $\rho \to 1$*.*

(ii) If the service distribution is unbounded and $\widetilde{m_2}(x)\mu(x) = O(1)$*, then* $E[T]^{FB} = O(1/(1-\rho))$ *as* $\rho \to 1$*.*

Note that the condition (ii) holds for most well-behaved unbounded distributions. For instance, if $E[X^2] < \infty$, then it simply requires that $\mu(x)$ is bounded, which occurs under all common unbounded distributions (though it is possible to construct examples where this is not the case, e.g., $f(x) = \sum_{n=1}^{\infty} 1_{[n,n+2^{-n}]}(x)$). On the other hand, if $E[X^2] = \infty$, then $\mu(x)\widetilde{m_2}(x) = O(1)$ requires a tradeoff between the growth of the second moment and the rate of decrease of the hazard rate. However, this tradeoff is met under most common distributions. For example, under regularly varying distributions $\mu(x) = \Theta(1/x)$ and $m_2(x) = O(x)$.

Proof. We will start by proving the result in the case of a bounded service distribution. Let x_U be the upper bound of the service distribution. Note that $\widetilde{\rho}(x) \geq \rho(x)$ and $\rho'(x) = \lambda x f(x)$. Then, we have for all $y \geq 0$,

$$\begin{aligned} E[T]^{FB} &\geq \int_0^{x_U} \frac{\lambda \widetilde{m_2}(x) f(x)}{2(1-\rho(x))^2} dx \\ &\geq \frac{\widetilde{m_2}(y)}{2} \int_y^{x_U} \frac{\lambda x f(x)}{(1-\rho(x))^2} \frac{1}{x} dx \\ &\geq \frac{\widetilde{m_2}(y)}{2x_U} \int_y^{x_U} \frac{\rho'(x)}{(1-\rho(x))^2} dx = \Omega\left(\frac{1}{1-\rho}\right) \text{ as } \rho \to 1. \end{aligned}$$

To prove the result in the case of an unbounded service distribution, note that $\widetilde{\rho}'(x) = \lambda \overline{F}(x)$. Then for all $z \geq 0$,

$$E[T]^{FB} = \int_0^\infty \frac{x}{1-\widetilde{\rho}(x)} f(x)dx + \int_0^\infty \frac{\lambda \widetilde{m_2}(x) f(x)}{2(1-\widetilde{\rho}(x))^2} dx$$
$$\leq \frac{E[X]}{1-\rho} + \int_0^z \frac{\lambda}{2} \frac{\widetilde{m_2}(x)}{(1-\widetilde{\rho}(x))^2} f(x)dx + \int_z^\infty \widetilde{m_2}(x)\mu(x) \frac{\widetilde{\rho}'(x)}{2(1-\widetilde{\rho}(x))^2} dx. \tag{3.26}$$

Since $\widetilde{m_2}(x)\mu(x) = O(1)$, there exists an x_0 and an N such that $\widetilde{m_2}(x)\mu(x) \leq N$ for $x \geq x_0$. Taking $z = x_0$ in (3.26) yields that

$$\begin{aligned} E[T]^{FB} &\leq \frac{E[X]}{1-\rho} + \frac{\lambda x_0 x_0^2 F(x_0)}{2(1-\widetilde{\rho}(x_0))^2} + \int_{x_0}^\infty \widetilde{m_2}(x)\mu(x) \frac{\widetilde{\rho}'(x)}{2(1-\widetilde{\rho}(x))^2} dx \\ &\leq \frac{E[X]}{1-\rho} + O(1) + \frac{N}{1-\rho} = O\left(\frac{1}{1-\rho}\right) \text{ as } \rho \to 1. \end{aligned}$$

□

3.2.5.3 When is FB competitive?

Now that we have spent some time analyzing the mean response time under FB, let us contrast the behavior of FB with that of SRPT. In particular, let us move to a discussion of the competitive ratio of FB. Since FB does not use job size information to schedule, it is clear that it cannot be competitive across all service distributions – we have already discussed a number of distributions where FB is worse than even PS. However, when the service distribution has a DFR, we have seen that FB manages to "prioritize small jobs" in a certain sense. That is, FB prioritizes jobs with small ages, and jobs with small ages are more likely to have small remaining sizes under DFR distributions. Thus, under DFR distributions, one expects that FB provides small $E[T]$, though the response times will certainly be larger than in policies such as PSJF that do not time-share.

Because FB optimizes $E[T]$ among blind policies when the service distribution is DFR, we know that for all DFR distributions, $E[T]^{FB} = O\left(\frac{1}{1-\rho}\right)$. However, since $E[T]^{FB} = E[X]/(1-\rho)$ in the M/M/1, it is clear that FB is not competitive for all DFR distributions. But, if the failure rate is decreasing strongly enough, FB is competitive. In particular, we will show that for distributions where the failure rate, denoted $\mu(x)$, is such that $\mu(x) = \Theta(1/x)$, FB is competitive. This property holds for a large class of practical distributions: *O-regularly varying* distributions. The class of O-regularly varying distributions generalizes regularly varying distributions, and thus includes a wide array of practical distributions such as Pareto distributions. For some background on these distributions, see Section 2.4.2.3.

Theorem 3.24

In an M/GI/1 queue with $\overline{F} \in \mathcal{OR}$, FB *is competitive with respect to $E[T]$.*

Proof. Let us begin by deriving a sufficient condition for proving that FB is competitive. In particular, we

will argue that FB is competitive when

$$\sup_x \frac{1-\rho(x)}{1-\widetilde{\rho}(x)} < \gamma < \infty \tag{3.27}$$

To prove this, we calculate as follows. Noting that $E[R(x)]^{FB}$ is exactly $\frac{1-\rho(x)}{1-\widetilde{\rho}(x)}E[R(x)]^{PSJF}$ and $E[W(x)]^{FB}$ is exactly $(1-\rho(x))^2/(1-\widetilde{\rho}(x))^2 E[W(x)]^{SRPT}$ we have

$$\begin{aligned} E[T(x)]^{FB} &= \frac{1-\rho(x)}{1-\widetilde{\rho}(x)}E[R(x)]^{PSJF} + \left(\frac{1-\rho(x)}{1-\widetilde{\rho}(x)}\right)^2 E[W(x)]^{SRPT} \\ &\leq \left(\frac{1-\rho(x)}{1-\widetilde{\rho}(x)}\right)^2 \left(E[R(x)]^{PSJF} + E[W(x)]^{SRPT}\right) \\ &\leq \gamma^2 \left(E[R(x)]^{PSJF} + E[W(x)]^{SRPT}\right) \end{aligned} \tag{3.28}$$

Further, we have that

$$\begin{aligned} E[R(x)]^{PSJF} - E[R(x)]^{SRPT} &= \frac{x}{1-\rho(x)} - \int_0^x \frac{dt}{1-\rho(t)} \\ &= \int_0^x \frac{(\rho(x)-\rho(t))dt}{(1-\rho(t))(1-\rho(x))} \\ &\leq \int_0^x \frac{(\rho(x)-\rho(t))dt}{(1-\rho(x))^2} \\ &= \frac{\lambda \int_0^x t^2 f(t)dt}{(1-\rho(x))^2} \\ &\leq 2E[W(x)]^{SRPT} \end{aligned} \tag{3.29}$$

Combining (3.28) and (3.29) we have that

$$E[T(x)]^{FB} \leq 3\gamma^2 E[T(x)]^{SRPT}$$

Thus, after proving that (3.27) holds for the class of O-regularly varying distributions, it follows that $E[T]^{FB} \leq 3\gamma^2 E[T]^{SRPT}$.

To prove that (3.27) holds, begin by noting that for all ρ bounded away from 1, (3.27) holds trivially. Next, notice $\rho - \rho(x) = \lambda \int_x^\infty t dF(t)$ and, from partial integration we have that

$$\int_x^\infty t dF(t) = -\int_x^\infty t d\overline{F}(t) = -t\overline{F}(t)|_{t=x}^\infty + \int_x^\infty \overline{F}(t)dt = x\overline{F}(t) + \int_x^\infty \overline{F}(t)dt$$

It then follows that

$$\lim_{x\to\infty}\lim_{\rho\to 1}\frac{1-\rho(x)}{1-\widetilde{\rho}(x)} = \lim_{x\to\infty}\frac{\int_x^\infty t dF(t)}{\int_x^\infty \overline{F}(t)dt} = 1 + \lim_{x\to\infty}\frac{x\overline{F}(x)}{\int_x^\infty \overline{F}(t)dt}$$

Now, we complete the proof by applying Karamata's Theorem for $\mathcal{OR}$ distributions from [56].

□

3.2.6 Other priority based policies

To this point, we have focused almost entirely on "smart" priority based policies, i.e. policies that prioritize small jobs. It is natural to focus on such policies since prioritizing small job sizes typically results in policies that provide small mean response times. However, it is important to at least mention policies that use a contrasting heuristic – policies that prioritize large job sizes.

We have seen that there are a wide variety of different policies that prioritize small jobs, similarly there are a number of different policies that prioritize large job sizes. The three most common policies are Longest-Remaining-Processing-Time first (LRPT), Preemptive-Longest-Job-First (PLJF), and non-preemptive Longest-Job-First (LJF). Though, for the most part, these policies are not of practical interest, they provide an interesting contrast in behavior when compared with policies that favor small jobs. Further, we will find them useful as upper bounds on a number of occasions in the thesis.

Our goal in this section is simply to provide some background introducing each of LRPT, PLJF, and LJF.

3.2.6.1 Longest-Remaining-Processing-Time (LRPT)

Under LRPT, the job in the system with the longest remaining size is given preemptive priority. Thus, under LRPT no job can finish service before the end of a busy period because if it were to do so it would have smaller remaining size than another job in the system. As a result, LRPT finishes every job at the last moment possible under any work conserving policy.

Using the above observation, the analysis of LRPT is fairly straightforward. In particular, a tagged job of size x will finish at the end of the residual busy period into which it arrives. That is, it will finish at the end of a busy period started by $Q + x$ work. Thus,

$$T(x)^{LRPT} = B(x + Q)$$

The moments and transform of LRPT follow immediately from the above formula.

$$\begin{aligned} E[T(x)]^{LRPT} &= \frac{x}{1-\rho} + \frac{\lambda E[X^2]}{2(1-\rho)^2} \\ Var[T(x)]^{LRPT} &= \frac{\lambda x E[X^2]}{(1-\rho)^3} + \frac{\lambda E[X^3]}{(1-\rho)^3} + \frac{3}{4}\left(\frac{\lambda E[X^2]}{(1-\rho)^2}\right)^2 \\ \mathcal{L}_{T(x)}(s)^{LRPT} &= \frac{(1-\rho)(s+\lambda-\lambda\mathcal{L}_B(s))}{s} e^{-x(s+\lambda-\lambda\mathcal{L}_B(s))} \end{aligned}$$

It is important to notice that $E[T]^{LRPT} = \Theta(1/(1-\rho)^2)$ as $\rho \to 1$, which provides an upper bound on the heavy traffic growth rate of any work conserving policy.

3.2.6.2 Preemptive-Longest-Job-First (PLJF)

The second policy that favors long jobs which we will introduce is PLJF. Under PLJF the job in the system with the largest original size receives preemptive priority. Thus, a job of size x can only receive service when there are no jobs of size $> x$ in the system.

As with PSJF, PLJF can be viewed as the limiting case of a preemptive priority queue. Thus, we can obtain the moments and transform of response time directly from the results in Section 3.2.3. For example, we can write the mean and variance of $T(x)$ as follows:

$$E[T(x)]^{PLJF} = \frac{x}{1-\rho+\rho(x)} + \frac{\lambda(E[X^2]-m_2(x))}{2(1-\rho+\rho(x))^2}$$

$$Var[T(x)]^{PLJF} = \frac{\lambda x(E[X^2]-m_2(x))}{(1-\rho-\rho(x))^3} + \frac{\lambda(E[X^3]-m_3(x))}{(1-\rho+\rho(x))^3} + \frac{3}{4}\left(\frac{\lambda(E[X^2]-m_2(x))}{(1-\rho+\rho(x))^2}\right)^2$$

3.2.6.3 Non-preemptive Longest-Job-First (LJF)

The last policy we will discuss in this section is LJF. Under LJF the job in the system with the largest original size receives non-preemptive service. That is, at completion instants, the largest job in the queue is chosen for service, and the service of this job is not interrupted, even if a larger job arrives.

As with SJF, LJF can be viewed as the limiting case of a non-preemptive priority queue. So, the moments and transform of the response time of LJF follow from taking the appropriate limits of the results in Section 3.2.2. In the case of the first and second moments of $T(x)$, this results in:

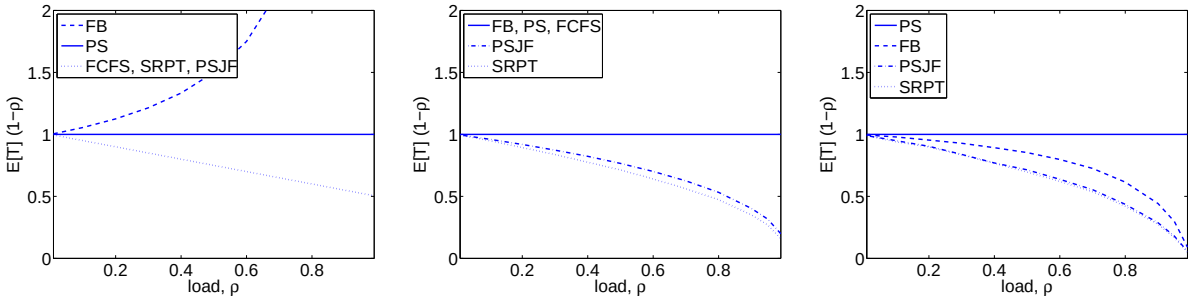
$$E[T(x)]^{LJF} = x + \frac{\lambda E[X^2]}{2(1-\rho+\rho(x))^2}$$

$$E[W(x)^2]^{LJF} = \frac{\lambda E[X^3]}{(1-\rho+\rho(x))^3} + \frac{\lambda^2 m_2(x) E[X^2]}{(1-\rho+\rho(x))^4}$$

3.3 Concluding remarks

In this chapter we have provided an overview of the common scheduling policies and analytic techniques that are the basis of the theoretic study of scheduling in queueing. We have covered a range of simple common policies, such as FCFS, PLCFS, and PS, in addition to a range of priority based policies, such as PSJF, SRPT, and FB. For each of these policies we have discussed primarily the mean response time metric, though we often presented the Laplace transform of response time as well. In addition, we presented a number of different techniques that are used to analyze scheduling policies in the M/GI/1 setting. These techniques have ranged from tagged job techniques and branching process methods, to asymptotic approaches and renewal arguments.

Figure 3.8 illustrates the comparison between a number of the policies we have discussed. We have seen that FCFS, probably the simplest and most common policy, actually performs very well when job sizes are not variable. For instance, under deterministic job sizes FCFS is optimal for $E[T]$. But, when job sizes are highly variable, FCFS is a disaster because many small jobs are forced to queue up behind large jobs. PS, another simple and common policy, time-shares the server in order to allow small jobs to bypass large jobs in the queue. This is effective when variability is high, and PS outperforms FCFS if $C^2[X] > 1$, but when



(a) Deterministic (b) Exponential (c) Pareto ($\alpha = 1.2$)

Figure 3.8: *This figure illustrates the behavior of mean response time as a function of load under a range of common policies. The mean response time is scaled by $(1 - \rho)$ in order to highlight differences between the policies. In (a) job sizes are deterministic, in (b) job sizes are exponential, and in (c) job sizes are Pareto with $\alpha = 1.2$ (thus they have $E[X^2] = \infty$). In all cases, $E[X] = 1$. Note that* FCFS *is not included in (c) because $E[T]^{FCFS} = \infty$ in this case.*

job sizes are not variable, PS can give up as much as a factor of 2 when compared with FCFS. However, beyond these simple policies, we have seen that policies that prioritize small jobs can provide dramatic improvements in mean response time. SRPT and PSJF perform well across all service distributions, and provide orders-of-magnitude improvements in mean response time when job size variability is high. Further, even without knowing job sizes, FB is able to almost match the performance of SRPT and PSJF when job sizes are highly variable.

In this chapter we have primarily presented classical results about scheduling policies. However, apart from the simplest policies, classical results provide very complex formulas characterizing the behavior of mean response time. For example, the in order to calculate $E[T]^{SRPT}$ it is necessary to numerically evaluate a triply-nested integral. Typically, it is faster to simulate the result than it is to evaluate it numerically. The complexity of the results have hidden many behavioral properties of these policies for many years. To remedy this, we have provided a number of new results characterizing the behavioral properties of priority-based policies such as SRPT, PSJF, and FB. In particular, we have provide new results showing that SRPT and PSJF are "nearly insensitive" to job size variability (Theorems 3.7 and 3.10) and we have proven a number of new results characterizing the growth rate of $E[T]$ as a function of load under priority based policies (e.g. Theorems 3.16 and 3.23). Surprisingly, we have found that the growth rate under policies that prioritize small jobs is strongly tied to properties of the service distribution, as illustrated in Figure 3.8.

Let us end this chapter by reminding the reader that there are many gaps between the traditional theoretical results that we have summarized in this chapter and the needs of system designers. As we discussed in Chapter 1, the idealized policies traditionally studied in theory are not implemented in practice. Further, many other metrics besides mean response time are important in practice. Finally, the M/GI/1 model that we have focused on in this chapter ignores many factors that are important in practice. Bridging these gaps will be the focus of the remainder of the thesis.

PART II

Scheduling Classifications: Moving Beyond Idealized Policies

Theoretical research studying the scheduling of queues has traditionally focused on a limited range of idealized scheduling policies, however these idealized policies are hardly ever implemented in their pure form in computer systems. For example, though many recent systems have been designed using the heuristic of "prioritizing small jobs" none have implemented pure SRPT. There are many reasons for this. One reason is that, in many cases, job sizes and remaining sizes are not known exactly, and must be estimated. Another reason is that metrics beyond mean response time are also important. For instance, if one wants to provide "fair" service or QoS guarantees, then hybrids of SRPT will outperform the pure version. In addition, one can easily list many other reasons why the idealized policies that are traditionally studied in theory are not used in practice.

To provide theoretical results that are applicable to the policies implemented in practice, it is important to move beyond the study of individual scheduling policies. In Part II of this thesis, we develop a new theory for scheduling based on studying scheduling heuristics and techniques instead of individual policies. For example, though pure SRPT is not implemented in practice, the policies that are implemented still obey the same general heuristic of "prioritizing small jobs." So, by characterizing the performance of all policies that "prioritize small jobs," we can provide theoretical results for the policies that are implemented in practice. Further, the analysis of scheduling classifications exposes the performance impact of scheduling techniques and heuristics, which provides a deeper theoretical understanding of scheduling than the analysis of individual policies.

Part II is divided into two chapters. In Chapter 4 we introduce classifications of scheduling policies based on *scheduling heuristics* (e.g. we introduce classes of policies that prioritize small/large jobs); and then in Chapter 5 we introduce classifications of scheduling policies based on *scheduling techniques* (e.g. we formalize the class of age based policies, the class of remaining size based policies, and others). Throughout these chapters, in addition to defining the classifications, we prove bounds on the mean response time of each class.

The classifications we introduce in Part II of the thesis serve as building blocks for the remainder of the thesis. We will return to these classes throughout the thesis in order to discuss their performance with respect to other performance metrics such as fairness and predictability.

CHAPTER 4

Classification via scheduling heuristics

Many recent designs of computer systems have been motivated by the optimality of SRPT for mean response time, but none have implemented pure SRPT. For example, in web servers, the version of SRPT that was implemented uses only 5-10 priority levels and prioritizes based on estimates of the remaining service demands of jobs. However, the policies that are implemented, though not pure SRPT, still obey the heuristic of "prioritizing small jobs." Thus, to provide theoretical results that are applicable to the policies implemented in practice, it is important to move beyond the study of individual scheduling policies and study scheduling heuristics instead. By proving results about a class of policies that "prioritizes small jobs" instead of just proving results about SRPT, we can provide results for the policies that are actually implemented in practice.

The heuristic of "prioritizing small jobs" is probably the most common scheduling heuristic, but many others are also used. In this chapter, we introduce and study classifications of scheduling policies that formalize four common scheduling heuristics:

(i) The SMART classification formalizes the heuristic of prioritizing small jobs
(ii) The FOOLISH classification formalizes the heuristic of prioritizing large jobs
(iii) The SYMMETRIC classification is a broad generalization of the classical PS policy
(iv) The PROTECTIVE class formalizes a notion of fairness

These classes are illustrated in Figure 4.1. The SMART class was introduced by Wierman et. al. [241]; the FOOLISH class is novel to this thesis; the SYMMETRIC class was introduced by Kelly [113] in the context of queueing networks; and the PROTECTIVE class was introduced by Henderson and Friedman [78].

The range of scheduling heuristics spanned by these four classes is broad enough to allow us to characterize the impact of disparate scheduling heuristics. These four heuristics represent the type of scheduling used in a wide range of computer applications. The heuristic of biasing towards small jobs sizes has been applied to a range of applications, including both web servers [52, 96, 182] and databases [138, 139]. While FOOLISH policies are not practical in many settings, in some cases it is necessary for large user tasks to receive precedence over smaller background tasks. SYMMETRIC disciplines, such as PS, are primarily used to model the behavior of scheduling policies in network applications, such as routers [34, 179]. Finally,

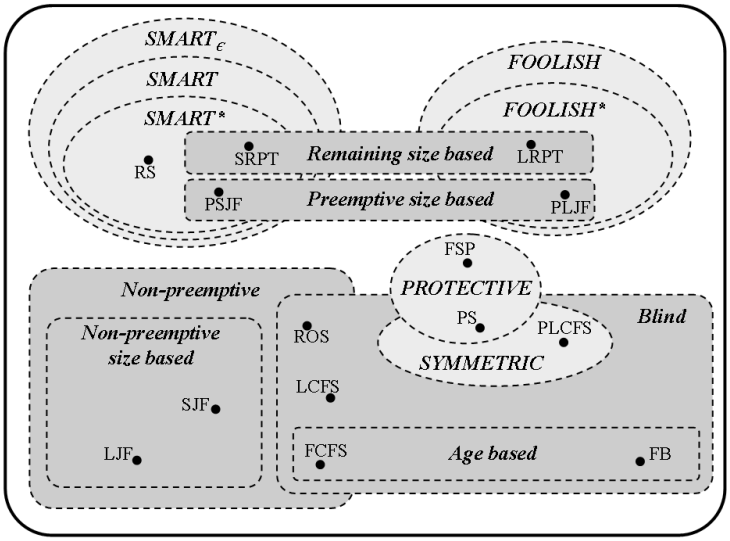


Figure 4.1: *An illustration of the common policies that fall into each of the classifications studied in this thesis. The scheduling heuristic classifications introduced in this chapter are shown in ovals and the scheduling technique classifications introduced in Chapter 5 are shown in rectangles.*

PROTECTIVE policies have recently been suggested for use in applications where concerns about fairness are prevalent, such as web servers.

For each of the four scheduling heuristics described above, this chapter provides an introduction to the classification that includes examples of policies in the class and bounds on both $E[T(x)]$ and $E[T]$ for policies in the class. The bounds on $E[T]$ help to characterize the overall efficiency of each heuristic and the bounds on $E[T(x)]$ illustrate the effect of scheduling heuristics on the response times of individual job sizes, which is important when contrasting the impact of the heuristics.

The chapter is organized as follows. We begin with the SMART class in Section 4.1. Because of the practical importance of "prioritizing small jobs," we will also discuss the tradeoff between breadth of the class and the tightness of results provable about the class in Section 4.2 using a generalization of SMART called SMART$_\epsilon$. The SMART$_\epsilon$ classification will allow us to include policies that "prioritize small jobs" without exact size information. Next, we move to a discussion of the FOOLISH classification in Section 4.3, which is followed by discussions of the SYMMETRIC class in Section 4.4 and the PROTECTIVE class in Section 4.5. Finally, we conclude the chapter in Section 4.6 by contrasting the bounds on $E[T]$ and $E[T(x)]$ under the four scheduling heuristics discussed in the chapter.

Though we will discuss a number of scheduling heuristics in this chapter, one heuristic in particular is by far the most interesting from a practical perspective: that of prioritizing small jobs. Thus, the definitions of SMART and SMART$_\epsilon$ represent an important contribution both theoretically and practically. We will prove that all SMART policies are within a factor of 2 of optimal with respect to mean response time. Further, we will show that even when policies prioritize small jobs using only estimates of job sizes, they are still within a constant factor of optimal with respect to mean response time, where the constant depends on the

accuracy of the estimates. These results provide a theoretical validation of recent designs for web servers, wireless networks, etc. that apply the heuristic of "prioritizing small jobs" but do not implement pure SRPT [182, 131, 130, 102, 136]. Further, these results aid system designers in determining how good job size estimates must be to provide a desired level of performance.

4.1 The class of SMART policies

It is well known that policies that "prioritize small jobs" perform well with respect to mean response time. As we have already discussed, this idea has been fundamental to many computer systems applications ranging from web servers and routers to supercomputing centers and operating systems. However, despite the fact that the same heuristic guides all these implementations, the policies that result differ due to (i) implementation restrictions and (ii) concerns about metrics other than mean response time (e.g. avoiding starvation of large jobs). In particular, hybrid policies are used instead of the idealized policies that prioritize small jobs that are studied in the theoretical literature, such as SRPT and PSJF. The SMART class formalizes the heuristic of "prioritizing small jobs" in order to provide "SMAll Response Times[1]" using three simple properties described below. These three properties are broad enough to allow the class to include practical hybrid policies and simple enough to be easy to understand and maintain.

In this section, we will define the class of SMART policies and then validate the heuristic of "prioritizing small jobs" by deriving simple bounds on the mean response time of any policy in the SMART class. Our bounds illustrate that *all* policies in the SMART class have near optimal mean response times. In fact, all SMART policies have mean response time within a factor of two of optimal across all loads and all service distributions. In addition to bounding the overall mean response time under SMART policies, we also prove stochastic bounds on the conditional response time, $T(x)$, under SMART policies.

Though we only bound the response time of SMART policies in this section, the SMART class will serve as a fundamental part of the thesis, and so we will return to the SMART class throughout the thesis in order to discuss how SMART policies perform with respect to many other important performance metrics, e.g. we will discuss the response time distribution under SMART policies in Chapter 6 and we will discuss the fairness of SMART policies in Chapter 7.

4.1.1 Defining SMART scheduling

We will now formally define the SMART class. Jobs will typically be denoted by a, b, or c. Job a will have remaining size r_a, original size s_a, and arrival time t_a. The original sizes, remaining sizes, and arrival times of b and c are defined similarly. Throughout, we define *job a to have priority over job b* if job b can never run while job a is in the system.

We now define SMART as follows.

Definition 4.1 *A work conserving policy* P *belongs to the* SMART *class, denoted* $\mathsf{P} \in \mathsf{SMART}$, *if it obeys the following properties.*

Bias Property: *If $r_b > s_a$, then job a has priority over job b.*

[1] We thank Hanoch Levy for his suggestion of this acronym.

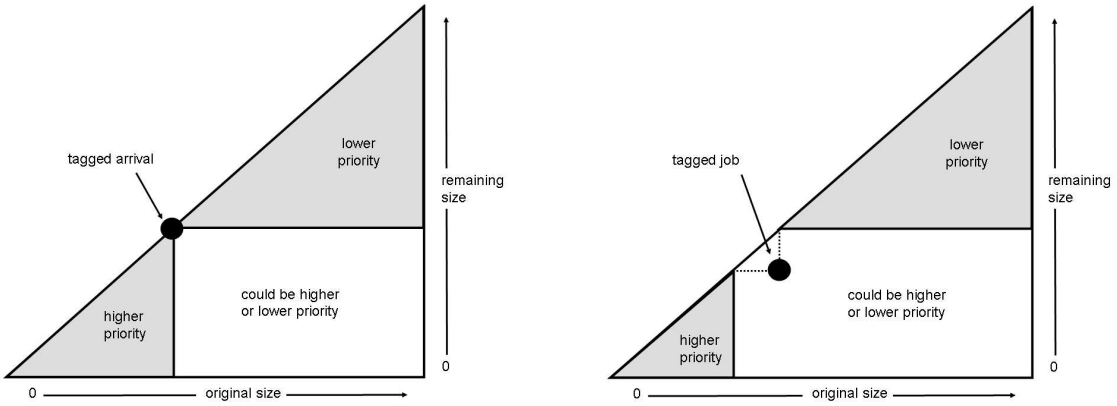


Figure 4.2: *These diagrams illustrate the priority structure induced by the Bias Property in the definition of* SMART *(Definition 4.1). Furthermore, the Consistency and Transitivity properties in the* SMART *definition guarantee that a job will find at most one job with higher priority in the white region upon arrival.*

Consistency Property: *If job a ever receives service while job b is in the system, thereafter job a has priority over job b.*

Transitivity Property: *If an arriving job b preempts job c at time t; thereafter, until job c receives service, every arrival, a, with size $s_a < s_b$ is given priority over job c.*[2]

This definition has been crafted to mimic the heuristic of biasing towards jobs that are (originally) short or have small remaining service requirements. Each of the three properties formalizes a notion of "smart" scheduling. The Bias Property guarantees that the job being run at the server has remaining size smaller than the original size of all jobs in the system. In particular, this implies that the server will never work on a *new arrival* of size greater than x while a previous arrival of original size x is in the system. The priority structure enforced by the Bias Property is illustrated in Figure 4.2.

The Consistency and Transitivity Properties ensure coherency in the priority structure enforced by the Bias Property. In particular, the Consistency Property prevents time-sharing by guaranteeing that after job a is chosen to run ahead of b, job b will never run ahead of job a. Said a different way, this means that once job a is given priority over job b, job a will forever have priority over b. This makes intuitive sense because our priority function is based on the heuristic of giving priority to small jobs, and as job a receives service, it can only get smaller. Finally, the Transitivity Property guarantees that SMART policies do not second guess themselves: if an arrival a is estimated to be "smaller" than job b (and hence is given priority over job b), future arrivals smaller than a are also considered "smaller" than b until b receives service.

It is important to point out that the definition of SMART has been constructed so as not to enforce a total ordering on the priorities of jobs in the system. Instead, only a *partial ordering* is forced, and thus SMART policies can, for instance, change how the policy makes decisions at arrival and departure instants. See Figure 4.3 for an example. This is an important point to bring out because traditional analysis of

[2]Note that every such job a would have had priority over job b at time t due to the Bias Property since $s_a < s_b = r_b(t)$, where $r_b(t)$ is the remaining size of b at time t.

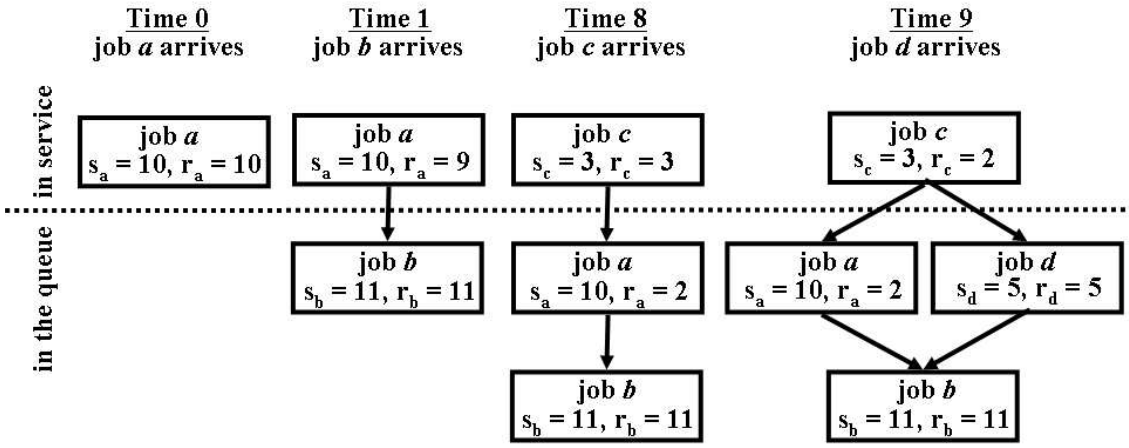


Figure 4.3: *This example illustrates that the* SMART *definition only enforces a partial ordering on the priorities of jobs in the system. Thus a* SMART *policy may change its priority rule over time, e.g. from* PSJF *to* SRPT *at time 9 in the example. In the diagram, an arrow from a to b indicates that a has priority over b. Up until time 9, jobs have been scheduled according to* PSJF *. However, after time 9, if* PSJF *scheduling is continued, job d will receive service before job a, and if* SRPT *is used instead, job a will receive service before job d. Both of these choices are possible regardless of the priority rule used up to time 9.*

scheduling policies assumes that policies obey one fixed priority rule, while SMART policies may change their prioritization rule over time. This fact complicates much of the analysis of the SMART class, but (as we will see) is important for the applicability of the classification to practical settings.

4.1.2 Examples of SMART policies

Many common policies are part of the SMART class – Figure 4.1 provides an overview. Of course, the SMART class includes SRPT and PSJF. Further, it is easy to prove that the SMART class includes the RS policy, which assigns to each job the product of its remaining size and its original size and then gives highest priority to the job with lowest product. Likewise, the SMART class includes many generalizations of these policies. Specifically, SMART includes all policies of the form R^iS^j, where $i, j > 0$ and a job is assigned the product of its remaining size raised to the ith power and its original size raised to the jth power (where again the job with highest priority is the one with lowest product). The SMART class also includes a range of policies having more complicated priority schemes. To illustrate this breadth, we introduce the SMART$*$ classification, a subset of SMART including a range of static priority policies.

Definition 4.2 *A policy $P \in$ SMART$*$ if, at any given time, P schedules the job with the highest priority and gives each job of size s and remaining size r a priority using a fixed priority function $p(s, r)$ such that for $s_1 \leq s_2$ and $r_1 < r_2$, $p(s_1, r_1) > p(s_2, r_2)$.*

Note that SMART$*$ includes all common SMART policies (e.g. SRPT, PSJF, and RS), but that there are still many SMART policies that are not in SMART$*$.

Theorem 4.1
SMART$* \subsetneq$ SMART

Proof. Suppose policy $P \in$ SMART$*$. To see that the Bias Property is maintained, let s_1 and r_1 be the original size and current remaining size of a tagged job in the queue. Suppose s_2 and r_2 correspond to the the original size and current remaining size of another job in the queue such that $r_2 > s_1$. It follows that $s_2 > r_2 > s_1 > r_1$. Thus, $p(s_2, r_2) < p(s_1, r_1)$, so job 2 will not be served.

To see that SMART$*$ policies obey the Consistency Property observe that $p(s, r_1) > p(s, r_2)$ for $r_1 < r_2$ under all SMART$*$ policies. Thus, while serving, a job can only increase its priority, which is already the highest in the system.

To see that SMART$*$ policies obey the Transitivity Property, assume that an arrival with size s_2 preempts a job in service with size s_1 and remaining size r_1. Thus $p(s_1, r_1) < p(s_2, s_2)$. Under any SMART$*$ policy, a future arrival of size $s_3 < s_2$, must have $p(s_3, s_3) > p(s_2, s_2) > p(s_1, r_1)$, which completes the argument.

Finally, notice that SMART is strictly larger than SMART$*$. We can see this by giving an example of a policy in SMART that is not in SMART$*$. One such example is a policy P that simply alternates the priority function across busy periods, i.e. uses priority function $p_1(s, r)$ for odd numbered busy periods and priority function $p_2(s, r)$ for even numbered busy periods where $p_1 \neq p_2$ are both in SMART$*$.
□

Given the range of performance metrics used in modern computer systems, it is of practical importance that SMART includes such a wide range of static priority policies. In particular, systems typically need to perform well for a combination of metrics, e.g., mean response time, mean slowdown, and response-time tail. For many of these metrics, the optimal SMART policy is not SRPT or PSJF, it depends on the service distribution. For instance, no single SMART policy can optimize the mean slowdown across all service distributions, thus the best choice for optimizing a combination of mean response time and mean slowdown depends on the service distribution. A key motivation for characterizing the class as a whole instead of studying the individual policies in the class is that no single SMART policy is optimal for all applications.

Apart from static priority policies, SMART also includes *time-varying policies*, i.e. policies that can change their priority rules over time based on system-state information or randomization. These generalizations are possible because the SMART definition enforces only a *partial ordering* on priorities of jobs in the system. It is of enormous practical importance that time-varying policies are included in SMART because it allows system designers to use the SMART class in order to perform *online multi-objective optimization*. Specifically, suppose a system designer wants to optimize a secondary objective while still providing small mean response times. In order to accomplish this, the system designer can implement a parameterized version of SMART, such as prioritizing based on $p(s, r) = s^{-i} r^{-j}$, and then use machine learning techniques to search the space (i, j) online for the SMART policy that optimizes the secondary objective. (Note that i and j can be chosen to achieve SRPT, PSJF, RS, and many other policies.) This technique can be extremely useful in web applications where the service distribution is time-varying and thus the optimal scheduling policy is not static. Because SMART includes time-varying policies, the bounds on the mean response time from prior work and on the tail of response time proven here will hold even as the priority

function varies. The inclusion of online optimization policies is another key benefit of studying the SMART class as a whole, as analysis of such policies is absent from the literature.

4.1.3 Policies excluded from SMART

To this point we have only discussed the breadth of SMART; however it is also important to note that many policies are excluded from SMART. Clearly, SMART does not include policies that give priority to large jobs such as LJF, PLJF, and LRPT. In addition, SMART does not include policies that only "weakly" prioritize small jobs. For example, SMART does not include any non-preemptive policies, not even ones like SJF that prioritize small jobs; nor does it include policies that do not use knowledge about the job sizes (blind policies), not even FB.

The exclusion of these policies is a result of the tension between the *breadth* of the class and the *tightness* of the results provable about the class. In particular, excluding policies such as SJF and FB that bias weakly towards small job sizes is necessary in order to show that SMART policies provide a near optimal mean response time across all service distributions and all loads. For example, though SJF can provide good mean response time when the second moment of the service distribution, $E[X^2]$ is small, the mean response time of SJF is arbitrarily larger than the optimal as $E[X^2] \to \infty$. Similarly, though FB can provide near optimal mean response time under service distributions having decreasing failure rates, when the service distribution has an increasing failure rate, FB is one of the worst disciplines to use. In particular, when the service distribution is deterministic, the quotient $E[T]^{\mathsf{FB}}/E[T]^{\mathsf{SRPT}}$ can be arbitrarily large.

The tension between the breadth and tightness of the class also leads to the exclusion of policies that use only job size estimates and that use only a finite number of priority levels (such policies violate the Bias Property). It is particularly unfortunate to exclude these policies because in many cases system designers are forced to use job size estimates and simplify implementations by using only 5-10 priority levels without sacrificing too much performance in practice, see for example [96, 180, 131]. We will discuss a generalization of SMART that includes such policies in Section 4.2.

4.1.4 Bounding response times for SMART policies

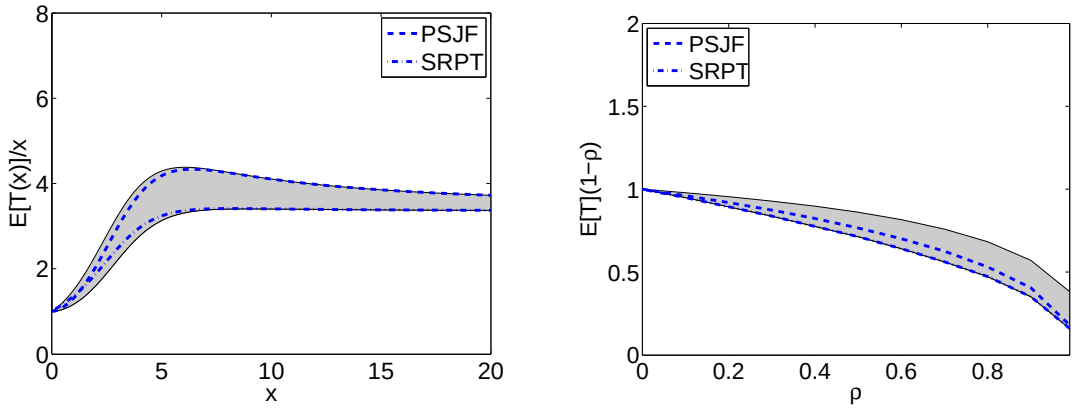
The strength of the SMART classification comes from the fact that we can show that all SMART policies have near optimal $E[T]$. Thus, all SMART policies, even practical hybrid policies, provide provably "SMAll Response Times" by "doing the SMART thing."

Theorem 4.2
In an M/GI/1 queue, for $P \in$ SMART:

$$E[T]^{SRPT} \leq E[T]^{P} \leq 2E[T]^{SRPT}$$

We defer details of the proof of Theorem 4.2 to Section 4.1.4.2.

Theorem 4.2 serves as a validation of the SMART class. It proves that all SMART policies behave like SRPT in a very strong sense. Further, this result is of practical importance because it guarantees that all SMART policies, even ones with strange time-varying priority schemes, provide near optimal mean response times regardless of the load and regardless of the service distribution. Thus, even as $\rho \to 1$ and $E[X^2] \to \infty$ the $E[T]^P$ for all $P \in$ SMART will stay within a factor of two of optimal. In contrast, in this



(a) $E[T(x)]/x$ (b) $E[T](1-\rho)$

Figure 4.4: *This figure illustrates the bounds on $E[T(x)]$ and $E[T]$ under the class of* SMART *policies in Theorems 4.4 and 4.3. The shaded area indicates the response times attainable using* SMART *policies. In addition, the behaviors of the two most common* SMART *policies,* SRPT *and* PSJF *, are illustrated. In both plots the service distribution is taken to be Exponential with mean 1, and the load in (a) is 0.7.*

setting $E[T]^{FCFS}$ and $E[T]^{PS}$ may be arbitrarily larger than the optimal $E[T]$.

Remark 4.1 *It is important to note that Theorem 4.2 is tight. Clearly, the lower bound is tight since* SRPT $\in$ SMART *and $E[T]^{SRPT}$ is optimal. To see that the upper bound is tight, we consider the M/D/1 setting where all jobs have size b. In this setting,* SRPT *is actually doing* FCFS*, thus*

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

Further, in this setting, PLCFS $\in$ SMART *(this only holds in the M/D/1 setting). Since $E[T]^{M/D/1/PLCFS} = b/(1-\rho)$, we can see that*

$$\lim_{\rho\to 1} \frac{E[T]^{PLCFS}}{E[T]^{SRPT}} = \lim_{\rho\to 1} \frac{1}{1-\rho/2} = 2.$$

This observation provides a nice intuitive understanding of why the factor of 2 arises in Theorem 4.2 – it comes from the freedom in how jobs of very similar sizes are scheduled.

Though Theorem 4.2 is simple and tight, it provides little intuition about how parameters such as load and job size variability influence the mean response times of SMART policies. The problem is that $E[T]^{SRPT}$ is quite complicated to express, as we discussed in Section 3.2.4. Though we have proven simple bounds on the behavior of SRPT in Theorem 3.10, naively combining these bounds with Theorem 4.2 results in bounds that are very loose in many cases, thus it is important to approach bounding the response times of SMART policies directly.

Theorem 4.3
Consider an M/GI/1 queue with $P \in$ SMART. Let X_1 and X_2 be independent random job sizes. Let K satisfy $\lambda m_2(x) \leq Kx\rho(x)$ and $h(\rho) = \left(\frac{1}{\rho}\right)\log\left(\frac{1}{1-\rho}\right)$. Then,

$$h(\rho)E[X] \leq E[T]^P \leq \left(\frac{K-1}{2(1-\rho)} + \frac{\rho}{4}\left(1 + \frac{E[(X_1 \wedge X_2)^2]}{E[X]^2}\right) + \left(\frac{K-3}{2}\right)h(\rho)\right)E[X]$$

We defer the proof of Theorem 4.3 to Section 4.1.4.3, however we provide a simple illustration of the bounds in Figure 4.4. Further, it is important to point out that the lower bound in Theorem 4.3 is the same as the lower bound we proved for SRPT in Theorem 3.10, thus it becomes tight when we consider Pareto distributions. Further, the upper bound in Theorem 4.3 becomes tight as the distribution approaches a deterministic distribution.

Though we defer the details of the proofs of Theorems 4.2 and 4.3 to Sections 4.1.4.2 and 4.1.4.3; the method we use to prove these theorems is the following. We first bound the conditional response time, $T(x)$, using the next theorem (Theorem 4.4) and then decondition to obtain bounds on the overall response time. Thus, Theorem 4.4 acts as a stepping stone toward obtaining Theorem 4.2. However, Theorem 4.4 is important in its own right as well: it will be of primary importance when studying fairness later in Chapter 7.

Theorem 4.4
In an M/GI/1 queue, for all x and all $P \in$ SMART,

$$R(x)^{SRPT} + W(x)^{PSJF} \leq_{st} T(x)^P \leq_{st} R(x)^{PSJF} + W(x)^{SRPT}$$

Further,

$$\int_0^x \frac{dt}{1-\rho(t)} + \frac{\lambda m_2(x)}{2(1-\rho(x))^2} \quad \leq \quad E[T(x)]^P \leq \frac{x}{1-\rho(x)} + \frac{\lambda \widetilde{m_2}(x)}{2(1-\rho(x))^2}$$

The proof of this theorem is deferred to Section 4.1.4.1.

Remark 4.2 *It follows from combining Theorem 4.4 and Theorem 3.20 that $T(x)^P \leq_{st} T(x)^{FB}$ for all $P \in$ SMART.*

Observe that the bounds in Theorem 4.4 are a combination of the response times of the two most common SMART policies, SRPT and PSJF. Intuitively, this is not surprising. PSJF maximizes residence time among SMART policies because it allows the greatest number of arrivals to preempt service, and minimizes the waiting time because it does not allow any jobs with larger original size and smaller remaining size to obtain higher priority. Further, SRPT minimizes the residence time among SMART policies because only arriving jobs with original size smaller than the tagged jobs remaining size are given higher priority, and maximizes waiting time among SMART policies because it allows the greatest amount of work already in the system to finish before an arriving job. This observation illustrates the tightness of the bounds, and the proof of the theorem formalizes these ideas. Note that, though the proof of Theorem 4.4 for SMART presented here is quite involved, a much simpler proof is possible if Theorem 4.4 is proven instead for only for $P \in$ SMART$*$.

It is important to note that, though SMART policies all have quite similar response time behavior, SMART policies can differ significantly in their performance on other metrics. For instance, in Chapter 7 we will see that different SMART policies can behave differently with respect to fairness and predictability measures. Thus, one way to view the SMART class is as a starting point for picking a scheduling policy when you want to optimize both $E[T]$ and some other secondary metric of interest.

4.1.4.1 Proof of Theorem 4.4

In this section we will prove Theorem 4.4. We start with two lemmas. Let D_x^P denote the time average portion of work in the system at the Poisson arrival of a tagged job of size x that will complete under P before the tagged job does. Note that under SMART policies, D_x^P may in general depend on the behavior of the system after x arrives.

Lemma 4.5
In an M/GI/1 queue, for all x and all $P \in$ SMART,

$$D_x^{PSJF} \leq_{st} D_x^P \leq_{st} D_x^{SRPT}.$$

Proof. For the lower bound, let Q_x^P be the work in the queue made up by jobs with original size $\leq x$. Since PSJF devotes the full server to jobs of size $\leq x$ (when such jobs exist), we have

$$D_x^{PSJF} = Q_x^{PSJF} \leq Q_x^P.$$

The Bias Property implies that $Q_x^P \leq_{st} D_x^P$, and the result follows.

The proof of the upper bound is much more involved. Consider a tagged job j_x of size x arriving to the steady state system at time t_{j_x}. In order to analyze D_x^P, we track "contributing" work. At time t_{j_x}, the "contributing" work will be equal to D_x^P.

We define "Small Contributors" as all jobs of original size $< x$. For SMART policies, all Small Contributors in the system at time t_{j_x} serve ahead of j_x and thus add their remaining size at time t_{j_x} to the response time of job j_x. We say a Small Contributor is "contributing" the whole time that it is in the system and its "contribution" at any time is its remaining size. Thus, at time t_{j_x} every Small Contributor in the system is "contributing" the amount of work it adds to the response time of j_x.

We define "Large Jobs" as all jobs of original size $\geq x$. For all SMART policies, at most *one* Large Job, c, in the system at time t_{j_x} can add to the response time of job j_x; call job c a "Large Contributor." The uniqueness of c is proven in Lemma 4.6. We say that Large Job c becomes a Large Contributor when r_c becomes x. The amount job c adds to the response time of j_x is the remaining size of c at time t_{j_x}, which can be at most x. We consider c to be "contributing" r_c at all times when $r_c \leq x$. Thus, at time t_{j_x}, c is "contributing" the amount it adds to the response time of j_x.

We now limit our discussion to times $t \in [t_0, t_{j_x}]$ where t_0 is the last moment before j_x arrives that no job is "contributing." So, at t_0 either a Large Job becomes a Large Contributor, a Small Contributor arrives, or j_x arrives ($t_0 = t_{j_x}$). Further, for $t \in (t_0, t_{j_x})$, there is always either a Large or Small Contributor in the system. We refer to t_0 as the beginning of the "contribution period" into which j_x arrives.

We define $D_x^P(t)$ as the total work being contributed by Small and Large Contributors in the system at time t under P, where, as usual, the definition of Contributors is relative to job j_x arriving at time t_{j_x}. It is

important to point out that $D_x^P(t_{j_x}) = D_x^P$, i.e. the work contributing when j_x arrives is exactly the work that will serve ahead of j_x.

There are three types of periods into which j_x can arrive:

Type (a) A period idle of contributing jobs (i.e. $t_{j_x} = t_0$). Thus, job j_x sees $D_x^P(t_0) = 0$ for all $P \in$ SMART.

Type (b) A contribution period started by a Small Contributor b arriving and contributing $s_b < x$. Thus, $D_x^P(t_0) = s_b$ under all $P \in$ SMART.

Type (c) A contribution period started by a Large Job c becoming a Large Contributor and contributing x, i.e. r_c becomes x at time t_0. Thus, $V_x^P(t_0) = x$ under all $P \in$ SMART.

Let p_a^P, p_b^P, and p_c^P be the time-average probability of j_x arriving into a contribution period of type (a), (b), and (c) respectively under policy $P \in$ SMART. Recall that j_x is a Poisson arrival, so PASTA applies. Notice that these are the only legal possibilities for what can occur at time t_0 and that there is zero probability of more than one event happening.

Claim (1) $p_a^P \geq p_a^{SRPT}$, $p_b^P \geq p_b^{SRPT}$, and thus $p_c^P \leq p_c^{SRPT}$.

CLAIM (1): We divide the proof of claim (1) into two parts.

Part (a): We will first show that p_a^P is minimized under SRPT. Under SRPT, the system is idle of Small and Large Contributors exactly when there are no jobs in the system having remaining size $< x$. Using PASTA and the fact that j_x is a Poisson arrival, this gives that $p_a^{SRPT} = 1 - \widetilde{\rho}(x)$, i.e. the time-average idle time in a system having arrival rate λ and job sizes $X_x = \min(x, X)$. All $P \in$ SMART are also guaranteed to be idle of Small and Large Contributors when there are no jobs in the system with remaining size $< x$; however they may also be idle of Contributors when there *exist* jobs in the system with remaining size $< x$ if these jobs will not receive priority over j_x when j_x arrives. Thus, $p_a^P \geq p_a^{SRPT}$.

Part (b): We now prove that $p_b^P \geq p_b^{SRPT}$. A type (b) period is started when a Small Contributor arrives into a system idle of contributors. Small Contributors arrive independently of P according to a Poisson process with rate $\lambda F(x)$. Thus, $p_b^P \geq p_b^{SRPT}$ because SRPT is the least likely $P \in$ SMART to be idle of contributing jobs (from part (a)). It follows that $p_c^P \leq p_c^{SRPT}$ since $p_a^P \geq p_a^{SRPT}$ and $p_b^P \geq p_b^{SRPT}$. We can also see that $p_c^P \leq p_c^{SRPT}$ directly by noting that *all* Large Jobs can become Large Contributors and thus start type (c) periods under SRPT. We are now finished with the proof of claim (1).

Consider what j_x sees when it arrives into the system. With probability $p_a^P \geq p_a^{SRPT}$, j_x sees a type (a) period, and with probability $p_b^P + p_c^P = 1 - p_a^P \leq 1 - p_a^{SRPT} = p_b^{SRPT} + p_c^{SRPT}$, j_x sees a contribution period. Thus, in proving $D_x^P \leq_{st} D_x^{SRPT}$ it suffices analyze the $D_x^P(t_{j_x})$ in a contribution period, i.e. given j_x arrives into a type (a) or (b) period.

We will complete the proof of the theorem by showing that

Claim (2) $D_x^P(t_0) \leq_{st} D_x^{SRPT}(t_0)$, i.e. the initial jump of the contribution period is smaller under P than under SRPT.

Claim (3) For $t \in (t_0, t_{j_x})$, $D_x^P(t)$ is always reduced at the full service rate and increases only at the Poisson arrivals of Small Contributors under all $P \in$ SMART.

Claim (4) $D_x^P(t_{j_x}) \leq_{st} D_x^{SRPT}(t_{j_x})$ for Poisson arrival j_x during a contribution period.

CLAIM (2): Note that the initial contribution in a type (b) period is at most the initial contribution in a type (c) period. The claim then follows because $p_b^P \geq p_b^{SRPT}$ and $p_c^P \leq p_c^{SRPT}$.

CLAIM (3): To prove claim (3), notice that, under all $P \in$ SMART, Large Jobs that are not Large Contributors cannot receive service given $t \in (t_0, t_{j_x})$ (Lemma 4.6). Thus, all $P \in$ SMART reduce $D_x^P(t)$ at the maximal rate for all t, i.e. the full service rate is devoted to contributing jobs. Further, under all $P \in$ SMART, arriving Large Jobs cannot become Large Contributors after time t_0 (Lemma 4.6). Thus, the only arrivals that affect $D_x^P(t)$ are Small Contributors, which arrive according to a Poisson process of rate $\lambda F(x)$ under all $P \in$ SMART, including SRPT.

CLAIM (4): To prove claim (4) we will analyze the contributing work that j_x sees upon arrival into a contribution period under $P \in$ SMART and SRPT . Note that j_x arriving into a contribution period under P sees $D_x^P | (D_x^P > 0)$ contributing work. By claim (2), $D_x^P(t_0) \leq_{st} D_x^{SRPT}(t_0)$. Thus, there is some random time $t^* > t_0$ when $D_x^P(t_0) \stackrel{\mathrm{d}}{=} D_x^{SRPT}(t^*)$ for the first time. If $t_{j_x} \geq t^* \geq t_0$ under SRPT then $D_x^P(t_{j_x}) = D_x^{SRPT}(t_{j_x})$ (by claim (3) and the definition of t^*). If $t_0 < t_{j_x} < t^*$, then j_x sees a stochastically larger amount of contributing work (by the definition of t^*). So, $D_x^P(t_{j_x}) \leq_{st} D_x^{SRPT}(t_{j_x})$.
□

We now prove a lemma used in the above proof in order to characterize the effect of the Consistency and Transitivity properties.

Lemma 4.6

There is at most one Large Contributor in the system at any time, where a Large Contributor is defined with respect to job j_x. Further, no Large Jobs that are not Large Contributors can receive service while a Large or Small Contributor is in the system.

Proof. Suppose b becomes a Large Contributor at time t_1 and is the only Large Contributor in the system at t_1. We will show that no other Large Jobs can become Large Contributors while b is in the system.

Note that a Large Job must be receiving service when it becomes a Large Contributor, and thus a Large Job can only become a Large Contributor when the system is idle of Small Contributors due to the Bias Property.

We first show that a Large Job $c \neq b$, in the system at time t_1, cannot become a Large Contributor. Note that c, by definition, is not a Large Contributor at t_1, and thus must receive service in order to become a Large Contributor. Further, c is in the queue at t_1 and b is at the server. So c can never receive service while b is in the system because of the Consistency Property.

To complete the proof, we will show that a Large Job c that arrives after t_1 cannot become a Large Contributor. Again, c must receive service before time t_{j_x} in order to become a Large Contributor. Further, c must be in the system at time t_{j_x} to be a Large Contributor. However, upon arrival $s_c = r_c > x$, so if job c runs ahead of job b, the Consistency Property gives job c priority over job b. Further, since c is in the system at time t_{j_x}, b cannot receive service until then, and thus the Transitivity Property will give j_x priority over b when j_x arrives. This contradicts the fact that b is a Large Contributor. Thus c can never run ahead of b, and c can never become a Large Contributor.
□

We can now prove Theorem 4.4.

Proof of Theorem 4.4. Let $P \in$ SMART. We break up the response time for a tagged job j_x of size x arriving to the steady state system into: (i) D_x^P, the portion of the work in the system when j_x arrives that will complete under P before j_x completes, (ii) x work made up by j_x, and (iii) the work done by P on jobs that arrive after j_x arrives.

Notice that the Bias property guarantees that (iii) includes, at most, all arriving jobs of size less than x. Thus, we can stochastically upper bound $T(x)^P$ with the length of a busy period that is started by $x + D_x^P$ work and made up only of arrivals having size smaller than x:

$$T(x)^P \leq_{st} B_x(x + D_x^P) \stackrel{\mathrm{d}}{=} B_x(x) + B_x(D_x^P). \tag{4.1}$$

The upper bound now follows from Lemma 4.5 and the facts that $W(x)^{SRPT} \stackrel{\mathrm{d}}{=} B_x(D_x^{SRPT})$ and $R(x)^{PSJF} \stackrel{\mathrm{d}}{=} B_x(x)$.

For the lower bound, note that due to the Consistency Property, the D_x^P work must be completed before j_x receives service. Thus, by the Bias Property, all jobs of size $< x$ that arrive during the time the D_x^P work is being completed are guaranteed to be served before j_x. So, the waiting time of j_x is $B_x(D_x^P)$. Further, once j_x begins service, the Bias property implies that, at minimum, all arrivals having size smaller than the remaining size of j_x have priority over j_x. Since the residence time under SRPT consists exactly of these jobs, we have

$$T(x)^P \geq_{st} R(x)^{SRPT} + B_x(D_x^P).$$

Applying Lemma 4.5 and the fact that $W(x)^{PSJF} \stackrel{\mathrm{d}}{=} B_x(D_x^{PSJF})$ completes the proof.
□

4.1.4.2 Proof of Theorem 4.2

In this section, we will prove that all SMART policies are 2-competitive and that PSJF, a common SMART policy, is $3/2$-competitive. Remember that the lower bound on SRPT serves as a lower bound on the mean response time of any policy in the SMART class since SRPT is known to be optimal with respect to overall mean response time. Also, recall the definition of $E[W_2]$ from Section 3.2.4:

$$E[W_2] = E[W]^{SRPT} - E[W]^{PSJF} = \int_0^\infty \frac{\lambda x^2 f(x)\overline{F}(x)}{2(1-\rho(x))^2} dx$$

Proof of Theorem 4.2. It is clear that $E[T]^{SRPT} \leq E[T]^P$ because SRPT is optimal with respect to mean response time. Thus we need only show the upper bound.

We will start the proof of the upper bound by studying $E[W_2]$. Although we cannot evaluate $E[W_2]$ exactly, we can show that the mean response time of PSJF is exactly $E[W_2]$ away from optimal. In particular,

using Lemma 3.11, we have:

$$\begin{aligned} E[T]^{SRPT} &= E[R]^{SRPT} + E[W]^{PSJF} + E[W_2] \\ &= \frac{1}{2}E[R]^{PSJF} + \frac{1}{2}E[R]^{SRPT} + E[W]^{PSJF} \\ &= E[T]^{PSJF} - \frac{1}{2}E[R]^{PSJF} + \frac{1}{2}E[R]^{SRPT} \\ &= E[T]^{PSJF} - E[W_2] \end{aligned}$$

Now, combining the above with Lemma 3.11 gives that

$$E[T]^{P} \leq E[T]^{SRPT} + 2E[W_2]$$

Finally, using Lemma 3.12, we have

$$\begin{aligned} E[T]^{P} &\leq E[T]^{SRPT} + 2E[W_2] \\ &= E[T]^{SRPT}\left(1 + 2\frac{\frac{1}{2}E[W_2] + \frac{1}{2}E[W_2]}{E[T]^{SRPT}}\right) \\ &\leq E[T]^{SRPT}\left(1 + \frac{E[W]^{PSJF} + E[W_2]}{E[T]^{SRPT}}\right) \\ &\leq 2E[T]^{SRPT} \end{aligned}$$

□

4.1.4.3 Proof of Theorem 4.3

In this section, we derive the bounds on $E[T]$ under SMART policies in Theorem 4.3. Since the lower bound on SRPT in Theorem 3.10 serves as a lower bound on the mean response time of any policy in the SMART class, we need only prove the upper bound in the theorem.

Proof of Theorem 4.3.

To begin, we will prove a lower bound on the residence time of SRPT. Recall that the p.d.f. of $\min(X_1, X_2)$ is $f_{min}(x) = 2f(x)\overline{F}(x)$. Thus

$$\begin{aligned} E[R]^{SRPT} &= \int_0^\infty f(x)\left(x + \int_0^x \frac{\rho(t)}{1-\rho(t)}dt\right)dx \\ &\geq \int_0^\infty f(x)\left(x + \int_0^x \rho(t)dt\right)dx \\ &= E[X] + \frac{1}{\lambda}\int_0^\infty \rho'(x)\rho(x)dx - \lambda\int_0^\infty t^2 f(t)\overline{F}(t)dt \end{aligned}$$

Using Theorem 3.7, Lemma 3.11, and the above, we have:

$$\begin{aligned}
E[T]^P &\leq E[T]^{PSJF} + E[W_2] \\
&\leq \frac{K-3}{2\lambda}\log(1-\rho) + \frac{K}{2}E[T]^{PS} - \frac{1}{2}E[R]^{SRPT} \\
&\leq \frac{K-3}{2\lambda}\log(1-\rho) + \frac{K}{2}E[T]^{PS} \\
&\quad -\frac{1}{2}\left(E[X] + \frac{\rho^2}{2\lambda} - \frac{\lambda}{2}E[\min(X_1, X_2)^2]\right) \\
&= \left(\frac{\rho}{4} + \frac{K-1}{2} - \frac{\rho^2}{4} + (1-\rho)\frac{\lambda E[\min(X_1, X_2)^2]}{4E[X]}\right. \\
&\quad \left. + \left(\frac{K-3}{2}\right)\left(\frac{1-\rho}{\rho}\right)\log(1-\rho)\right) E[T]^{PS}
\end{aligned}$$

This final form is equivalent to the form in Theorem 4.3, which completes the proof. □

4.2 Generalizing the SMART class

Implicit in the definition of the SMART class is a tension between the *breadth* of the class and the *tightness* of the results provable about the class. In particular, there is a tension between the strength of bias towards small jobs required in the Bias Property and the goal of showing that all policies are within a constant factor of the optimal mean response time, i.e. are *competitive* with respect to mean response time. In this section, we will explore this question further by developing an understanding of how weakening the bias towards small jobs affects the *competitive ratio* of the class with respect to mean response time, i.e. the k such that $E[T] \leq kE[T]^{SRPT}$.

This tension between the breadth of the SMART class and the tightness of results provable about SMART policies is of huge practical importance since one fundamental goal of studying classifications is to bridge the gap between the results provided by theoreticians and the needs of practitioners. This gap results from (i) implementation restrictions and (ii) concerns about metrics other than mean sojourn time (e.g. avoiding starvation of large jobs). As defined so far, SMART does a nice job of bridging (ii), but does little to help bridge (i). In particular, two key implementation restrictions that SMART does not bridge are the following. First, the overhead involved in distinguishing between an infinite number of different priority classes typically causes system designers to discretize policies such as SRPT and PSJF so that they use only a small number (5-10) of priority classes [96, 180]. Second, in many cases information about the service demands (sizes) of jobs is inexact. For instance, when serving static content, web servers have exact knowledge of the sizes of the files being served, but have inexact knowledge of network conditions. Thus, the web server only has an estimate of the true service demands [131, 182]. In this section we will explore how to broaden SMART to handle these implementation restrictions while still keeping the class tight enough to guarantee that it is competitive with respect to mean response time.

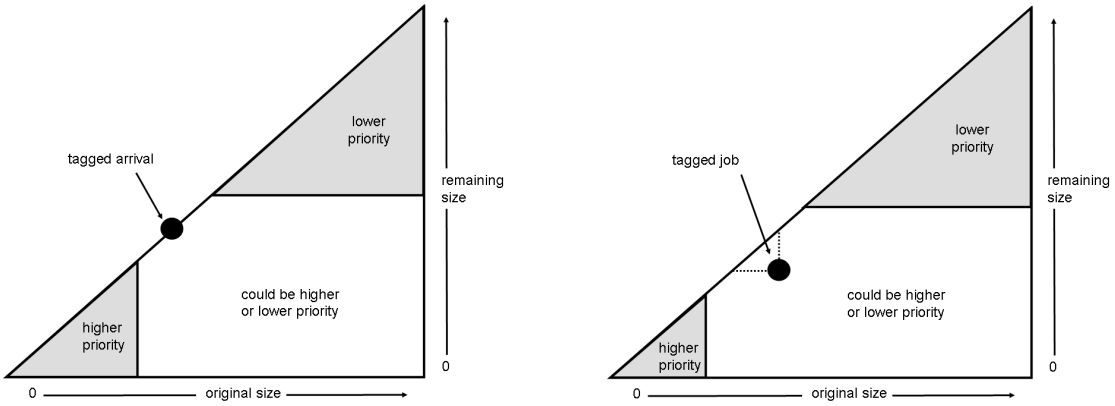


Figure 4.5: *This diagram illustrates the priority structure induced by the Bias Property in the definition of* SMART$_\epsilon$ *(Definition 4.3).*

4.2.0.4 The tension between breadth and tightness

To begin our exploration, let us consider the case of non-preemptive policies. Is it possible to include non-preemptive policies that bias towards small job sizes, e.g. SJF, in a generalized SMART class and still provide a competitive guarantee under all service distributions and all loads? It is easy to see that that answer is no: any non-preemptive policy must have infinite $E[T]$ if $E[X^2]$ is infinite, while $E[T]^{SRPT}$ is finite whenever $E[X] < \infty$. Thus, a generalized SMART class cannot both include non-preemptive policies and provide a competitive guarantee on mean response time.

Similarly, the SMART class cannot be generalized to include policies blind to job sizes while still providing a competitive guarantee on mean response time. To see this, recall that for the M/M/1 queue all blind scheduling policies have the same mean response time, $E[T] = E[X]/(1-\rho)$. But, in this setting $E[T]^{SRPT}(1-\rho) \to 0$ as $\rho \to 1$ (see Theorem 3.14). Thus, there is no $k < \infty$ such that $E[T]^P \leq kE[T]^{SRPT}$ under all service distributions and all loads for any blind policy P.

Summarizing the discussion so far, we can conclude that, in an M/GI/1 queue, for a policy to be competitive with respect to mean response time it must (i) use job size information to schedule and (ii) be preemptive. Thus, in studying the tradeoff between the breadth (in terms of policies) and the tightness (in terms of mean response time) within the SMART class, we can limit ourselves to discussing preemptive policies that use job size information. We capture a wide range of such policies in the definition of SMART$_\epsilon$, a generalization of the SMART class.

4.2.1 Defining SMART$_\epsilon$

We now formally define the class of SMART$_\epsilon$ policies.

Definition 4.3 *For all $x \in [0, \infty)$, define $x_\epsilon = \epsilon(x)$ for some non-decreasing function ϵ where $\epsilon(x) \geq x$. A work conserving policy $P \in$*SMART$_\epsilon$ *if it obeys the following properties at all times.*

Bias Property: *If $s_a = x$ and $r_b > x_\epsilon$, then job a has priority over job b.*

Consistency Property: *If job a ever receives service while job b is in the system, then at all times thereafter job a has priority over job b.*

Transitivity Property: *If an arriving job b preempts job c, then thereafter, until job c receives service, every arrival a with size $s_a = x$ such that $x_\epsilon < s_b$ is given priority over job c.*

The SMART$_\epsilon$ class is defined in an almost parallel way to the SMART class. In particular, the SMART class is a subclass of SMART$_\epsilon$ that can be obtained by setting $x_\epsilon = x$. The parameter x_ϵ provides a formal way to capture the effect of "weakening the bias towards small jobs.' If, for example, we think of $\epsilon(x) = (1+\sigma)x$, as σ grows the bias towards small jobs sizes required of SMART$_\epsilon$ policies decreases. Thus, by varying ϵ, we can study the impact of broadening the class on the competitive ratio of the class. Refer to Figure 4.5 for an illustration of the SMART$_\epsilon$ Bias Property.

As with the definition of the original SMART class, each of the three properties in the definition of SMART$_\epsilon$ formalizes a notion of "smart" scheduling. The Weak Bias Property guarantees that the job being run at the server has remaining size not too much larger than the original size of every job in the system, which formalizes the idea of "prioritizing small jobs." The Consistency and Transitivity Properties ensure coherency in the priority structure enforced by the Bias Property. In particular, the Consistency Property prevents time-sharing by guaranteeing that after job a is chosen to run ahead of b, job b will never run ahead of job a. This makes intuitive sense because all SMART$_\epsilon$ policies are based on the heuristic of giving priority to small jobs, and as job a receives service, it can only get smaller. Finally, the Transitivity Property guarantees that SMART$_\epsilon$ policies do not second guess themselves: if an arrival a is determined to be "smaller" than job b in some sense (i.e. is given priority over job b), future arrivals with smaller size than a should also considered "smaller" than b until b receives service.

4.2.2 Examples of SMART$_\epsilon$ policies

Of course, the SMART$_\epsilon$ class includes all SMART policies. Thus, it includes SRPT and PSJF in addition to a wide range of hybrids of these policies, including policies with time-varying behavior. As we have already discussed, the inclusion of these time-varying policies is particularly important because it allows system designers to use the class in order to perform online multi-objective optimization, which is extremely useful in web applications where the service distribution is time-varying and thus the optimal scheduling policy often is not static.

In addition to SMART policies, SMART$_\epsilon$ includes many practical policies that are excluded from SMART because of the rigidity of the original Bias Property. First, notice that SMART$_\epsilon$ can include policies that have only a finite number of priority levels. In particular, it can include preemptive threshold based policies where there are a finite number of thresholds $0 = t_1, \ldots, t_n = \infty$ and a job of size s is assigned priority $p(s,r) = i$ if $p_1(s,r) \in [t_i, t_{i+1})$ for some static priority function $p_1(s,r) \in$ SMART such as $p_1(s,r) = s$ (i.e. PSJF). The inclusion of these policies is of particular practical importance because in many cases system designers simplify implementations by using only 5-10 priority levels.

In addition to including threshold based policies, SMART$_\epsilon$ includes SMART policies that are being run using inexact job size information. The inclusion of these policies is again of practical interest. For many applications information about the service demands of jobs is inexact. For instance, when serving static content, web servers have exact knowledge of the sizes of the files being served, but have inexact knowledge of network conditions. Thus, web servers have only an estimate of the true service demands.

When one performs SRPT or some other SMART policy on job size estimates, the resulting policy is not in SMART and is difficult to study directly; however the resulting situation does fall into SMART_ϵ for a suitable $\epsilon(x)$. As an example, if the inexact job sizes are a result of a time-varying service capacity (such as a web server with only an estimate of network conditions), taking $\epsilon(x) = (1+\sigma)x$ models the situation where the maximum change in service rate is σ. Notice, that this does not assume any distribution on the job size estimates; thus the estimates may be adversarial.

4.2.3 Bounding response times for SMART_ϵ policies

Our goal in the remainder of this section is to characterize the relationship between the breadth of SMART_ϵ (i.e. x_ϵ) and the tightness of the bounds on mean response time (i.e. the competitive ratio of SMART_ϵ policies with respect to mean response time).

As with the analysis of SMART policies, we approach the analysis of mean response time by first analyzing the behavior of the conditional response time. Thus, we start by extending the stochastic bounds on SMART in Theorem 4.4 to SMART_ϵ.

Theorem 4.7
In an M/GI/1 queue, for all $P \in \mathsf{SMART}_\epsilon$,

$$T(x)^P \leq_{st} B_{x_\epsilon}(x + D_{x_\epsilon}^{SRPT})$$

Thus,

$$E[T(x)]^P \leq \frac{x}{1-\rho(x_\epsilon)} + \frac{\lambda \widetilde{m_2}(x_\epsilon)}{2(1-\rho(x_\epsilon))^2}$$

Proof. Let $P \in \mathsf{SMART}_\epsilon$. We break up the response time for a tagged job j_x of size x arriving to the steady state system into: (i) D_x^P, the portion of the work in the system when j_x arrives that will complete under P before j_x completes, (ii) x work made up by j_x, and (iii) the work done by P on jobs that arrive after j_x arrives.

Notice that the Bias Property guarantees that (iii) includes, at most, all arriving jobs of size less than x_ϵ. Thus, we can stochastically upper bound $T(x)^P$ with the length of a busy period that is started by $x + D_x^P$ work and made up only of arrivals having size smaller than x_ϵ:

$$T(x)^P \leq_{st} B_{x_\epsilon}(x + D_x^P). \tag{4.2}$$

The upper bound now follows from the observation that $D_x^P \leq_{st} D_{x_\epsilon}^{SRPT}$, which follows from an argument that parallels the proof of Lemma 4.5 for the SMART class.
□

Using these stochastic bounds on the conditional response time, we can decondition to obtain bounds on the overall mean response time. These bounds on the overall mean response time characterize the relationship between x_ϵ and the competitive ratio of SMART_ϵ.

Theorem 4.8
Consider an M/GI/1 queue with $P \in$ SMART$_\epsilon$. If there exists an $\sigma \geq 0$ such that $x_\epsilon \leq (1+\sigma)x$ for all x and a $0 \leq \gamma < 1$ such that $\rho(x_\epsilon) - \rho(x) \leq \gamma(\rho - \rho(x))$, then

$$E[T]^{SRPT} \leq E[T]^P \leq 2\left(\frac{1+\sigma}{1-\gamma}\right)^2 E[T]^{SRPT}$$

That is, P is $2\left(\frac{1+\sigma}{1-\gamma}\right)^2$-competitive with respect to mean response time.

Note that if we take $\gamma = \sigma = 0$ we get back Theorem 4.2, which says that all SMART policies are 2-competitive with respect to mean response time. Before proving Theorem 4.8, we first prove the following two simple lemmas.

Lemma 4.9
If there exists a $0 \leq \gamma < 1$ such that $\rho(x_\epsilon) - \rho(x) \leq \gamma(\rho - \rho(x_\epsilon))$, then

$$\frac{1-\rho(x)}{1-\rho(x_\epsilon)} \leq \frac{1}{1-\gamma}$$

Proof.

$$\begin{aligned}
\frac{1-\rho(x)}{1-\rho(x_\epsilon)} &= 1 + \frac{\rho(x_\epsilon) - \rho(x)}{1-\rho+\rho-\rho(x_\epsilon)} \\
&\leq 1 + \frac{\rho(x_\epsilon) - \rho(x)}{\rho - \rho(x_\epsilon)} \\
&= 1 + \frac{1}{\frac{\rho-\rho(x)}{\rho(x_\epsilon)-\rho(x)} - 1} \\
&\leq 1 + \frac{1}{\frac{1}{\gamma} - 1} \\
&= \frac{1}{1-\gamma}
\end{aligned}$$

□

Lemma 4.10
If there exists a $\sigma \geq 0$ such that $x_\epsilon \leq (1+\sigma)x$ for all x, then

$$\frac{\widetilde{m_2}(x_\epsilon)}{\widetilde{m_2}(x)} \leq (1+\sigma)^2$$

Proof.

$$\begin{aligned}\frac{\widetilde{m_2}(x_\epsilon)}{\widetilde{m_2}(x)} &= 1 + \frac{\int_x^{x_\epsilon} t\overline{F}(t)dt}{\int_0^x t\overline{F}(t)dt} \\ &\leq 1 + \frac{\overline{F}(x)\int_x^{x_\epsilon} tdt}{\overline{F}(x)\int_0^x tdt} \\ &\leq 1 + \frac{x_\epsilon^2 - x^2}{x^2} \\ &\leq (1+\sigma)^2\end{aligned}$$

□

We now prove Theorem 4.8 using the above two lemmas.

Proof of Theorem 4.8. By Theorem 4.7, we have that

$$E[T(x)]^P \leq \frac{1-\rho(x)}{1-\rho(x_\epsilon)} E[R(x)]^{PSJF} + \left(\frac{1-\rho(x)}{1-\rho(x_\epsilon)}\right)^2 \left(\frac{\widetilde{m_2}(x_\epsilon)}{\widetilde{m_2}(x)}\right) E[W(x)]^{SRPT}.$$

We now apply Lemmas 4.9 and 4.10. Thus, we have that

$$\begin{aligned}E[T(x)]^P &\leq \frac{E[R(x)]^{PSJF}}{1-\gamma} + \left(\frac{1+\sigma}{1-\gamma}\right)^2 E[W(x)]^{SRPT} \\ &\leq \left(\frac{1+\sigma}{1-\gamma}\right)^2 \left(E[R(x)]^{PSJF} + E[W(x)]^{SRPT}\right)\end{aligned}$$

Noting that $E[R(x)]^{PSJF} + E[W(x)]^{SRPT}$ is the upper bound used in the proof of Theorem 4.2 completes the proof.
□

Theorem 4.8 presents two properties that x_ϵ must maintain in order for SMART$_\epsilon$ to be competitive with respect to mean response time. The first of these conditions is that $x_\epsilon \leq (1+\sigma)x$ for all x. This condition bounds how large a job can be and still get priority over a job of size x. The second condition in Theorem 4.8 is that $\rho(x_\epsilon) - \rho(x) \leq \gamma(\rho - \rho(x))$. This condition bounds the percentage of the load made up by jobs larger than size x that can have priority over a job of size x. Thus, the two conditions present complementary formulations of how much the Bias Property can be weakened: you can let significantly larger jobs have priority without paying a price in mean response time as long as the larger jobs do not make up too much load. The tradeoff between these two conditions will vary depending on the service distribution, but under the practical case of a Pareto distribution, the two conditions are actually equivalent, i.e. only the $x_\epsilon \leq (1+\sigma)x$ constraint is necessary. Recall that a Pareto distribution is defined by $\overline{F}(x) = (x_L/x)^\alpha$.

Corollary 4.11

Consider an M/GI/1 queue with $P \in$ SMART$_\epsilon$ and $X \sim Pareto(\alpha, x_L)$ having finite mean. If there exists

an $\sigma \geq 0$ such that $x_\epsilon \leq (1+\sigma)x$ then

$$E[T]^P \leq 2(1+\sigma)^{2\alpha} E[T]^{SRPT}.$$

Proof. Assume that $x_\epsilon \leq (1+\sigma)x$. We will show that this guarantees that

$$\rho(x_\epsilon) - \rho(x) \leq (1-(1+\sigma)^{1-\alpha})(\rho - \rho(x)).$$

This bound gives a γ for Theorem 4.8 such that

$$(1-\gamma) \leq (1+\sigma)^{1-\alpha},$$

which in combination with Theorem 4.8 completes the proof.

In order to prove that $\rho(x_\epsilon) - \rho(x) \leq (1-(1+\sigma)^{\alpha-1})(\rho - \rho(x))$, we calculate as follows:

$$\begin{aligned}
\frac{\rho(x_\epsilon) - \rho(x)}{\rho - \rho(x)} &= \frac{\int_x^{x_\epsilon} t f(t) dt}{\int_x^{\infty} t f(t) dt} \\
&= \frac{\frac{1}{x^{\alpha-1}} - \frac{1}{x_\epsilon^{\alpha-1}}}{\frac{1}{x^{\alpha-1}}} \\
&= 1 - \left(\frac{x}{x_\epsilon}\right)^{\alpha-1} \\
&\leq 1 - (1+\sigma)^{1-\alpha}
\end{aligned}$$

□

In addition to characterizing the effect of the rigidity of the bias towards small jobs in a policy, Theorem 4.8 and Corollary 4.11 can also be used to characterize the effect of inexact knowledge of job size information. Most prior work in this regard has used simulation experiments, e.g. [131], however SMART$_\epsilon$ allows easy back-of-the-envelope calculations. For example, if we have a Pareto service distribution with $\alpha = 1.1$ and job sizes can be estimated to within a factor of 50%, any SMART$_\epsilon$ policy (which includes running SRPT or PSJF on the job size estimates) still provides a mean response time within a factor of 4.88 of the optimal regardless of the load. This factor is quite small given that the variance is infinite, the bound holds for ρ arbitrarily close to 1, and adversarial errors in the estimates are allowed. As a comparison no finite constant bound is possible for many common policies, including FCFS and PS. Beyond this simple example, Figure 4.6 illustrates the effect of inexact job size information. Figure 4.6 (a) illustrates that the competitive ratio increases quickly as the worst case accuracy of the job size estimates increases. Figure 4.6 (b) illustrates that SMART$_\epsilon$ policies may perform worse than non-size based policies, e.g. PS, under low load, but at high load they can significantly outperform non-size based policies. Further, the penalty they pay at low load is exaggerated due to the fact that adversarial error sequences are allowed. In reality, the performance under low load of policies that use inexact estimates is not nearly as bad as the upper bound in Figure 4.6 (b) suggests.

Finally, we can also view Theorem 4.8 and Corollary 4.11 as a strong statement about the behavior

of policies with a finite number of priority classes. Though there has been a lot of work analyzing such policies [61, 120], it is still difficult to understand how far from optimal the mean response times of these policies are due to the fact that determining the optimal threshold values is typically not tractable (see [61] for a discussion). However, SMART$_\epsilon$ again provides a simple way to understand how close the mean response times of policies with only a finite number of priority classes are to optimal. In particular, the condition of $x_\epsilon \leq (1+\sigma)x$ provides a direct relation between the spacing of thresholds (i.e. the range of job sizes included in each priority level) and the guarantee on mean response time. Interestingly, there is a stark contrast between the behavior of policies that use only a finite number of priority classes under (i) unbounded service distributions and (ii) bounded service distributions.

First, let us consider the case of an unbounded service distribution, e.g. the Pareto. Clearly, no finite number of thresholds can satisfy $x_\epsilon \leq (1+\sigma)x$ for all x; thus, we cannot apply Theorem 4.8 or Corollary 4.11. This may seem like a restriction, but it turns out that under many unbounded distributions for any finite number of classes, $E[T]$ is not within a constant of $E[T]^{SRPT}$ as $\rho \to 1$. For instance, from Theorems 3.4 and 3.6 we can conclude that under a Pareto distribution no finite number of classes is enough to give $E[T]$ within a constant of $E[T]^{SRPT}$ for all loads.

Next, let us consider the case of bounded service distributions. Let x_U be the upper bound on the service distribution. Clearly in this case policies with a finite number of priority levels have $E[T]$ within a constant factor of $E[T]^{SRPT}$ since we have already proven that both are $\Theta(1/(1-\rho))$ in this setting. Unfortunately though, we cannot directly apply Theorem 4.8 to determine the constant. Choosing a finite number of thresholds such that $x_\epsilon \leq (1+\sigma)x$ is satisfied is no problem since the range of the distribution is finite. However, choosing a γ such that $\rho(x_\epsilon) - \rho(x) \leq \gamma(\rho - \rho(x))$ is a problem since for x in the highest priority class, we have $x_\epsilon = x_U$ Thus, $\rho(x_\epsilon) = \rho$ for such x and thus $\gamma = 1$, which means that Theorem 4.8 does not apply. This turns out to be easily remedied though. In particular, we need only adjust the requirements for x such that $x_\epsilon = x_U$. Let $t = \inf_x\{x_\epsilon \geq x_U\}$. Then, we have the following.

Corollary 4.12

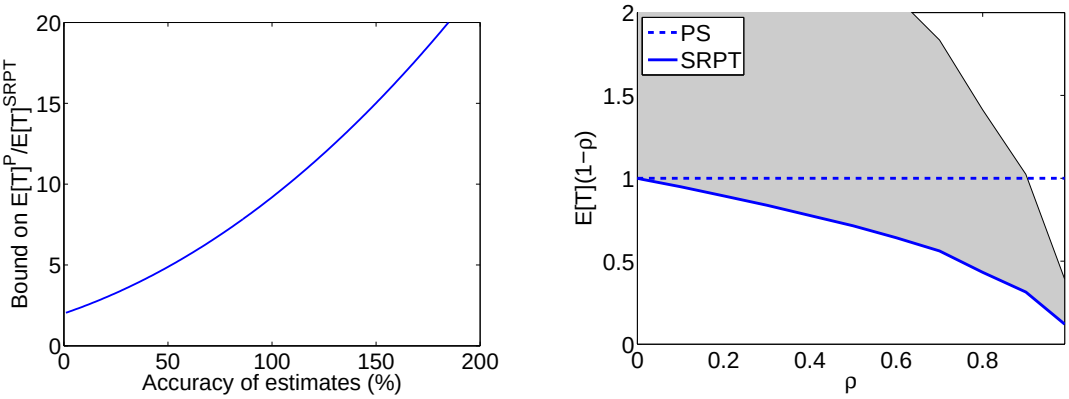
Consider an M/GI/1 queue with $P \in$ SMART$_\epsilon$ and a bounded service distribution. Let $\sigma \geq 0$ such that $x_\epsilon \leq (1+\sigma)x$ for all x and $0 \leq \gamma < 1$ such that $\rho(x_\epsilon) - \rho(x) \leq \gamma(\rho - \rho(x))$. If t is such that $1-\rho(t) \leq (1-\rho)/(1-\gamma)$ and $E[X^2]/\widetilde{m}_2(t) \leq (1+\sigma)^2$ then

$$E[T]^{SRPT} \leq E[T]^P \leq 2\left(\frac{1+\sigma}{1-\gamma}\right)^2 E[T]^{SRPT}$$

This corollary gives a simple bound on the behavior of threshold based policies in terms of (i) the spacing between the priority thresholds and (ii) the load in the lowest priority class. Thus, it provides back-of-the-envelope calculations determining how many classes are necessary in order to achieve $E[T]$ within a certain constant of optimal.

Proof. To prove the result, we need only make minor adjustments to Lemmas 4.9 and 4.10 in order to handle the case when $x_\epsilon = x_U$. To adjust Lemma 4.10 we simply note that for x such that $x_\epsilon = x_U$, $\widetilde{m}_2(x_\epsilon) = E[X^2]$ and, by assumption, for such x we have

$$\frac{\widetilde{m}_2(x_\epsilon)}{\widetilde{m}_2(x)} \leq \frac{E[X^2]}{\widetilde{m}_2(t)} \leq \gamma$$



(a) Accuracy of estimates (σ) vs. competitive ratio

(b) ρ vs $E[T]$

Figure 4.6: *An illustration of the impact of the accuracy of the estimates used in* SMART$_\epsilon$ *policies. Plot (a) shows the relationship between the accuracy of the estimates and the worst case competitive ratio of the resulting policy. Plot (b) shows the attainable response times under* SMART$_\epsilon$ *policies as a function of load.* SMART$_\epsilon$ *policies all have response time within the shaded region. Notice that the benefit of using job size estimates is very dependent on the load. The accuracy assumed in (b) is 50%. In both cases, the service distribution is Pareto with mean 1 and* $\alpha = 1.2$.

We will now adjust Lemma 4.9. Note that for x such that $x_\epsilon = x_U$, we have that $\rho(x_\epsilon) = \rho$. Thus

$$\begin{aligned}
\frac{1-\rho(x)}{1-\rho(x_\epsilon)} &= 1 + \frac{\rho(x_\epsilon) - \rho(x)}{1-\rho} \\
&\leq 1 + \frac{\rho - \rho(t)}{1-\rho} \\
&= \frac{1-\rho(t)}{1-\rho} \\
&\leq \frac{1}{1-\gamma}
\end{aligned}$$

Now, we complete the proof by combining the above with Lemmas 4.9 and 4.10, and bounding $E[T]$ as in the proof of Theorem 4.8.
□

4.3 The class of FOOLISH policies

Now that we have formalized the common heuristic of "prioritizing small jobs," and proved that all such policies are near-optimal for mean response time, it is natural to ask how policies that bias towards large

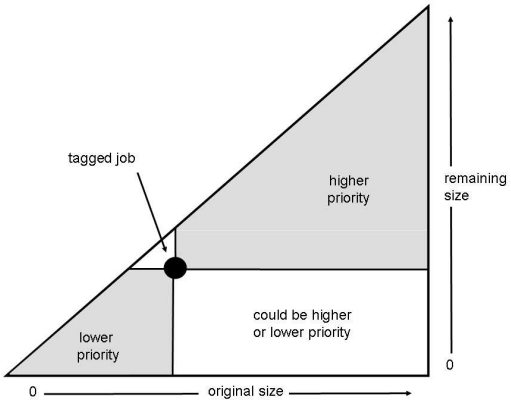


Figure 4.7: *This diagram illustrates the priority structure guaranteed by the Bias Property in the definition of the* FOOLISH *class. Note that the Bias Property under* FOOLISH *is more strict than the Bias Property under* SMART *, and was chosen in this way so that* PLJF *could serve as a lower bound for* FOOLISH *policies.*

job sizes compare. In this section, we will introduce the class of FOOLISH policies, which are policies that bias towards jobs with either large sizes or large remaining sizes. Though FOOLISH policies may not be practical in settings where providing small response times is the goal, policies that prioritize large jobs are known to perform well when minimizing the makespan (time until the last job completes) is the goal. See, for example, [176] for more details. In addition, it is interesting to study them in order to contrast the behavior of policies that bias towards large jobs with the behavior of policies using other heuristics, such as those that bias towards small jobs.

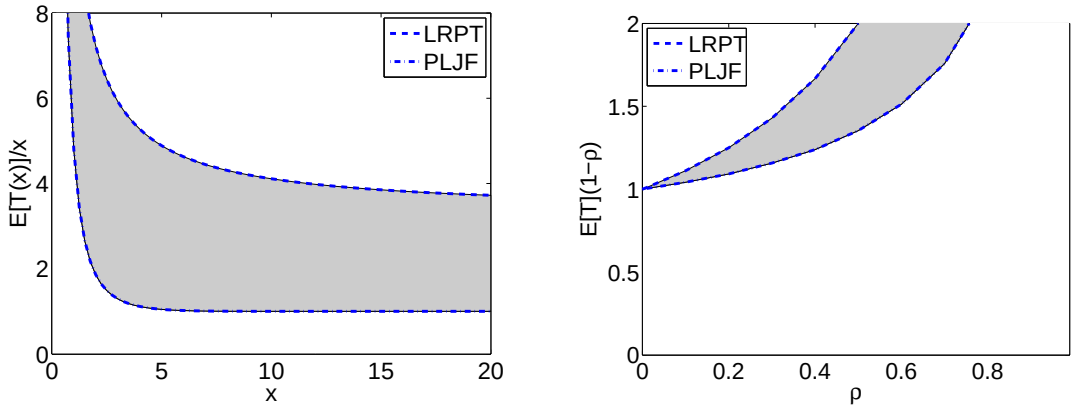
4.3.1 Defining FOOLISH scheduling

We now formally define the FOOLISH class. Recall that jobs will typically be denoted by a, b, or c. Job a will have remaining size r_a, original size s_a, and arrival time t_a. The original sizes, remaining sizes, and arrival times of b and c are defined similarly.

Definition 4.4 *A work conserving policy* $P \in$ FOOLISH *if it obeys the following property:*

Bias Property: *If* $r_b > r_a$ *and* $s_b > s_a$*, then job* b *has priority over job* a*.*

Notice that Definition 4.4 parallels Definition 4.1 of SMART policies. In particular, the Bias Properties in these definitions are similar in form. The relationship between the Bias Properties in the SMART and FOOLISH definitions can be seen by comparing Figures 4.2 and 4.7. However, an important difference between the two definitions is that the definition of FOOLISH policies does not include a Consistency or Transitivity Property like the SMART definition does. This is because these properties are used by the SMART definition to avoid time-sharing and other such behaviors that increase response times. However, all FOOLISH policies will have large response times, so there is no need to exclude these behaviors from the FOOLISH class.



(a) $E[T(x)]/x$ (b) $E[T](1-\rho)$

Figure 4.8: *This figure illustrates the bounds on $E[T(x)]$ and $E[T]$ under the class of* FOOLISH *policies in Theorems 4.14 and 4.15. The shaded area indicates the response times attainable using* FOOLISH *policies. In addition, the behaviors of the two most common* FOOLISH *policies,* PLJF *and* LRPT *, are illustrated. The service distribution is taken to be Exponential with mean 1, and the load in (a) is 0.7.*

4.3.2 Examples of FOOLISH policies

Many common policies are part of the FOOLISH class – Figure 4.1 provides an overview. Of course, the FOOLISH class includes LRPT and PLJF. Further, it is easy to prove that the FOOLISH class includes a range of policies having more complicated priority schemes. To illustrate this breadth, we introduce the FOOLISH$*$ classification, a subset of FOOLISH including a range of static priority policies.

Definition 4.5 *A policy $P \in$ FOOLISH$*$ if, at any given time, P schedules the job with the highest priority and gives each job of size s and remaining size r a priority using a fixed priority function $p(s,r)$ such that for $s_1 \leq s_2$ and $r_1 < r_2$, $p(s_1,r_1) < p(s_2,r_2)$.*

Theorem 4.13
FOOLISH$* \subsetneq$ FOOLISH

Note that FOOLISH$*$ includes all common FOOLISH policies (e.g. LRPT and PLJF), but that there are still many FOOLISH policies that are not in FOOLISH$*$. In particular, as with SMART, the definition of the FOOLISH class induces only a partial priority ordering on jobs in the queue. Thus, a FOOLISH policy may use time-varying prioritization where the priority function changes over time based on system-state information or randomization.

4.3.3 Bounding response times for FOOLISH policies

Now that we understand the definition and breadth of the FOOLISH class, we can move to bounding response times under FOOLISH policies. We will first derive bounds on $T(x)^P$ for $P \in$ FOOLISH and then we will use the bounds on $T(x)^P$ in order to derive bounds on $E[T]^P$.

Theorem 4.14
In a GI/GI/1 queue, for $P \in$ FOOLISH,

$$T(x)^{PLJF} \leq_{st} T(x)^{P} \leq_{st} T(x)^{LRPT}$$

Thus, for the M/GI/1,

$$\frac{x}{1-\rho+\rho(x)} + \frac{\lambda(E[X^2]-m_2(x))}{2(1-\rho+\rho(x))^2} \quad \leq \quad E[T(x)]^P \leq \frac{x}{1-\rho} + \frac{\lambda E[X^2]}{2(1-\rho)^2}$$

Proof. Consider the response time of tagged job j_x of size x under $P \in$ FOOLISH. First, notice that under LRPT every job finishes at the last moment of the residual busy period they arrive into and this is the last possible moment under any work conserving policy. Thus, $T(x)^P \leq_{st} T(x)^{LRPT}$.

To prove the lower bound, note that the moment before j_x completes, every job with size $> x$ (regardless of their remaining size) will have higher priority than j_x. Thus, the response time of j_x must include, at minimum, the time to complete every job of size $> x$ that is in the system at the arrival of j_x and that arrives while j_x is in the system. The arriving work of size $> x$ is exactly the arriving work that is completed under PLJF. Further, since PLJF always devotes the full server to jobs with size $> x$ when they exist, PLJF minimizes the work in the system made up of jobs with size $> x$. Thus, $T(x)^{PLJF} \leq_{st} T(x)^P$.

The bounds on $E[T(x)]^P$ can now be derived from the stochastic bounds.
□

The bounds in Theorem 4.14 are pictured in Figure 4.8 and show a stark contrast in behavior when compared with the bounds on $P \in$ SMART. Under FOOLISH policies $E[T(x)]/x$ has a decreasing trend as compared with an increasing trend under SMART policies. Further, $E[T(x)]/x$ is unbounded under FOOLISH policies and bounded under SMART policies.

We will now use the bounds on the conditional mean response time of FOOLISH policies in order to derive bounds on the overall mean response time of FOOLISH policies.

Theorem 4.15
In an M/GI/1 queue with $P \in$ FOOLISH,

$$E[T]^{PLJF} \leq E[T]^{P} \leq E[T]^{LRPT}$$

Further,

$$\frac{E[X]}{2}\left(\frac{1}{1-\rho} + \frac{1}{\rho}\log\left(\frac{1}{1-\rho}\right)\right) \leq E[T]^P \leq \frac{E[X]}{1-\rho}\left(1+\lambda E[\mathcal{E}]\frac{1}{1-\rho}\right)$$

Note that the bounds on $E[T]^P$ for $P \in$ FOOLISH in terms of $E[T]^{PLJF}$ and $E[T]^{LRPT}$ follow immediately from Theorem 4.14; however, though they are simple and elegant, they provide little information about the behavior of $E[T]$ as a function of the load or the job size variability under FOOLISH policies due to the complexity of the formula for mean response time under PLJF (LRPT has a simple mean response time). Thus, we provide a simple lower bound on $E[T]^{PLJF}$ that provides intuitive understanding of it's behavior.

Proof. The upper bound follows immediately from the observation that $E[T]^P \leq E[T]^{LRPT}$ for all work conserving policies. To prove the lower bound, let $g(x) = \rho - \rho(x)$ and thus $g'(x) = -\rho'(x)$ in the following:

$$\begin{aligned}
E[W]^P &\geq \int_0^\infty \frac{\lambda \int_x^\infty t^2 f(t)dt f(x)}{2(1-\rho+\rho(x))^2} dx \\
&\geq \frac{1}{\lambda} \int_0^\infty \frac{\rho'(x)(\rho-\rho(x))}{2(1-\rho+\rho(x))^2} dx \\
&= \frac{E[X]}{\rho} \int_0^\infty \frac{-g'(x)g(x)}{(1-g(x))^2} dx \\
&= -\frac{E[X]}{2\rho} \left(\frac{1}{1-g(x)} + \log(1-g(x)) \right|_0^\infty \\
&= \frac{E[X]}{2\rho} \left(\frac{1}{1-\rho} - 1 + \log(1-\rho) \right) \\
&= \frac{E[X]}{2(1-\rho)} + \frac{1}{2\lambda} \log(1-\rho)
\end{aligned}$$

Noting that

$$\begin{aligned}
E[R]^P &\geq \frac{1}{\lambda} \int_0^\infty \frac{\rho'(x)}{1-\rho+\rho'(x)} \\
&= -\frac{1}{\lambda} \log(1-\rho)
\end{aligned}$$

completes the proof.
□

The bounds on $E[T]$ under FOOLISH policies are illustrated in Figure 4.8. It is again interesting to compare the bounds on $E[T]$ under FOOLISH and SMART policies. The mean response time under SMART policies is significantly smaller than the mean response time under FOOLISH policies, and in fact, under many service distributions (for instance the Exponential used in Figure 4.8) all SMART policies have smaller mean response time than all FOOLISH policies under all loads.

4.4 The class of SYMMETRIC policies

We now move to the study of SYMMETRIC scheduling policies. The class of SYMMETRIC policies was introduced by Kelly nearly 30 years ago [113] and has proved fundamental to the study of queueing networks. SYMMETRIC disciplines have the important property that the departure process is stochastically identical to the arrival process when time is reversed. This *quasi-reversibility* property allows the decomposition of queueing networks where each server uses a symmetric discipline, and has led to the importance of the SYMMETRIC class to queueing networks.

However, in this thesis, we consider the class of SYMMETRIC disciplines not because of their behavior

in queueing networks, but because they provide an interesting generalization of classical PS scheduling, which is one of the most common models of scheduling in computer systems. In addition, SYMMETRIC policies provide "fairness" in the sense that they do not schedule based on any job traits – all arrivals are treated equivalently. This fact will have strong implications when we discuss fairness in Chapter 7.

4.4.1 Defining SYMMETRIC scheduling

We will now formally define the SYMMETRIC class. Consider a queue containing customers in positions $1, 2, \ldots, n$ where upon the completion of the ith job the jobs in positions $i+1, \ldots, n$ move to positions $i, \ldots, n-1$ and upon an arrival to the ith position the jobs in position $i, \ldots, n$ move to positions $i+1, \ldots, n+1$.

Definition 4.6 *A scheduling policy is a* SYMMETRIC *discipline if when n jobs are in the queue, the service rate is $\gamma(n)$, a proportion $\delta(i,n)$ of the server is directed to the ith customer in the queue, and an arrival enters position i with probability $\delta(i,n)$. Of course, $\sum_{i=1}^{n} \delta(i,n) = 1$ for all n. Further, a scheduling policy is an **M/GI/1** SYMMETRIC **discipline** when $\gamma(n) = 1$ for all n. Unless otherwise stated, we will only be considering M/GI/1* SYMMETRIC *disciplines in this thesis.*

Intuitively, SYMMETRIC policies are policies where the arrival and service rates for each position in the queue are "symmetric." Thus, the symmetry between the arrival rate and the service rate leads to the name SYMMETRIC.

The class of SYMMETRIC policies presents a nice counterpoint to the SMART and FOOLISH classes because SYMMETRIC policies treat all jobs equivalently as opposed to making scheduling decisions based on the size of arriving jobs. Thus, it is interesting to contrast the behavior of response times under SYMMETRIC policies with the behavior of response times under SMART and FOOLISH policies.

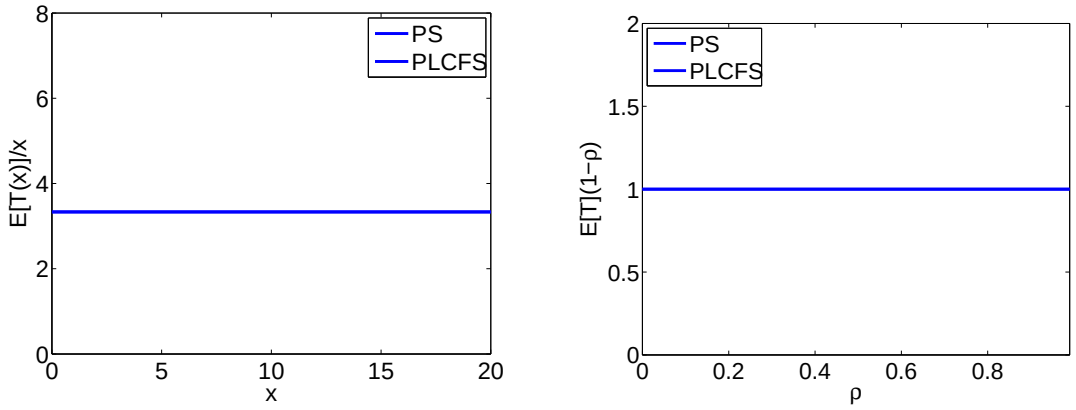
4.4.2 Examples of SYMMETRIC scheduling

Figure 4.1 provides an overview of policies in the SYMMETRIC class. It may not be immediately apparent, but the class of SYMMETRIC policies is actually quite broad, even in the case when $\gamma(n) = 1$ (i.e. the M/GI/1 setting). For instance, by taking $\delta(i,n) = 1/n$, we obtain PS. Further, we can obtain PLCFS by taking $\delta(i,n) = 1$ for $i = n$ and $\delta(i,n) = 0$ otherwise. In fact, we can obtain a wide variety of other hybrids between PLCFS and PS. For example, the SYMMETRIC policy with $\delta(i,n) = 1/(k \wedge n)$ for $i < k$ and $\delta(i,n) = 0$ otherwise performs PS among the last k jobs to arrive.

Though, we will primarily be concerned with M/GI/1 SYMMETRIC policies, it is worthwhile to point out the increased generality of the SYMMETRIC class in the case when $\gamma(n)$ is allowed to vary. For instance, we can obtain an M/GI/∞ queue as follows:

$$\begin{aligned} \gamma(n) &= n \\ \delta(i,n) &= 1 \end{aligned}$$

Further, we can obtain a M/GI/k/k system (a system with k servers where jobs either receive service imme-



(a) $E[T(x)]/x$ (b) $E[T](1-\rho)$

Figure 4.9: *This figure illustrates the behavior of $E[T(x)]$ and $E[T]$ under the class of* SYMMETRIC *policies. Recall that all* SYMMETRIC *policies have equivalent $E[T(x)]$ and $E[T]$. In both plots the service distribution is taken to be Exponential with mean 1, and the load in (a) is 0.7.*

diately upon arrival or are rejected) as follows:

$$\gamma(n) = \begin{cases} n & n = 1, \ldots, k \\ \xi & n = k+1, \ldots \end{cases}$$

$$\delta(i,n) = \begin{cases} 1/n & i = 1, \ldots, n; \;\; n = 1, \ldots, k \\ 1 & i = n; \;\; n = k+1, \ldots \\ 0 & i = 1, \ldots, n \;\; n = k+1, \ldots \end{cases}$$

where ξ is very large.[3]

Before moving on, it is important to also discuss the policies that are excluded from the SYMMETRIC class. Clearly any policy that prioritizes based on job size information is not in the SYMMETRIC class. Thus, the SYMMETRIC class is distinct from the SMART and FOOLISH classes. In addition, the SYMMETRIC class excludes policies that prioritize based on the age of jobs such as FCFS and FB. It may at first seem surprising that FCFS is not in the SYMMETRIC class, but notice that under FCFS the arrival rate is only positive for the last spot in the queue while the service rate is only positive for the first spot in the queue.

4.4.3 Bounding response times for SYMMETRIC policies

Now that we have explored the policies in the SYMMETRIC class, we are ready to bound the response times of SYMMETRIC policies. We will start by presenting results characterizing the stationary queue state under SYMMETRIC policies, and then we will use these results in order to analyze the response times under SYMMETRIC policies. All of the results in this section are adapted from Section 3.3 of [113] and

[3] For technical reasons, ξ cannot be infinite, see [113] for a discussion.

Section 5.7 of [192].

We need to begin by defining some notation. Let $\vec{x} = (x_1, x_2, \ldots, x_n)$ be the queue state where x_i is the attained service (age) of the ith job in the queue. Using this notation it is possible to characterize the stationary system state of SYMMETRIC policies as follows.

Theorem 4.16

In an M/GI/1 queue with $P \in$ SYMMETRIC the probability that the queue contains n jobs is

$$P(N = n) \quad = \quad \rho^i(1 - \rho)$$

Further, given there are n jobs in the queue, the age (excess) of each job is i.i.d. and follows the equilibrium distribution. Thus,

$$P(\vec{x} = (x_1, x_2, \ldots, x_n)|N = n) = \prod_{i=1}^{n} \frac{\overline{F}(x_i)}{E[X]}$$

Amazingly, this theorem states that all SYMMETRIC policies have stochastically equivalent queue states. The proof of this theorem is presented for the special case of PS in Section 5.7 of [192], and in complete generality in Section 3.3 of [113]. This theorem is a direct consequence of the fact that SYMMETRIC policies are pseudo-reversible. It is important to point out that since Theorem 4.16 is in the M/GI/1 setting, it also characterizes the system state at arrival and departure instants.

Using Theorem 4.16, it is easy to derive the mean conditional response time of SYMMETRIC policies, from which the overall mean response time follows immediately.

Theorem 4.17

In an M/GI/1 queue with $P \in$ SYMMETRIC,

$$E[T(x)]^P \quad = \quad \frac{x}{1 - \rho}$$

Thus,

$$E[T]^P = \frac{E[X]}{1 - \rho}.$$

It is quite interesting that the mean response time of all SYMMETRIC policies is the same. Not only that, the mean response time (and queue length distribution) of SYMMETRIC policies is insensitive to the service distribution beyond the mean. Further, the fact that $E[T(x)]$ is purely linear in x is an important property with many connotations for fairness that we will discuss in detail in Chapter 7.

Interestingly, Figure 4.9 illustrates that the behavior of $E[T(x)]$ under SYMMETRIC policies is quite different from the behavior of $E[T(x)]$ under the SMART and FOOLISH classes. While $E[T(x)]/x$ is a constant under SYMMETRIC policies, which illustrates an even treatment of all job sizes, $E[T(x)]/x$ has an increasing (decreasing) trend across x under SMART(FOOLISH) policies, which indicates the bias toward small (large) job sizes under these policies.

Beyond $E[T(x)]$ and $E[T]$, little is known about the distribution of response time under the class of SYMMETRIC disciplines as a whole. However, it is clear that SYMMETRIC policies are not equivalent with respect to higher moments of response time. A few results have been obtained for individual policies such as PS and PLCFS, but little has been proven about response times of other SYMMETRIC policies.

Two rare exceptions are the work of Avi-Itzhak and Halfin [9], where $Var[T(x)]$ under PLCFS, PS, and one other less common SYMMETRIC policy are compared, and the work of Kella, Zwart, and Boxma [112], where the tail behavior of some time-dependent properties of SYMMETRIC policies are derived. Since, the study of the behavior of higher moments of response time under the class of SYMMETRIC policies is difficult, in this thesis we will typically limit our focus to the most common two SYMMETRIC policies PS and PLCFS.

4.5 The class of PROTECTIVE scheduling

In designing scheduling policies there is always a tradeoff between providing small mean response times and providing "fairness." We have seen in Section 4.1 that it is possible to provide near optimal mean response times by prioritizing small job sizes; however it is commonly suggested that such policies may *starve* large jobs. In contrast, PS is typically thought of as a "fair" policy since it shares the server evenly among all jobs in the system, but PS has a mean response time that is far from optimal.

Despite the growing amount of research, the search for a fair policy with near optimal performance proved elusive until the 2003 ACM Sigmetrics conference when Friedman and Henderson presented a new policy called Fair-Sojourn-Protocol (FSP), which provides the first example of a fair policy that significantly improves upon the performance of Processor-Sharing (PS) [78]. The idea behind FSP is that it computes the times at which jobs would complete if the system were running PS and then prioritizes the jobs in terms of their PS completion times. That is, FSP devotes the full processor to the (uncompleted) job with the earliest PS completion time. Thus, FSP can be thought of as performing SRPT on the remaining times of a virtual PS system. Using simulations, Friedman and Henderson show that FSP has a very small mean response time. In fact, in may cases the mean response time of FSP is quite close to that of SRPT [78].

Following the introduction of FSP, the ideas behind FSP have led to the development of a number of other policies that also guarantee that *all* jobs have smaller response times than they would have had under PS on every sample path. The PROTECTIVE class, introduced by Friedman and Hurley, includes all such policies [79].

In this section we will first present the details of FSP and then we will use the ideas from the discussion of FSP in order to motivate and introduce the class of PROTECTIVE policies.

4.5.1 Fair-Sojourn-Protocol (FSP)

FSP is motivated by one simple idea: at any given point, it is easy to tell what the next job to finish under PS is. Given this information, it is possible to avoid wasting time time-sharing among jobs, and thus improve the response times of PS dramatically.

The easiest way to understand the FSP policy is to imagine that at any point in time you know the full state of a virtual PS queue, with the same arrival process. (Note, this won't actually be needed for the implementation below.) Under the FSP policy, the job being run is always the job that the virtual PS queue would have completed first. Observe that the FSP policy, like PS, is work-conserving; it just avoids time-sharing by choosing to focus all attention on one job at a time.

To understand the power and efficiency of FSP, consider the simple scenario of 3 jobs of size $1-\varepsilon$, 1, $1+\varepsilon$ that all arrive at time 0 at the server. Under PS, all jobs would time-share the server, slowing each

other down, and would all finish at about time 3. Under FSP, an ordering would be assigned to the jobs, and consequently the first arrival would finish at time $1-\epsilon$, the second at time $2-\epsilon$, and the third at time 3.

Remark 4.3 *Observe that the* FSP *algorithm is related to Weighted-Fair-Queueing (*WFQ*). Both policies involve simulating* PS*, and* FSP *can be viewed as a* preemptive *version of* WFQ *where each packet forms its own stream. Note that this is askew from the way* WFQ *is used in practice, which is to evenly distribute bandwidth among connections in a network while adhering to packetized constraints. In such a setting, non-preemptive implementations are used. Further, due to the more general parameters of* WFQ*, analyses do not provide bounds on performance any better than to state that* WFQ *is not worse than* PS *by the length of the largest packet [66, 171]. The specific settings of* WFQ *that give rise to* FSP *have not been analyzed.*

4.5.1.1 FSP in practice

The implementation of FSP is not very complex, in fact it is quite similar to the implementation of the SRPT algorithm. As in SRPT, preemptions may only occur at moments when a new job arrives under FSP or when a job departs under FSP. Thus the total number of preemptions is at most twice the number of arriving jobs, which in practice is far less than the number of preemptions under implementations of PS (which involve time quantums). Aside from preemptions, as in SRPT, there are also priority updates needed under FSP. These priority updates occur only at "event times," where an event is an arrival or departure under FSP or under the virtual PS system. Again, the number of updates is clearly not great. It is at most three times the number of outside arrivals.

We now explain the priority update needed at event points. Let E_i denote the most recent prior event and let E_{i+1} denote the current event. Observe that during the time between any two events there is only one job, call it j, in service under FSP. During the time between E_i and E_{i+1}, call this time t, the remaining size of job j under FSP decreases by t. Further, at the moment of E_{i+1}, we need to decrease the remaining time of every job under the virtual PS system by t/n, where n is the number of jobs in PS. Observe that our definition of events ensures that the number of jobs under PS does not change between two consecutive events.

We have seen that the implementation of FSP is quite similar to that of SRPT. The SRPT scheduling policy has been implemented in many real-world applications, such as scheduling in web servers [52, 96, 203]. The implementation of SRPT in [96] involves updating the priority of sockets in the Linux kernel, based on the remaining processing time required, and then draining these sockets into the network in order of their priority. An implementation of FSP would be equally simple, the only change would be that the priority updates would occur at the event points described above.

4.5.1.2 Bounding response times for FSP

The power of FSP comes from the following property of the policy: FSP finishes every job at least as early PS would. Intuitively, this is simply because, by its definition, FSP is only reordering the work that is being done so as to be more efficient. This result was proven by Friedman and Henderson in [78], but we include it here because the ideas are provide an important viewpoint on the behavior of FSP.

Let $\vec{r}^{PS}$ and $\vec{r}^{FSP}$ be vectors indicating the remaining work of each job under PS and FSP respectively. The vectors are ordered in the same way such that $r_1^{PS} \leq r_2^{PS} \leq \ldots \leq r_n^{PS}$. Thus, r_i^{PS} and r_i^{FSP} refer to the remaining work of the same job in PS and FSP respectively. Notice that (i) $\vec{r}^{FSP}$ is not necessarily ordered according to increasing remaining sizes and (ii) it is possible for $r_i^{FSP} = 0$ while $r_i^{PS} > 0$ for

several values of i.

Proposition 4.18
At all arrival and completion instants, for all $m \leq n$,

$$\sum_{i=1}^{m} r_i^{PS} \geq \sum_{i=1}^{m} r_i^{FSP}$$

where n is the number of jobs in the system.

Notice that this result immediately implies that every job finishes no later under FSP than it would have under PS.

Proof. We prove the result using induction on the sequence of events in a busy period. An event is either a virtual service completion (a completion in PS), an arrival, or a service completion under FSP. Let $\vec{r}^{PS} \gg \vec{r}^{FSP}$ indicate that for all $m \leq n$, $\sum_{i=1}^{m} r_i^{PS} \geq \sum_{i=1}^{m} r_i^{FSP}$. We can see that the claim holds for the first arrival of a busy period; thus the base case holds trivially.

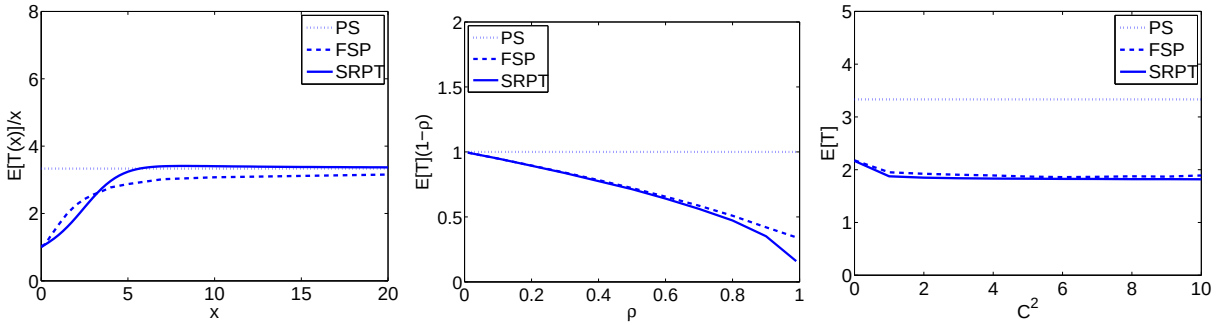
Now, suppose $\vec{r}^{PS} \gg \vec{r}^{FSP}$ at the time of event E_e, $e \geq 1$. Let $\vec{r}^{PS}$ and $\vec{r}^{FSP}$ be the vectors at the time of event E_e and let t be the time between the occurrence of event E_e and event E_{e+1}. Let $\vec{r}^{PS'}$ and $\vec{r}^{FSP'}$ be the updated vectors just before event E_{e+1}. Let i be the index of the first nonzero value in $\vec{r}^{FSP}$. Then

$$\begin{aligned} r_j^{FSP'} &= r_j^{FSP} = 0,\ j = 1, \ldots, i-1 \\ r_i^{FSP'} &= r_i^{FSP} - t \\ r_j^{FSP'} &= r_j^{FSP},\ j = i+1, \ldots, n \\ r_j^{PS'} &= r_j^{PS} - t/n,\ \forall j \end{aligned}$$

Notice that, because t is defined as the time between event E_e and E_{e+1}, $t \leq r_i^{FSP}$ and $t/n \leq r_j^{PS}$ for all j. We can now see that $\vec{r}^{PS'} \gg \vec{r}^{FSP'}$ is maintained immediately before event E_{e+1}, which completes the proof.
□

Apart from Proposition 4.18, there exists very little work analyzing the performance of FSP. Thus, in order to understand how much FSP improves response times over PS, researchers have been limited to simulation studies such as those in [78, 87, 88]. Following the lines of these studies, we will now briefly illustrate how the mean response time of FSP compares to that of PS and further, to the optimal mean response time, $E[T]^{SRPT}$.

Figure 4.10 illustrates a simple comparison of the response times under FSP, PS, and SRPT. In Figure 4.10 (a) the behavior of $E[T(x)]$ under each of these policies is shown. We can see that though FSP provides improvements over PS in response times for all job sizes, the improvements are most dramatic for small job sizes. But, the bias towards small job sizes is not as extreme under FSP as it is under SRPT. Moving to Figure 4.10 (b) and (c), we can see that the overall response times of FSP and SRPT are very similar as long as the load is not too high. Only when $\rho \to 1$ does SRPT provide dramatic gains in $E[T]$ over FSP. Further, we can see that FSP, like SRPT, has a mean response time that is nearly insensitive to service demand variability.



(a) $E[T(x)]/x$ (b) $E[T](1-\rho)$ (c) $E[T]$ vs. $C^2[X]$

Figure 4.10: *These plots show a comparison between* PS, FSP, *and* SRPT *with respect to both* $E[T(x)]$ *and* $E[T]$. *The service distribution in (a) and (b) is Exponential with mean 1. In (c) the service distribution is a 2-phase Coxian with mean 1 and varying* $C^2[X]$. *The load in (a) and (c) is 0.7.*

4.5.2 Defining PROTECTIVE scheduling

We will now generalize the ideas from FSP in order to present the class of PROTECTIVE policies, which were originally introduced by Friedman and Hurley in [79].

Definition 4.7 *A scheduling policy* P *is* PROTECTIVE *if for any input sequence, the response time of any job under* P *is not greater than it would have been under* PS.

Clearly, FSP $\in$ PROTECTIVE while SRPT and PSJF are not PROTECTIVE policies. In fact, the class of PROTECTIVE policies is actually quite narrow, and includes only policies that behave very similarly to FSP.

In order to characterize the class of PROTECTIVE policies we will use the concept of *slack*. The slack of a job represents how tightly the "protectiveness" constraint binds each job. Formally, the slack of the mth job under policy P, denoted s_m, is defined as

$$s_m = \left(\sum_{i=1}^{m} r_i^{PS}\right) + (n-m+1)r_m^{PS} - \left(\sum_{i=1}^{m} r_i^{P}\right)$$

Further, $s(t) = (s_1, \ldots, s_n)$ is the vector of the slacks of each job at time t.

Though the formula for slack may seem complicated, this is a very intuitive concept. Consider the first job in the virtual PS queue. If no further jobs arrive, the first job will complete at time nr_1^{PS} in the virtual PS queue. If the our PROTECTIVE policy P works on this job it will finish at time r_1^P. Thus, the slack of this job is $nr_1^{PS} - r_1^P$. Similarly, the second job will complete at time $r_1^{PS} + (m-1)r_2^{PS}$ in the virtual PS queue and at time $r_1^P + r_2^P$ under policy P. Thus, the slack is $s_2 = r_1^{PS} + (m-1)r_2^{PS} - r_1^P + r_2^P$.

Proposition 4.19
A scheduling policy P *is* PROTECTIVE *if and only if* $s(t)$ *is always non-negative.*

Using this proposition, we can now easily develop a large number of PROTECTIVE policies. Clearly, FSP always maintains a positive $s(t)$, however in addition two other policies are of interest: Optimistic FSP (OFSP) and Pessimistic FSP (PFSP). In describing these policies the following definitions are useful:

Definition 4.8 *A job is **servable** at time t if a* PROTECTIVE *policy can serve it at time t, independent of future arrivals. Further, a job is **completable** at time t if a* PROTECTIVE *policy can serve it to completion starting at time t, independent of future arrivals.*

It follows immediately from the above definition that job m is servable if and only if for all $i < m$, $s_i > 0$, and job m is completable if and only if for all $i < m$, $s_i > r_m^P$.

These two concepts provide a nice description the three most common PROTECTIVE policies:

FSP FSP serves the lowest indexed uncompleted, servable job.

OFSP Optimistic FSP serves the servable job with the smallest remaining processing time.

PFSP Pessimistic FSP serves the completable job with the smallest remaining processing time.

By definition, all three of these policies have smaller mean response time than PS under all service distributions and arrival processes. Further, Friedman & Hurley show that PFSP has a smaller mean response time than FSP under all service distributions and all loads [79]. Interestingly though, OFSP and PFSP have the property that no PROTECTIVE policy can have smaller mean response times across all loads and all service distributions. Additional results on the worst case behavior of these three policies can be found in [79]; however, no queueing results exist for any PROTECTIVE policies.

4.5.3 Bounding response times for PROTECTIVE policies

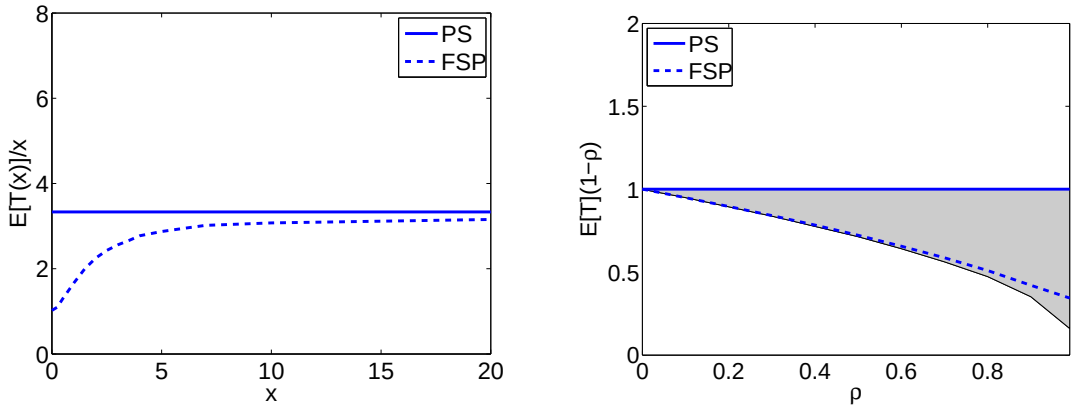
We will now present simple bounds on the response times of protective policies. As we have seen, the description of PROTECTIVE policies relies heavily on decisions based on the current state of PS. Thus, a tight analysis of the class of PROTECTIVE policies is extremely difficult. We can obtain some simple bounds on the class that provide some indication on the behavior of response times.

We begin by noting that the definition of the PROTECTIVE class immediately gives the following bounds on overall mean response time. Note that the upper bound is tight since PS is a PROTECTIVE policy, but the lower bound is simply the lower bound on SRPT proven in Theorem 3.10 and is thus loose as $\rho \to 1$.

Theorem 4.20
In an M/GI/1 system, for $P \in$ PROTECTIVE, For all $P \in$ PROTECTIVE,

$$\frac{E[X]}{\rho} \log\left(\frac{1}{1-\rho}\right) \leq E[T]^{SRPT} \leq E[T]^{P} \leq \frac{E[X]}{1-\rho}$$

It turns out that it is hard to obtain a better lower bound on the overall mean response time of PROTECTIVE policies than the above because the tagged job approach is not feasible. That is, you cannot obtain a tight lower bound on the overall mean response time by analyzing the mean response time of a tagged job because a PROTECTIVE policy can be designed so as to perform very well for any particular job size at the expense of other job sizes. In particular, imagine a policy where a job of size x receives service whenever it is servable.



(a) $E[T(x)]/x$ (b) $E[T](1-\rho)$

Figure 4.11: *This figure illustrates the behavior of $E[T(x)]$ and $E[T]$ under the class of* PROTECTIVE *policies. The shaded area indicates the response times attainable using* PROTECTIVE *policies. Recall that* PS *provides an upper bound on both $E[T(x)]$ and $E[T]$ under* PROTECTIVE *policies. We do not have a lower bound on $E[T(x)]$ under* PROTECTIVE *policies, so none is shown; however, we have proven that $E[T(x)]/x \to 1/(1-\rho)$ as $x \to \infty$ and $E[T(x)]/x \to 1$ as $x \to \infty$. Thus any lower bound on the* PROTECTIVE *class will have a similar behavior to that of* FSP *, which is illustrated in the figure. In both plots the service distribution is taken to be Exponential with mean 1, and the load in (a) is 0.7.*

However, we can get a feel for the behavior of PROTECTIVE policies by studying the behavior of the conditional response time under PROTECTIVE policies, $T(x)$. Again, deriving tight bounds on $T(x)$ for all x is difficult, so we focus on the limiting cases of $x \to 0$ and $x \to \infty$. Interestingly, PROTECTIVE policies can behave very differently with respect to small job sizes, $\lim_{x\to 0} T(x)^{PS}/x = 1/(1-\rho)$ a.s. while $\lim_{x\to 0} T(x)^{FSP}/x = 1$ a.s. However, all PROTECTIVE policies treat large jobs equivalently:

Theorem 4.21
Consider a GI/GI/1 system with $P \in$ PROTECTIVE. For all x we have

$$1 \leq \frac{E[T(x)]}{x} \leq \frac{1}{1-\rho}$$

Further, as $x \to \infty$,

$$\lim_{x\to\infty} \frac{T(x)}{x} = \frac{1}{1-\rho}$$

The proof of the bounds on $E[T(x)]/x$ is immediate from the definition of PROTECTIVE policies and the . The proof of the limit as $x \to \infty$ is also straightforward, but requires techniques that we introduce in Section 7.2.1 where we study the limiting behavior of $T(x)/x$ as $x \to \infty$ under a range of common policies, so we will omit it.

It is interesting to contrast the results in Theorems 4.20 and 4.21 with the corresponding results for the

SMART, FOOLISH, and SYMMETRIC classes. Figure 4.11 provides an illustration of the behavior of PROTECTIVE policies. Upon comparison with Figures 4.4, 4.8, and 4.9, it seems that PROTECTIVE policies are obtaining the best of both worlds. They guarantee that $E[T(x)] \leq E[T(x)]^P$ for all x and all $P \in$ SYMMETRIC, but still provide mean response times that are far lower than those of FOOLISH policies, and are even comparable with those of SMART policies. However, analysis has not yet yielded tight lower bounds on the performance of PROTECTIVE policies, thus it is difficult to make any definitive conclusions about the comparison between $E[T]$ under SMART policies and PROTECTIVE policies. Simulations seem to indicate that in many cases, especially under high load, SMART policies can significantly outperform PROTECTIVE policies with respect to $E[T]$. However, this is exactly the setting where starvation becomes an issue in SMART policies, which is an indication of the inherent tradeoff scheduling policies must make between efficiency and fairness.

4.6 Concluding remarks

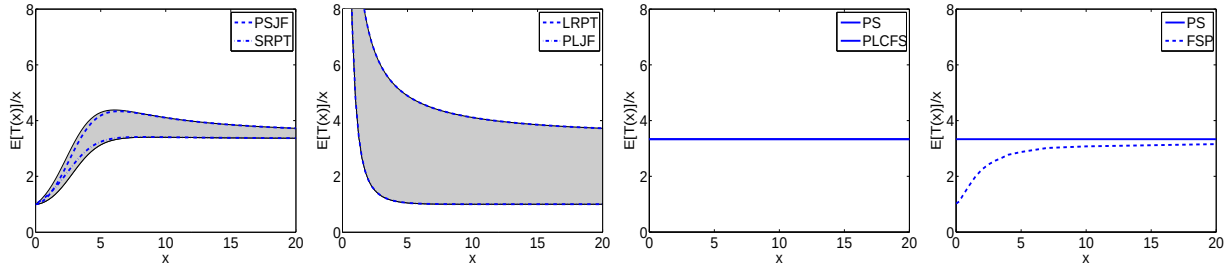
The work in this chapter develops a new style of research that attempts to bridge the gap between theoreticians and practitioners by studying classifications of scheduling policies instead of individual idealized policies. The goal is to formalize a scheduling heuristic such as "prioritizing small jobs" and then study the impact of this *heuristic* instead of studying one specific policy that obeys the heuristic. The hope is that the analysis of these heuristic classifications provides both practical and theoretical benefits. Theoretically, such results add structure to the space of scheduling policies that cannot be obtained by analyzing individual policies, and practically, such results provide analyses of the policies that are implemented in practice.

In particular, we presented four heuristic classes: the class of SMART policies (and its generalization – the SMART$_\epsilon$ class), the class of FOOLISH policies, the class of SYMMETRIC policies, and the class of PROTECTIVE policies. In addition to defining the four classifications, we began to analyze the performance of each heuristic by proving bounds on the overall and conditional mean response times of policies in each class. These bounds illustrate the enormous impact that the underlying scheduling heuristic used in a policy has on determining the performance of the policy. Figures 4.12 and 4.13 illustrate the contrasting response time behavior under the four heuristic classes studied in this chapter. There are many interesting comparisons that can be made using these figures.

First of all, notice the contrast between the behavior of SMART and FOOLISH policies: while $E[T(x)]/x$ has an increasing trend under SMART policies, it has a decreasing trend under FOOLISH policies. This is indicative of the bias towards small jobs under SMART policies and the bias towards large jobs under FOOLISH policies. Interestingly though, there are some FOOLISH policies that have larger $E[T(x)]/x$ for large job sizes than any SMART policy does. With respect to $E[T]$, the comparison between SMART and FOOLISH policies is as expected: $E[T]$ for all SMART policies is lower than $E[T]$ for any FOOLISH policy.

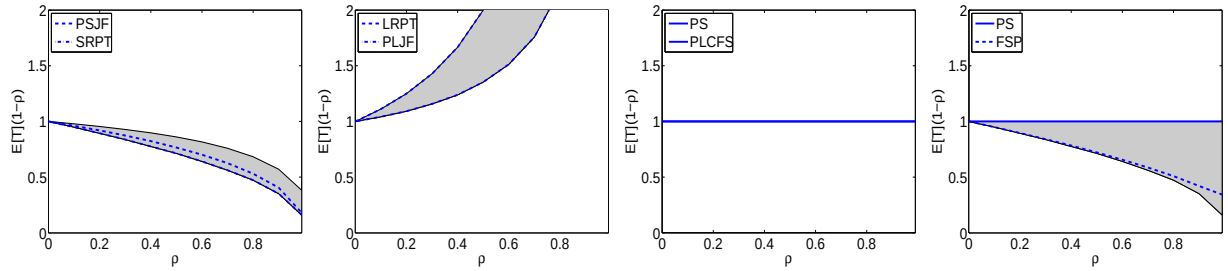
It is also interesting to observe the similar behaviors of SYMMETRIC and PROTECTIVE policies. Policies in both cases provide fair response times, in the sense that no job size receives "unfairly" large $E[T(x)]/x$, but PROTECTIVE policies can provide strictly better $E[T(x)]/x$ and $E[T]$ that SYMMETRIC policies.

Finally, it is important to notice the contrast between the size based classes (SMART and FOOLISH) and the fairness based classes (SYMMETRIC and PROTECTIVE). Under the fairness based classes



(a) SMART policies (b) FOOLISH policies (c) SYMMETRIC policies (c) PROTECTIVE policies

Figure 4.12: *These figures illustrate the bounds on $E[T(x)]/x$ under scheduling heuristics. The service distribution is taken to be Exponential with mean 1, and the load is 0.7. Note that no bounds are shown for* PROTECTIVE *policies because, though $E[T(x)]/x \leq 1/(1-\rho)$ under* PROTECTIVE *policies, we have no good lower bound for $E[T(x)]^P/x$ for $P \in$* PROTECTIVE*. However, we have proven that $E[T(x)]/x \to 1/(1-\rho)$ as $x \to \infty$ and $E[T(x)]/x \to 1$ as $x \to \infty$.*



(a) SMART policies (b) FOOLISH policies (c) SYMMETRIC policies (d) PROTECTIVE policies

Figure 4.13: *These figures illustrate the bounds on $E[T]$ under scheduling heuristics. The service distribution is taken to be Exponential with mean 1.*

$E[T(x)]/x$ is fairly constant or moderately increasing, while under the size based classes, the bias towards particular job sizes shows up strongly in the behavior of $E[T(x)]/x$. This observation will have ramifications later in the thesis when we discuss the fairness of these heuristic classifications.

Let us end this chapter with one final note. Though we have discussed a number of scheduling heuristics in this chapter, one heuristic in particular is by far the most interesting from a practical perspective: that of prioritizing small jobs. Thus, the definitions of SMART and SMART$_\epsilon$ represent an important contribution both theoretically and practically. In this chapter, we have proven that all SMART policies have mean response time within a factor of 2 of optimal. This result can be seen as a theoretical validation of recent designs suggesting the use of variants of SRPT in web servers and wireless networks. Further, we have seen that even when policies prioritize based on estimated job sizes (SMART$_\epsilon$), they are still within a constant of the optimal $E[T]$. This is especially important since, in many applications, exact job size information

is not known. For example, in web servers and wireless networks, designs that prioritize the job with the smallest *estimated* remaining size have been proposed [182, 131, 130, 102, 136]. But, in such designs a key question is "how good must job size estimates be to provide improvements in response time?" The reason this question is so important is that there are overheads involved in estimating the job sizes. For example, in a web server, estimating the network delay a request will experience requires using packet probing techniques. The SMART_ϵ classification allows us to provide simple bounds that illustrate the tradeoff between the accuracy of job size estimates and the performance of the resulting policy. Further, our results illustrate how the underlying true job size distribution affects this tradeoff.

Due to their importance, we will return to the SMART and SMART_ϵ classifications often throughout the remainder of the thesis. For example, we will show that all SMART policies are asymptotically equivalent with respect to the response time distribution (Chapter 6) and have similar fairness properties (Chapter 7).

CHAPTER 5

Classification via scheduling techniques

The *scheduling heuristic* applied in a scheduling policy is only one defining aspect of the policy, another is the *scheduling technique* used by the policy. For instance, SRPT is defined both by the fact that it prioritizes small jobs (the scheduling heuristic) and the fact that it uses remaining sizes to prioritize (the scheduling technique). While the scheduling heuristic used in a policy is typically determined by the system designer, the scheduling technique used is often determined by the application itself. For example, a designer may wish to apply SRPT, since it optimizes mean response time, but the application itself may prevent the use of SRPT, thus forcing the use of a policy that prioritizes small jobs using a different scheduling technique.

There are many factors in applications that limit the use of certain scheduling techniques. Consider the case of SRPT. If an application does not have knowledge of the remaining sizes of jobs (e.g. an operating system deciding which process to run or a router deciding which flow to schedule) scheduling with SRPT is impossible. SRPT is also not feasible for applications where preemption is not possible, e.g. in systems such as supercomputers and databases where preemption is too computationally expensive to use. Finally, SRPT is not an option when the system cannot distinguish between an infinite number of classes of jobs. Under SRPT the system must distinguish between every possible remaining size, but in cases such as web servers this infinite precision requires too much overhead.

The limitations applications place on the scheduling techniques that are available to system designers have a huge impact on the performance attainable for that application. Consider the availability of preemption. In an application such as databases, where preemption is too computationally expensive and the scheduler does not have job size information, one cannot hope to obtain the same performance as in web server scheduling where preemption is possible and job sizes can be estimated accurately.

Our focus in this chapter is on formalizing classes of policies based on scheduling techniques. We will then use these classes throughout the thesis to illustrate the impact of scheduling techniques on the efficiency and fairness of policies. In particular, we formalize four classifications of scheduling policies based on the technique used:

(i) Preemptive size based policies
(ii) Remaining size based policies

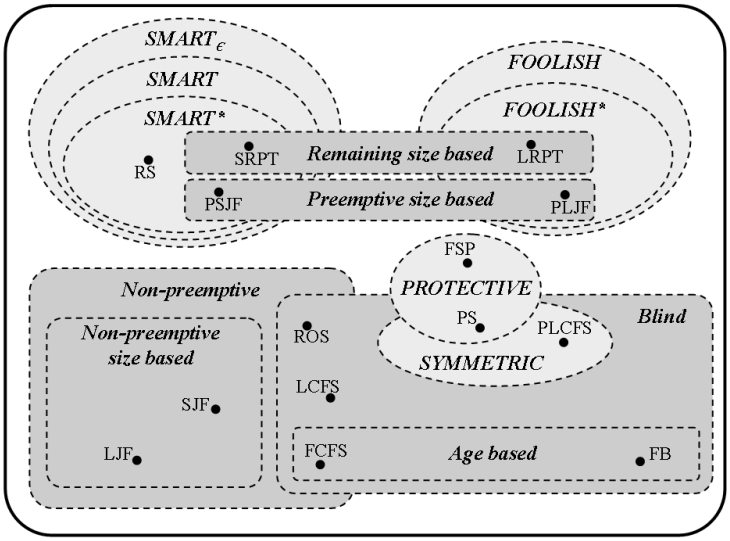


Figure 5.1: *An illustration of the common policies that fall into each of the classifications studied in this thesis. The scheduling heuristic classifications introduced in Chapter 4 are shown in ovals and the scheduling technique classifications introduced in this chapter are shown in rectangles.*

(iii) Age based policies
(iv) Non-preemptive policies

These classes are illustrated in Figure 5.1 and cover a wide variety of scheduling techniques that have been applied across a range of applications including web servers, routers, disks, operating systems, and others. The classes of remaining size based policies, preemptive size based policies, and age based policies were introduced by Wierman et al. in [238, 240], while the class of non-preemptive policies has been studied often in the literature, see for example [119, 120, 247].

We begin our discussion of scheduling techniques by discussing the class of preemptive size based policies in Section 5.1. Then, in Section 5.2, we move to the class of remaining size based policies. Next, in Section 5.3, we consider age based scheduling, which uses no job size information. And finally, we consider non-preemptive policies in Section 5.4. For each scheduling technique, we will introduce a formal definition of the class, discuss examples of policies in the class, and derive bounds on the attainable response times of policies in the class.

The bounds on the attainable response times help to characterize the overall efficiency of each of these techniques and also illustrate the effect of scheduling techniques on the response times of individual job sizes. Since these classes are defined only by the scheduling technique and allow arbitrary scheduling heuristics to be applied, the bounds are often quite broad and seem to provide little information about the behavior of scheduling policies in the class. However, these bounds will be of great use later when discussing the fairness of scheduling techniques.

5.1 The class of preemptive size based policies

We have seen many times in this thesis already that when applications know the sizes of the jobs in the queue, using this information to schedule can provide substantial gains in mean response time. In this section, we study the class of all policies that prioritize according to some bijection of (original) job sizes. Thus, the results apply to PSJF , which prioritizes jobs with small original size, and PLJF, which prioritizes jobs with large original size, in addition to hybrid policies having more complex priority assignments, e.g. policies that prioritize both small and large job sizes to ease fairness concerns.

5.1.1 Defining a class of preemptive size based policies

Formally, we define the class of preemptive size based policies as follows.

Definition 5.1 *Under a **preemptive size based policy**, the priority of a job is assigned based on a fixed priority function that is a bounded bijection from job sizes to priorities. Priorities are assigned upon arrival and cannot be adjusted. The job with the highest priority is run at all instants, and if two jobs of the same size (and thus priority) are in the system, then the job that arrived first is given higher priority.*

It is important to point out the breadth of this definition. Clearly PSJF and PLJF are both preemptive size based policies. Thus, the class of preemptive size based policies includes some SMART policies that prioritize small job sizes as well as some FOOLISH policies that prioritize large job sizes. In addition, the class of preemptive size based policies includes many hybrid policies that are neither SMART or FOOLISH. For example, the class of preemptive size based policies includes policies where both jobs with small sizes and some large job sizes receive elevated priority in order to curb unfairness.

Despite the breadth of this definition, it also has some limitations that would be interesting to address in future work. Two important extensions of this group would be (i) to include the possibility that ranges of job sizes all receive the same priority and (ii) to include policies where the priority function can depend on the service distribution and load. However, such extensions would make the class significantly more difficult to study analytically.

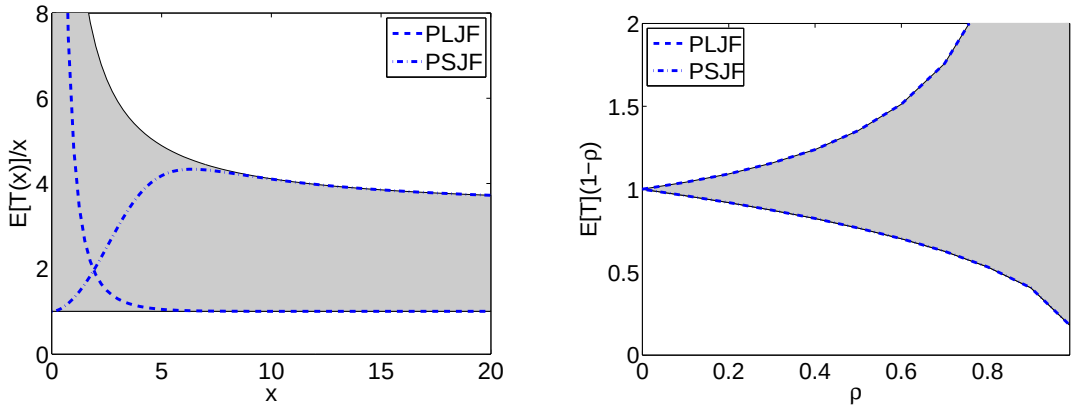
5.1.2 Bounding response times for preemptive size based policies

As with all of the classifications based on scheduling techniques, the class of preemptive size based policies includes policies that use a wide range of scheduling heuristics, and so, policies in the class have a a wide range of attainable response times. For example, the class includes PSJF, which has $E[T]$ within a constant factor of optimal, and PLJF, which has $E[T]$ nearly as large as possible. However, it is still possible to obtain bounds that are useful for analyzing the fairness of policies in the class.

We begin the analysis by bounding $E[T(x)]$.

Proposition 5.1
For any preemptive size based policy P,

$$x \leq E[T(x)]^P \leq \frac{x}{1-\rho} + \frac{\lambda E[X^2]}{2(1-\rho)^2}.$$



(a) $E[T(x)]/x$ (b) $E[T](1-\rho)$

Figure 5.2: *This figure illustrates the bounds on $E[T(x)]$ and $E[T]$ under the class of preemptive size based policies. The shaded area indicates the response times attainable using preemptive size based policies. In addition, the behaviors of the two most common preemptive size based policies,* PSJF *and* PLJF*, are illustrated. In both plots the service distribution is taken to be Exponential with mean 1, and the load in (a) is 0.7.*

Further,

$$E[T]^{PSJF} \leq E[T]^{P} \leq E[T]^{PLJF}.$$

Proof. The optimality of PSJF (among preemptive size based policies) for $E[T]$ follows immediately from Phipps [175] or similar arguments in Aczel [10], and the fact that PLJF maximizes $E[T]$ follows from parallel arguments. Thus, we restrict our focus to proving the bounds on $E[T(x)]$. The lower bound follows from the fact that $E[T(x)]^P$ must at least be long enough to serve the job itself. The upper bound follows from the fact that a job must complete by the end of the residual busy period it arrives into, $E[B(x+Q)]$, where Q is the steady-state work in the system.
□

The bounds in Proposition 5.1 are pictured in Figure 5.2.

Note that Proposition 5.1 is tight. Clearly the bounds on $E[T]$ are tight since PSJF and PLJF are both preemptive size based policies. To see that the bounds on $E[T(x)]$ are tight consider a policy P_x that gives highest priority to a job of size x. Under any service distribution where jobs of size x make up zero probability mass, $E[T(x)]^{P_x} = x$ since any arriving job of size x receives preemptive priority. Similarly if a policy P'_x gives jobs of size x lowest priority then under any service distribution where jobs of size x make up zero mass, an arriving job of size x will finish at the end of the busy period into which it arrives, which matches the upper bound. Thus, for all x, there is some preemptive size based policy that achieves the bounds in Proposition 5.1.

However, though the bounds are tight, they convey little information about the behavior of any individual size based policy: the lower bounds on $E[T(x)]$ and $E[T]$ are (near) optimal and the upper bounds are

(nearly) as large as possible under work conserving policies. We will remedy this in Chapter 7 where we will show that any individual size based policy must have some x_1 such that $E[T(x_1)]$ matches the lower bound in Proposition 5.1 and some x_2 such that $E[T(x_2)]$ matches the upper bound in Proposition 5.1. This improved result will be fundamental to understanding the fairness of preemptive size based policies.

5.2 The class of remaining size based policies

In applications where the remaining size of a job is known, it is an invaluable resource for use in scheduling. We have already seen that SRPT is optimal with respect to mean response time, but many hybrids of SRPT also maintain near optimal mean response times. In this section, we study the class of all policies that prioritize using only the remaining size of jobs. Thus, the results apply to SRPT as well as LRPT, which prioritizes jobs with large remaining sizes. In addition, the results apply to a wide range of other disciplines having more complex priority schemes in order to curb fairness concerns, e.g. those suggested by Gong and Williamson [87, 88].

5.2.1 Defining a class remaining size based policies

Formally, we define the class of remaining size based policies as follows.

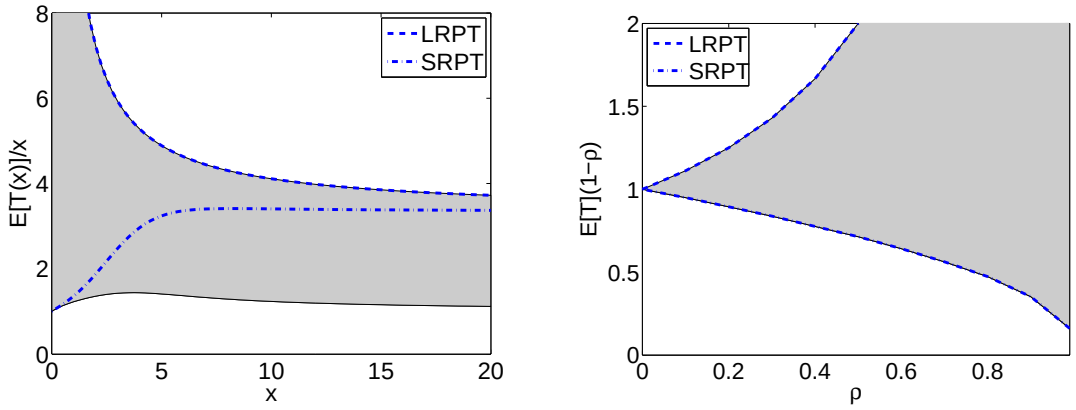
Definition 5.2 *Under a **remaining size based policy**, the priority of a job is assigned based on a fixed priority function that is a bounded bijection from remaining sizes to priorities. The priority of a job is updated as the remaining size of the job changes, and the job with the highest priority is preemptively given service. If two jobs have the same remaining size, the job that attained that remaining size first is given higher priority.*

It is important to point out the breadth of this definition. Clearly SRPT and LRPT are both remaining size based policies. Thus, the class of remaining size based policies includes some SMART policies that prioritize small job sizes as well as some FOOLISH policies that prioritize large job sizes. In addition, the class of remaining size based policies includes many hybrid policies that are neither SMART or FOOLISH. For example, the class of remaining size based policies includes policies where small remaining sizes receive high priority and some large remaining sizes also receive high priority in order to curb unfairness.

As with the class of preemptive size based policies, there are some limitations to the definition of preemptive size based policies that hopefully can be addressed in future research. The class of remaining size based policies does not include policies where jobs with different remaining sizes have equivalent priorities. Further, the results in this section do not include randomized policies.

5.2.2 Bounding response times for remaining size based policies

We now move to bounding response times under remaining size based policies. Clearly, since both SRPT and LRPT are remaining size based policies, there are a wide range of $E[T]$ that are possible within this class. For instance, SRPT optimizes $E[T]$ while LRPT has the largest possible $E[T]$ among work conserving policies. Thus, no non-trivial bounds are possible for $E[T]$ under remaining size based policies.



(a) $E[T(x)]/x$ (b) $E[T](1-\rho)$

Figure 5.3: *This figure illustrates the bounds on $E[T(x)]$ and $E[T]$ under the class of remaining size based policies. The shaded area indicates the response times attainable using remaining size based policies. In addition, the behaviors of the two most common remaining size based policies,* SRPT *and* LRPT*, are illustrated. In both plots the service distribution is taken to be Exponential with mean 1, and the load in (a) is 0.7.*

However, it is possible to obtain useful bounds on the behavior of $E[T(x)]$ under remaining size based policies. Recall that Q_x^{SRPT} is the stationary work in the SRPT queue made up by jobs with remaining size $< x$.

Proposition 5.2

For any remaining size based policy P,

$$x + \frac{\lambda \widetilde{m}_2(x)}{2(1-\rho(x))} \leq E[T(x)]^P \leq \frac{x}{1-\rho} + \frac{\lambda E[X^2]}{2(1-\rho)^2}$$

Further,

$$E[T]^{SRPT} \leq E[T]^P \leq E[T]^{LRPT}$$

Proof. We have already discussed the bounds on $E[T]$, so we need only prove the bounds on $E[T(x)]$. The upper bound follows from the fact that every job must finish before the end of the residual busy period into which it arrives, i.e. $E[B(x+Q)]$. Note that this bound is achieved for all x under LRPT.

The lower bound follows from the fact that all work in the system having remaining size less than x will complete before an arriving tagged job of size x. Further, SRPT always devotes the full processor to completing such work when it exists and this quantity of work makes up the same average load under all scheduling policies. Thus SRPT minimizes the time average quantity of such work and the result follows using PASTA and the fact that the work having remaining size less than x under SRPT is $\frac{\lambda \widetilde{m}_2(x)}{2(1-\rho(x))}$.

□

The bounds in Proposition 5.2 are illustrated in Figure 5.3. Clearly, the upper bound in Proposition 5.2 is tight since LRPT is a remaining size based policy. However, it seems that the lower bound is likely loose. In particular, it is unclear how to optimize $E[T(x)]$ for a fixed x because giving jobs with remaining size x highest priority (as we did for preemptive size based policies) is far from optimal! Further, both SRPT and LRPT have $E[T(x)]/x \to 1/(1-\rho)$ as $x \to \infty$ so it seems that this is likely true for all remaining size based policies, but the lower bound has $E[T(x)]/x \to 1$. Thus, an interesting question for future work is to derive a tight lower bound.

Though the bounds on $E[T]$ and $E[T(x)]$ seem to indicate that the behavior of policies in the class is too disparate to allow non-trivial analysis, we will be able to make strong conclusions about the fairness and predictability of response times under remaining size based policies in Chapter 7.

5.3 The class of age based policies

In many modern computer systems, scheduling decisions must be made without knowledge of the service requirements of jobs. For example, in routers, the length of the current flow being scheduled is completely unknown and in operating systems the service demand of any process being executed is unknown. In these settings, the age (i.e. attained service) of a job can serve as an indication of the remaining size of a job. For instance, when the service distribution has a decreasing (increasing) failure rate, jobs with large ages likely have larger (smaller) remaining sizes. Two of the most common age based policies are FCFS and FB. FCFS prioritizes according to an increasing function of the age of a job (the larger the age, the higher the priority); whereas FB prioritizes according to a decreasing function of the age of a job (the larger the age, the lower the priority).

In this section, we study all policies that prioritize according to some bijection of the ages of jobs. Thus, the results apply to FCFS and FB as well as hybrid policies having more complex priority curves. Such hybrid policies have been suggested by a number of researchers, e.g. [180], as a way to curb the unfairness to large job sizes under FB.

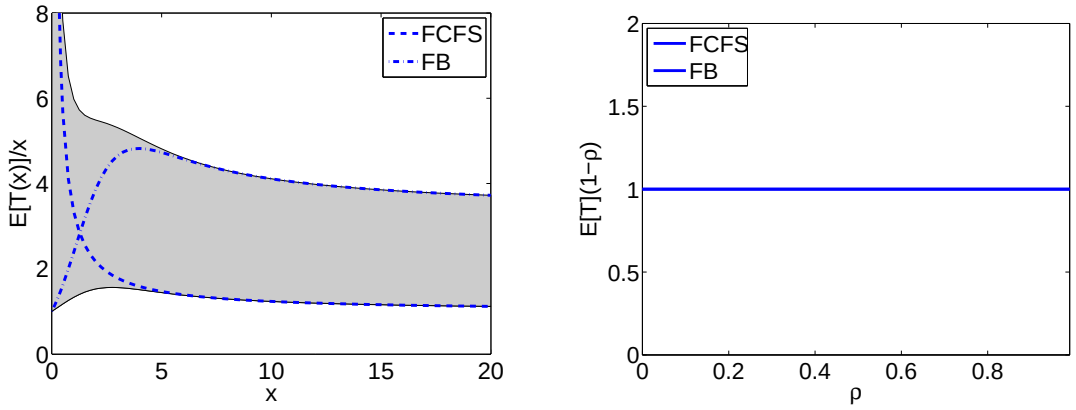
5.3.1 Defining a class of age based policies

Formally, we define age based policies as follows.

Definition 5.3 *Under an **age based policy**, the priority of a job is assigned based on a fixed priority function that is a bounded bijection from ages to priorities. The priority of a job is updated as the age (attained service) of the job changes. The job with the highest priority is preemptively given service, and if two jobs have the same age (and thus priority), the job that attained that age first is given higher priority.*

Clearly, FCFS obeys this definition. To see that FB obeys this definition, observe that under FB , priority is strictly decreasing with age. Thus, a new arrival will run alone until it achieves the age, a, of the youngest job in the system; and then those jobs of age a will timeshare. This timesharing is caused by the fact that if one job starts to run, its priority will drop, causing a different job to immediately run, and so on.

In addition to these two common policies, a wide range of other disciplines fall into the class of age based policies. For instance, one can imagine a priority curve that assigns high priority to jobs having both small and large ages in order to relieve fairness concerns.



(a) $E[T(x)]/x$ (b) $E[T](1-\rho)$

Figure 5.4: *This figure illustrates the bounds on $E[T(x)]$ and $E[T]$ under the class of age based policies. The shaded area indicates the response times attainable using age based policies. In addition, the behaviors of the two most common age based policies,* FCFS *and* FB*, are illustrated In both plots the service distribution is taken to be Exponential with mean 1, and the load in (a) is 0.7. Recall that all blind policies, which includes all age based policies, have equivalent $E[T]$ when the service distribution is Exponential; however the response times of these policies differ under any distribution that does not have a constant failure rate.*

We will find when studying the behavior of higher moments of $T(x)$ under age based policies that the age based policies which are non-preemptive (and thus equivalent to FCFS) behave quite differently from the rest of the age based policies. In order to separate such policies from the rest of the class of age based policies, we define the following subclass of age based policies.

Definition 5.4 *The class of **preemptive age based policies** is a subclass of age based policies where there exists some finite age a such that the priority of jobs with age a is lower than the priority of jobs with age 0, i.e. the age based policy is not equivalent to a non-preemptive policy.*

5.3.2 Bounding response times for age based policies

We now move to the task of bounding the response times of age based policies. The behavior of age based policies is quite interesting. For example, in the special case of an Exponential service distribution, all age based policies have the same $E[T]$, despite the fact that $E[T(x)]$ can behave very differently under different policies in this setting.

Though age based policies have not been studied prior to this work, the more general class of blind scheduling policies (policies "blind" to job size information) has been studied, so many of the results here are special cases of results proven for the class of blind policies.

We begin by presenting a bound on $E[T(x)]$ under age based policies proven by Kleinrock [121].

Proposition 5.3
For any age based policy P,

$$x + \frac{\lambda \widetilde{m_2}(x)}{2(1-\widetilde{\rho}(x))} \leq E[T(x)]^P \leq \frac{x}{1-\widetilde{\rho}(x)} + \frac{\lambda E[X^2]}{2(1-\widetilde{\rho}(x))(1-\rho)}$$

The bounds in Proposition 5.3 are illustrated in Figure 5.4.

Note that for all x, there is some age based policy P that has $E[T(x)]^P$ equal to the bounds in Proposition 5.3. However, despite the fact that these bounds are tight, they do little to identify the behavior of $E[T(x)]/x$ under age-based policies. For instance, the bounds seem to allow the possibility that a policy has $E[T(x)]/x$ significantly lower than $1/(1-\rho)$ for all x. However, in Chapter 7 we will be prove that that all age based policies have some peak where $E[T(x)]/x > 1/(1-\rho)$.

Further, despite the fact that the bounds on $E[T(x)]/x$ are tight, they provide little information about the behavior of age based policies with respect to $E[T]$. In many cases it is possible to obtain much better bounds on $E[T]$ using other techniques. For example, if the service distribution has a decreasing failure rate, we know that $E[T]$ is minimized under FB and maximized under FCFS (among blind scheduling polices) [188, 189]. And if the service distribution has an increasing failure rate the opposite is true. However, under distributions that do not have monotonic failure rates, it is not clear how to derive tight bounds on $E[T]$ under age based policies.

We will wrap up this section by contrasting the behavior of $E[T(x)]$ and $E[T]$ under age based policies with the behaviors we have already observed for remaining size based and preemptive size based policies. Though the bounds on $E[T(x)]$ under age based policies are quite broad, similarly to those on $E[T(x)]$ under preemptive size based and remaining size based policies, the bounds on $E[T]$ under age based policies are much more restrictive than the bounds on $E[T]$ under either preemptive size based or remaining size based policies. In particular, all age based policies are equivalent when the service distribution is Exponential. Further, though differences do appear when the service distribution is non-Exponential, the differences only become severe under service distributions with high variability and the differences are never as severe as in cases of either remaining size based or preemptive size based policies.

5.4 The class of non-preemptive policies

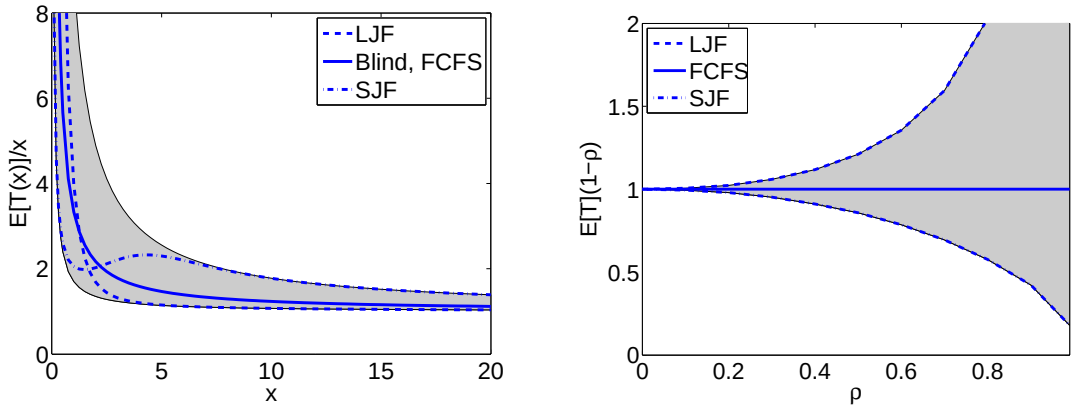
So far, all the scheduling techniques we have discussed are preemptive, they allow jobs to be interrupted and restarted without penalty. However, in many applications preemption is costly and thus only non-preemptive polices are appropriate, e.g. databases and supercomputers.

In this section, we study the class of non-preemptive policies; thus the results apply to non-preemptive blind policies such as FCFS and ROS, as well as non-preemptive size based policies such as SJF and LJF.

5.4.1 Defining classes of non-preemptive policies

Formally, the class of non-preemptive policies is defined as follows.

Definition 5.5 *Under a **non-preemptive policy** a job cannot be interrupted once it has begun service.*



(a) $E[T(x)]/x$ (b) $E[T](1-\rho)$

Figure 5.5: *This figure illustrates the bounds on $E[T(x)]$ and $E[T]$ under the class of non-preemptive policies. The shaded area indicates the response times attainable using non-preemptive policies. In addition, the behaviors of the three most common non-preemptive policies (*FCFS*,* SJF*, and* LJF*) are illustrated. Recall that all blind non-preemptive policies have $E[T(x)]$ and $E[T]$ equal to that under* FCFS*. The service distribution is taken to be Exponential with mean 1, and the load in (a) is 0.7.*

However, in some cases (especially when considering fairness metrics) it will be useful to divide up non-preemptive policies into two subclasses: policies that are blind to job size information and policies that schedule using job size information.

Definition 5.6 *Under a **non-preemptive blind policy** a job cannot be interrupted once it has begun service and no job size information can be used to make scheduling decisions.*

Definition 5.7 *Under a **non-preemptive size based policy**, the priority of a job is assigned based on a fixed priority function that is a bounded bijection from job sizes to priorities. Priorities are assigned upon arrival and cannot be adjusted. The job with the highest priority is run non-preemptively, and if two jobs of the same size (and thus priority) are in the queue, then the job that arrived first is given higher priority.*

Thus, the class of non-preemptive blind policies includes, among other, FCFS, ROS, and LCFS. In contrast, the class of non-preemptive size based policies includes SJF, LJF, and a range of policies with more complex priority curves.

5.4.2 Bounding response times for non-preemptive policies

Unlike the scheduling techniques we have described so far in this chapter, all non-preemptive policies behave very similarly with respect to both $E[T(x)]$ and $E[T]$. This is because, in many cases, response times of non-preemptive policies are dominated by the excess of the job in service.

This factor is easily seen in the following bounds on $E[T(x)]$ under non-preemptive policies:

Proposition 5.4
In an M/GI/1, for any non-preemptive policy P,

$$x + \frac{\lambda}{2}E[X^2] \leq E[T(x)]^P \leq x + \frac{\lambda E[X^2]}{2(1-\rho)^2}$$

Proof. The lower bound follows from the fact that, at minimum, at job must wait behind the excess of the current job in service. Further, the upper bound follows from the fact that, at most, an arriving tagged job must wait behind $B(Q)$, where Q is the steady-state work in the system.
□

These bounds are illustrated in Figure 5.5.

Clearly, the fact that a policy is non-preemptive already provides a lot of information about the behavior of $T(x)$, even without any information about how the policy prioritizes. In fact, using only these bounds, we will be able to classify the fairness of non-preemptive policies in Chapter 7.

Moving to the bounding the overall mean response time, $E[T]$, we can again obtain tight bounds. In particular, Phipps has proven that SJF minimizes $E[T]$ [175], and it follows from a parallel argument that LJF maximizes $E[T]$. Further, as we discussed in Chapter 3 all non-preemptive blind policies have equivalent $E[T]$. Summarizing the above, we have

Proposition 5.5
In an M/GI/1 queue, for any non-preemptive policy P,

$$E[T]^{SJF} \leq E[T]^P \leq E[T]^{LJF}.$$

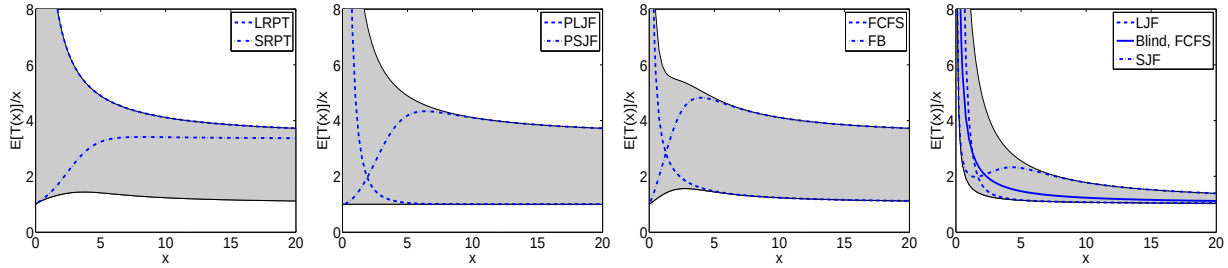
Further, for any blind non-preemptive policy P, $E[T]^P = E[T]^{FCFS}$.

These bounds are illustrated in Figure 5.5.

The combination of the bounds in Proposition 5.4 and Proposition 5.5 present an enormous contrast to the bounds we have proven on other scheduling techniques in this chapter. In particular, as illustrated in Figure 5.5 the bounds on $E[T(x)]/x$ give an good idea of the behavior of all non-preemptive policies, which has not been the case for the bounds on the other scheduling techniques discussed in this chapter, e.g. preemptive size based and age based policies. Further, because of the dominant effect the excess of the job at the server has on the overall mean response time of non-preemptive policies, even the behavior of $E[T]$ is well characterized by the simple bounds in Proposition 5.5. Again, this is a huge contrast to the bounds on $E[T]$ under remaining size based and preemptive size based policies, which are too disparate to provide a useful understanding of any individual policy.

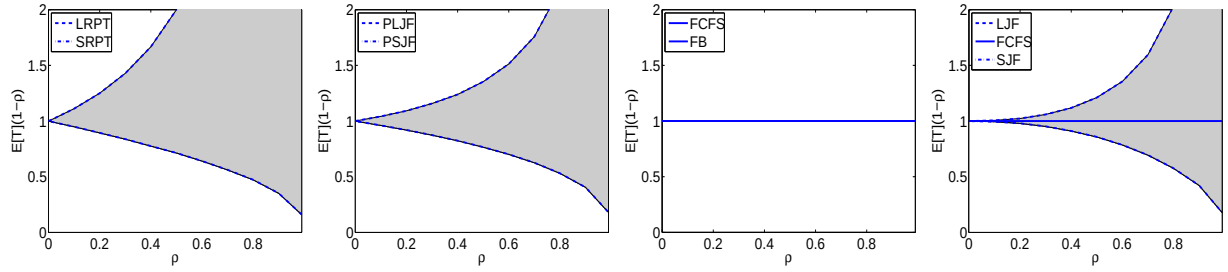
5.5 Concluding remarks

In this chapter, we have defined formal classifications of scheduling policies based on *scheduling techniques*. We focused on four particular techniques that span a wide range of applications: remaining size based scheduling, preemptive size based scheduling, age based scheduling, and non-preemptive scheduling. In



(a) Remaining size based policies (b) Preemptive size based policies (c) Age based policies (d) Non-preemptive policies

Figure 5.6: *These figures illustrate the bounds on $E[T(x)]/x$ under scheduling techniques. The service distribution is taken to be exponential with mean 1, and the load is 0.7.*



(a) Remaining size based policies (b) Preemptive size based policies (c) Age based policies (d) Non-preemptive policies

Figure 5.7: *These figures illustrate the bounds on $E[T]$ under scheduling techniques. The service distribution is taken to be exponential with mean 1.*

addition to defining the four classifications, we began our analysis of each class by proving simple bounds on the overall and conditional mean response times attainable by policies in each class. These bounds go a long ways towards illustrating the impact of the scheduling technique used by a policy on its performance.

Figures 5.6 and 5.7 illustrate the bounds on the attainable response times under each scheduling technique. There are a few notable contrasts that are worth expanding on.

The most obvious contrast in Figure 5.6 is how different the bounds on non-preemptive polices are from the bounds on the other technique classifications. In particular, under preemptive size based policies, remaining size based policies, and age based policies the behavior of $E[T(x)]$ can be quite varied; while all non-preemptive policies have a strongly decreasing trend. To isolate the impact of preemption, it is interesting to contrast the behavior of preemptive size based policies and non-preemptive size based policies. Though both classes include size based policies, the the impact of preemption is dramatic. In particular, looking at the behavior of $E[T(x)]/x$ we see that all non-preemptive policies give very large job sizes approximately the same $E[T(x)]/x$ while preemptive size based policies can differ significantly in terms of the response times of large jobs. Further, notice that non-preemptive policies treat large job sizes nearly

optimally. Thus, even when a non-preemptive policy biases against large job sizes, large jobs have near optimal response times, while under preemptive size based policies biasing against large job sizes leads to a significant increase in the response time of large job sizes.

Another interesting comparison across scheduling techniques is the comparison of $E[T]$ illustrated by Figure 5.7. Under many of the scheduling heuristics $E[T]$ can be quite disparate. For instance, remaining size based policies can achieve both the optimal and worst possible $E[T]$. Similarly, preemptive size based policies can be both near optimal and very far from optimal. In contrast, other scheduling techniques guarantee very similar mean response times in certain settings. For instance, when job sizes are exponentially distributed, all age based policies are equivalent and non-preemptive policies all have very similar response times. However, the bounds on $E[T]$ under these scheduling techniques can also become quite disparate when the job size distribution has high variability.

The scheduling technique based classifications discussed in this chapter are complementary to the heuristic classifications introduced in Chapter 4. By comparing the two types of classifications we can develop a new understanding of the differing impacts of scheduling techniques and heuristics. Understanding these different impacts will be a key theme throughout the thesis; however we can already begin to make a few observations by recalling Figures 4.12 and 4.13 and comparing them to Figures 5.6 and 5.7.

The biggest contrast in these figures is that the scheduling heuristic used plays a more defining role in determining $E[T]$ than the scheduling technique. Some scheduling techniques guarantee similar $E[T]$ under certain distributions (e.g. the age based and non-preemptive techniques under the exponential distribution); however to a large extent the scheduling technique used provides little information about the behavior of $E[T]$. For instance, under service distributions with large variance, all four scheduling techniques contain policies with disparate $E[T]$. In contrast, all of the scheduling heuristics isolate $E[T]$ across all service distributions. Prioritizing small jobs always leads to near optimal $E[T]$ while prioritizing large jobs always leads to very large $E[T]$. Further, PROTECTIVE and SYMMETRIC scheduling always lead to similar $E[T]$.

Similarly, when comparing the behavior of $E[T(x)]/x$ under scheduling heuristics with that under scheduling techniques, we see that the scheduling heuristic used plays a much larger role in determining the behavior of $E[T(x)]/x$. Under the scheduling heuristics, all the policies in each class have similar functional behavior: they are all increasing/decreasing/constant across x. However, under the scheduling technique classifications, each class has policies with a range of behaviors. For instance, the class of remaining size based policies includes both policies that have $E[T(x)]/x$ increasing with x and policies that have $E[T(x)]/x$ decreasing with x.

The trend of the scheduling heuristic playing a defining role in the performance of the policy will extend throughout the thesis under both fairness and efficiency measures. However, we will see in Chapter 7 that, with respect to fairness measures, the scheduling technique used provides a more useful way of understanding performance. Further, some scheduling techniques, especially the class of non-preemptive policies, will play a strong role in determining the performance of policies under other efficiency measures such as the behavior of the tail of the response-time distribution.

Let us end this chapter with one final note. Chapters 4 and 5 have introduced a wide variety of classifications. These classifications are in many cases novel, e.g. the SMART, SMART$_\epsilon$, FOOLISH, preemptive size based, remaining size based, and age based classifications. Other classifications have been studied for many years, but are viewed in a new light in by this thesis, e.g. the SYMMETRIC and non-preemptive classifications. Another, the PROTECTIVE classification, is an example of a new classification that was

introduced by the new focus on classifications provided by this thesis. Further, the PROTECTIVE classification is not the only example of other researchers developing new classifications, motivated by the work of this thesis. In particular, following the work of Wierman & Harchol-Balter [238] there has been a growing focus of research introducing and analyzing scheduling classifications. For example, many other researchers also became interested in the SMART class, and this led to a collaboration with Bert Zwart and Misja Nuyens analyzing the distribution of response time under SMART policies in the large buffer large deviations regime (see Chapter 6 [161]. In addition, it led to a collaboration with Sanjay Shakkottai and Chang Woo Yang on the analysis of SMART policies in the many sources large deviations regime [248]. Further, other researchers have started to introduce their own scheduling classifications. In addition to the PROTECTIVE class that we discussed in Chapter 4, Feng, Misra, & Rubenstein [74], Nunez-Queija & Kherani [118], and Kherani [117] have all introduced interesting classifications of other scheduling techniques and heuristics.

PART III

Diverse Metrics: Moving Beyond Mean Response Time

Until this point in the thesis, we have focused almost entirely on characterizing the mean response time under a range of individual policies and scheduling classifications. However, in practice, many other metrics are also important to computer systems. It is not enough for a new design to provide an improved mean response time, it must also guarantee fairness, provide quality of service (QoS) guarantees, limit buffer overflows, limit power usage, etc. In Part III of the thesis, we move beyond the study of mean response time and consider a variety of other metrics that are important across applications. In Chapter 6 we study the distributional behavior of response time and in Chapter 7 we introduce and study a variety of fairness metrics.

Extending our discussion beyond mean response time to the distribution of response time is essential for applicability in modern computer applications because users can become even more frustrated by highly variable service times than by having large response times on average. Further, providing QoS guarantees under scheduling policies depends on knowledge of the distribution of response times. Unfortunately, studying the distribution of response times under scheduling policies directly is known to be an extremely difficult task. As a result, in Chapter 6, we study only the tail behavior of the distribution, i.e. we study $P(T > x)$ as $x \to \infty$. By limiting ourselves to this asymptotic regime we are able to derive new results characterizing the response time distribution under both common individual scheduling policies and scheduling classifications.

Extending our discussion to consider fairness metrics is also essential to the applicability of our results to real systems. Fairness metrics are important in any computer system where there are human users. Although typically not the primary metric of interest, it is important that low priority users, which are in many cases the users with large service demands, are not starved of service in order to obtain efficiency gains. However, fairness is an amorphous concept, and thus is difficult to define. This difficulty has traditionally stifled research into the fairness of scheduling policies. In Chapter 7, we introduce a variety of novel measures of fairness that are motivated by computer applications such as routers and web servers. In addition, we analyze both common individual scheduling policies and scheduling classifications with respect to these new fairness metrics.

CHAPTER 6

The distribution of response time

Until this point in the thesis we have focused almost entirely on the mean response time of scheduling policies. Though, providing small mean response times is typically the primary goal for computer applications, extending our discussion beyond the mean response time to the distribution of response time is essential for real world applicability. In fact, users have been shown to prefer response times that are larger on average if the response times are less variable, and thus more predictable [65, 255]. Further, understanding the distributional behavior of response time is fundamental when considering QoS, admission control, and capacity planning applications where guarantees of the form "90% of the time the response time is $< C$" are desired.

Clearly, the study of the distribution of response times is important, and as a result understanding it is a classical problem in queueing. However, it is a problem that has proven to be extremely difficult, especially in the case of complex scheduling policies. In particular, exact derivations of $P(T > x)$ are only possible in very specialized settings, such as the M/M/1, and under only very simple policies, such as FCFS. Instead, the traditional approach for understanding $P(T > x)$ has been to derive the Laplace transform of response time. However, though obtaining the Laplace transform of response time is useful in characterizing the moments of response time, the "Laplace curtain" hides the behavioral properties of the distribution – other techniques are necessary to "see through the curtain."

Due to the difficulty of exact analysis, modern studies of the response time distribution tend to either use (i) numerical techniques or (ii) asymptotic techniques. Numerical approaches typically to rely on either using transform inversion techniques, e.g. [7, 8], or using phase type (PH) service and arrival distributions combined with matrix analytic techniques, e.g. [125, 163]. We will apply these matrix analytic techniques in Chapter 9 when we analyze multiserver systems. However, such an approach is not ideal because sit can only be applied for light-tailed service distributions, and heavy-tailed service distributions are prevalent in computer applications [28, 69, 127, 174]. Asymptotic approaches to studying the distribution of $P(T > x)$ have no such limitations – they can be applied to both light-tailed and heavy-tailed service distributions.

This chapter will focus on one particular asymptotic regime for studying $P(T > x)$: the *large buffer large deviations* regime. Under the large buffer regime, the asymptotic tail of response time is studied, i.e. the behavior of $P(T > x)$ as $x \to \infty$ is characterized. This is a natural asymptotic regime to study since it provides bounds on the likelihood of large delays, which are exactly what QoS and buffer provisioning applications require. Results in the large buffer framework provide an understanding of what "critical events"

lead to large response times, and thus, how scheduling can and priority mechanisms can limit the likelihood of large delays.

The large buffer framework has received an enormous amount of attention over the last decade. Motivated by the emergence of measurements indicating heavy-tailed file size distributions in many computer applications [28, 69, 127, 174], there has been an explosion of results analyzing the tail behavior of a variety of scheduling policies in the heavy-tailed setting. Beginning with results for FCFS [40, 60, 169], and eventually leading to the analysis a a wide range of policies such as PS [41, 90], PLCFS [141], SRPT [158], LCFS [40], FB [158, 159], and others. The results for these individual policies are quite illustrative. In particular, two common behaviors have emerged: common policies either (i) have a response time tail proportional to the tail of the service distribution (e.g. SRPT and PS) or (ii) have a response time tail proportional to the tail of the residual of the service distribution, i.e. the tail of the excess of the service distribution (e.g. FCFS and LCFS).

In the process of this explosion of work, four general analytic approaches emerged. Results were obtained either from (i) an analytical approach relying on Tauberian theorems relating the tail behavior of a distribution to the Laplace transform of the distribution; (ii) a probabilistic approach using Markov's inequality to relate the occurrence of a large response time to the occurrence of a large service demand; (iii) a sample path approach that directly characterizes the critical event; or (iv) a probabilistic approach based on an explicit random walk representation of the waiting time distribution. We will provide an illustration of each of these techniques in this chapter, however the interested reader can also refer to the excellent survey paper by Borst et al. [40].

Following the explosion of results in the heavy-tailed setting, in the past few years, there has been a growth in work studying the tail behavior of response time in the light-tailed setting. Many of the same proof techniques apply to both the heavy and light-tailed settings; thus analyses of many common policies have quickly emerged: FCFS [181], PS [135, 41, 72], SRPT [162], PLCFS [170], FB [134, 161], ROS [135], and others. As in the heavy-tailed setting, it seems that two types of tails are emerging: policies either (i) have a response time tail proportional to the stationary workload in the queue (e.g. FCFS) or (ii) have a response time tail proportional to the length of a busy period (e.g. SRPT and PS). Interestingly, these two behaviors, in a sense, parallel what happens in the heavy-tailed setting: in the heavy-tailed setting a busy period has the same tail as the service distribution and the stationary workload has the same tail as the residual of the service distribution. The difference is that, in the light-tailed setting, the tail of a busy period is heavier than the tail of the workload, while the opposite is true in the heavy-tailed setting. Thus, there is a general trend indicating that policies which behave well under heavy-tailed service distributions behave poorly under light-tailed service distributions. This contrast between the behavior of scheduling policies under light and heavy-tailed service distributions has spurred research in this area.

Our goal in this chapter is to shed light on the contrast between the behavior of scheduling policies in the light and heavy-tailed settings by taking a new approach to the study of response time tails: analyzing the behavior of *scheduling classifications* instead of focusing on individual policies. By characterizing the behaviors of scheduling classifications we hope to develop an understanding of the scheduling heuristics and techniques that underly the contrast between the light-tailed and heavy-tailed settings.

However, before we can study scheduling classifications, we must begin by understanding the behavior of individual policies. Thus, in Section 6.2 we will provide an overview of results about individual scheduling policies. This section will include results about FCFS, SRPT, PS, FB, and LCFS. Note that some of these results are new to this thesis. For example, we extend the analysis of FB and SRPT from the M/GI/1

setting to the GI/GI/1 setting. In addition, this section will include a number of proof sketches illustrating the common techniques used in this area.

After taking a thorough look at the behavior of individual policies, in Section 6.3, we will move to the analysis of scheduling classifications. In this section, we will derive results characterizing the response time tail under the class of non-preemptive policies, the SMART class, and the FOOLISH class. These results represent a departure from the standard results in the field because of the focus on *classes* of policies instead of *individual* policies. The benefits this of this new focus are that (i) it provides a deeper understanding of what in a policy determines the behavior of the response time distribution, and (ii) it allows the results to be applied not only to the idealized policies studied in theory, but also to the policies that are actually implemented in practice.

The analyses of the tail behavior of response time under these scheduling classifications illustrates a number of important contrasts between scheduling heuristics/techniques. For instance, we will show that SMART policies provide an asymptotically optimal response time tail when the service distribution is heavy-tailed, but provide a response time tail that is as heavy as possibly under light-tailed service distributions. In contrast, FOOLISH policies have response time tails that are as heavy as possible under both light-tailed and heavy-tailed service distributions. Similarly, non-preemptive policies have as heavy a response time tail as possible under heavy-tailed service distributions, but can have an asymptotically optimal response time tail under light-tailed service distributions. These results shed new light on the reasons for the contrasting behavior of individual policies under light-tailed and heavy-tailed service times. Further, these results have a clear impact for system design. In particular, they highlight the need for understanding the tail behavior of job sizes before making design decisions about which scheduling policy to use. Further, the derivations of the results provide insight into the causes of large delays under different policies. For example, under non-preemptive policies, the analysis formalizes the idea that when a tagged job experiences a long delay it is likely due to a large job being at the server when the tagged job arrives. In contrast, under SMART policies, the analysis illustrates that when a tagged job experiences a long delay it is likely the result of a burst of arrivals (having smaller sizes) arriving just after the arrival of the tagged job.

6.1 Preliminaries

Before proceeding, it is important that we spend some time introducing the notation and distributions that we will be using in this chapter. We will be considering two classes of service distributions, one heavy-tailed and one light-tailed.

The class of *heavy-tailed* distributions that we will focus on are those of intermediate regular variation, $\mathcal{IR}$, see Section 2.4.2.3 for an introduction to these distributions. This class generalizes the class of regularly varying distributions, and thus includes Pareto distributions. In addition it includes policies that "dominate" a Pareto tail. In addition to $\mathcal{IR}$ distributions, we will occasionally discuss the broader class of subexponential distributions, $\mathcal{S}$. Refer to Section 2.4.2.4 for an introduction to these distributions.

The class of *light-tailed* distributions we study obeys the following assumptions:

Assumption 6.1 $\mathcal{M}_X(s) < \infty$ *for some* $s > 0$.

Assumption 6.2 $P(X = x_F) = 0$.

Note that the distributions that satisfy both of these assumptions include light-tailed distributions with infinite endpoints (e.g., exponential, gamma, and certain Weibull distributions), as well as all continuous distributions with finite support (e.g., uniform and beta distributions).

When studying light-tailed distributions, we will describe the logarithmic behavior of the tail of the response time distribution using the *decay rate*.

Definition 6.1 *The **(asymptotic) decay rate** $\gamma(Y)$ of a random variable Y is defined by*

$$\gamma(Y) = \lim_{x\to\infty} \frac{-\log P(Y > x)}{x},$$

given that the limit exists.

Informally, for large x, one may write $P(Y > x) \approx e^{-\gamma(Y)x}$. It should be noted that a smaller decay rate corresponds to a larger tail of the distribution.

In both the light and heavy tailed case, our analysis will depend heavily on the use of busy periods (B). Thus, it is important to recall the wide variety of busy periods that we introduced in Section 3.2.1. Further, it is important to understand the tail behavior of the busy period in both the light-tailed and heavy-tailed settings.

In the heavy-tailed setting, De Meyer and Teugels [141] have proven

$$P(B > x) \sim P(X > (1-\rho)x) \quad \text{as } x \to \infty \tag{6.1}$$

in an M/GI/1 queue with regularly varying job sizes. Further, (6.1) has been shown to hold in a GI/GI/1 queue under the more general class of subexponential distributions if the distribution is also *square-root insensitive*, i.e. if $P(X > x) \sim P(X > x - \sqrt{x})$ as $x \to \infty$ [21, 105]. Note that, for example, the Pareto distribution is always square-root insensitive but the Weibull distribution is not.

In the light-tailed setting, the decay rate of the busy period can be expressed in terms of the moment generating functions of the interarrival times, A, and the service times, X [162]. We state the result here in a form that will be of use later in our analysis of SMART policies.

Lemma 6.1
For $0 < x \leq \infty$,

$$\gamma(B_x) = \sup_{s\geq 0}\left[s + \mathcal{M}_A^{-1}\left(\frac{1}{\mathcal{M}_{XI(X<x)}(s)}\right)\right]$$
$$\gamma(\widetilde{B_x}) = \sup_{s\geq 0}\left[s + \mathcal{M}_A^{-1}\left(\frac{1}{\mathcal{M}_{X\wedge x}(s)}\right)\right]$$
$$\gamma(B) = \sup_{s\geq 0}\left[s + \mathcal{M}_A^{-1}\left(\frac{1}{\mathcal{M}_B(s)}\right)\right]$$

Note that the expressions for $\gamma(B), \gamma(B_x)$ and $\gamma(\widetilde{B_x})$ can in general be solved numerically. However, if the arrival process is Poisson with rate λ, we can obtain a more explicit formula:

$$\gamma(B) = \sup_{s\geq 0}\left[s - \lambda(\mathcal{M}_X(s) - 1)\right]$$

Further, specializing this to the $M/M/1$ queue, where the service times have an exponential distribution with rate μ, we get the expression

$$\gamma(B) = \mu(1 - \sqrt{\rho})^2.$$

6.2 The response time tail under individual policies

With the preliminaries out of the way, we are now ready to study the tail behavior of response time under common individual scheduling policies. Our goal here is twofold. First, and foremost, we hope to provide an overview of the known results about tail behavior for the common policies studied in the queueing literature so that the results we will later prove about classes of policies can be placed in the greater context of the field. However, just as importantly, we also hope to provide an overview of the common analytic techniques used in the literature so that a non-expert can more easily understand the arguments we will use to analyze scheduling classifications.

We will organize this subsection as follows. We will have subsections for each of the common individual policies. In each case, we will survey results characterizing the tail behavior of response time under both light-tailed and heavy-tailed service distributions. In addition, for many policies we will provide an overview of the derivation of the results, and in some cases we will provide multiple derivations for the same policy in order to contrast the intuition provided by each argument.

6.2.1 FCFS

As you might expect, FCFS was one of the first policies for which the tail behavior of response time was studied. Initially, the behavior of FCFS was studied under light-tailed service distributions in simple models such as the M/M/1. Then, as empirical observations provided increasing support for the importance of heavy-tailed distributions in practice, the focus of analysis shifted to the heavy-tailed setting. Thus, we will start by providing a brief summary of results about FCFS in the light-tailed setting, and then we will survey the results and techniques for studying FCFS in the heavy-tailed setting.

Light-tailed service times

The first results about the tail behavior of FCFS emerged in simple queueing models and are part of most introductory queueing theory texts, e.g. [119, 247, 222]. For instance, it is quite straight forward to study the tail behavior of FCFS in simple models such as the M/M/1. Specializing the Pollaczek-Khinchin transform formula to the case of the M/M/1 yields

$$\mathcal{L}_T(s)^{FCFS} = \frac{\mu(1-\rho)}{s + \mu(1-\rho)}$$

where μ is the service rate. This is clearly the transform of an exponential distribution, thus we can invert the transform to obtain the following p.d.f. of the response time

$$f_T(x)^{FCFS} = \mu(1-\rho)e^{-\mu(1-\rho)x}$$

However, the analysis of the GI/GI/1 tail behavior of FCFS under light tailed service times did not emerge until much later. It is easy to see that the decay rate of FCFS matches the decay rate of the workload; however, deriving an explicit form for the decay rate of the workload is more difficult. We defer the details of the proof to [17, 162] and simply state the result here. In particular, we have that (under Assumption 6.1)

$$\log P(Q > x) \sim -\gamma(Q)x \quad \text{as } x \to \infty$$

where

$$\gamma(Q) = \sup\{s : \mathcal{M}_A(-s)\mathcal{M}_X(s) \leq 1\}$$

To understand this decay rate, it is useful to contrast it with the decay rate of a busy period. It is easy to see that $\gamma(B) \leq (1-\rho)\gamma(Q)$ for all $\rho < 1$ (see [162] for the details). Thus, the tail of the busy period is always heavier than that of the workload. In fact, Ramanan and Stolyar have shown that the decay rate of the workload is as large as possible (thus the tail is as light as possible) [181].

Heavy-tailed service times

The first analysis of FCFS in the heavy-tailed setting was provided by Cohen [60], who showed that

$$P(W^{FCFS} > x) \sim \frac{\rho}{1-\rho} P(\mathcal{E} > x), \quad \text{as } x \to \infty \tag{6.2}$$

in the GI/GI/1 queue when the service distribution is regularly varying. However, many other authors have since studied the tail behavior of FCFS, and (6.2) has been shown to hold in much more general settings. In particular, Pakes proved that (6.2) holds whenever the excess of the service distribution is subexponential [169]. In addition, Korshunov [122] established a converse result for the GI/GI/1 which shows that $\mathcal{E} \in \mathcal{S}$ is not only sufficient, but is also necessary. That is

$$\mathcal{E} \in \mathcal{S} \quad \Leftrightarrow \quad W^{FCFS} \in \mathcal{S} \quad \Leftrightarrow \quad P(W^{FCFS} > x) \sim \frac{\rho}{1-\rho} P(\mathcal{E} > x)$$

The importance of subexponentiality in this setting should not be too surprising given the background we have provided on the composition of the workload in the FCFS queue. In particular, we can recall from (3.3) that the waiting time under FCFS has a random sum decomposition. In particular, we have that

$$P(W > x) = (1-\rho) \sum_{n=0}^{\infty} \rho^n P(\mathcal{E}_1 + \ldots + \mathcal{E}_n > x)$$

From the definition of subexponential distributions in Section 2.4.2.4 it is clear that subexponential distribu-

tions are extremely well-suited to analyze such a random sum. In particular, if $\mathcal{E} \in \mathcal{S}$, we have

$$\begin{aligned} P(W > x) &= (1-\rho)\sum_{n=0}^{\infty} \rho^n P(\mathcal{E}_1 + \ldots + \mathcal{E}_n > x) \\ &\sim (1-\rho)\sum_{n=0}^{\infty} \rho^n n P(\mathcal{E} > x) \\ &= \frac{\rho}{1-\rho} P(\mathcal{E} > x) \end{aligned}$$

Note that the interchange of the summation and the limit can be justified using the dominated convergence theorem in combination with an upper bound such as the one in [18] which says that if $\mathcal{E} \in \mathcal{S}$, and $\epsilon > 0$, then there exists a K such that $P(\mathcal{E}_1 + \ldots + \mathcal{E}_n) > x) \leq K(1+\epsilon)^n P(\mathcal{E} > x)$.

Clearly, subexponential distributions provide a useful class for studying the tail behavior of FCFS in the heavy-tailed setting. However, due to its simplicity, FCFS provides a venue for illustrating some other common analytic techniques. Thus, we will also sketch two other derivations of (6.2): one using a transform approach and one using a sample path argument.

We will start by illustrating the transform approach. Whenever the transform of the quantity being studied is known, a simple way to derive the tail asymptotics of the transform is to apply a Tauberian theorem, such as Theorem 2.2 for regularly varying distributions. Combining Theorem 2.2 with the P-K transform formula (see (3.2)) we can easily derive the asymptotic tail behavior of W^{FCFS} in the M/GI/1 setting with a regularly varying service distribution. For illustrative purposes, let us assume that $\overline{F} \in \mathcal{RV}(\alpha)$ for $1 < \alpha < 2$, however this can easily be extended. To begin, recall that when X is $\mathcal{RV}(\alpha)$, $\mathcal{E}$ is $\mathcal{RV}(\alpha-1)$. Thus, applying Theorem 2.2, we have that

$$1 - \mathcal{L}_{\mathcal{E}}(s) = 1 - \frac{1-\mathcal{L}_X(s)}{E[X]s} = -\left(\frac{\Gamma(1-\alpha)}{E[X]} + o(1)\right) s^{\alpha-1} L(1/s), \quad \text{as } s \downarrow 0,$$

where $L(\cdot)$ is a slowly varying function. Further, we have that

$$\mathcal{L}_W(s)^{FCFS} = \frac{1-\rho}{1-\rho L_{\mathcal{E}}(s)} \sim \frac{\rho}{1-\rho} \frac{\Gamma(1-\alpha)}{E[X]} s^{\alpha-1} L(1/s)$$

Another application of Theorem 2.2 gives (6.2).

The the transform argument that we just worked through is simple and direct, but it provides little intuition for why (6.2) holds. The sample-path approach we will describe now is a technique for formalizing intuition about the *cause* of a large delay. In particular, under FCFS the cause of a large delay is intuitively the arrival of one very large job (when the service distribution is heavy-tailed). Amazingly, it turns out that it is possible to formalize this simple heuristic into an argument for obtaining tail asymptotics by simply computing the probability of this scenario occurring and then showing that all other scenarios happen with negligible probability.

To illustrate how such a derivation is performed, let us focus on the workload in the system at some time $t = 0$. Our intuition tells us that a large workload at $t = 0$ is likely due to the arrival of a large job at some prior time, $t = -y$. Following time $t = -y$, the workload in the queue decreases roughly linearly at rate $1 - \rho$ (since ρ work arrives and the server works at rate 1). Thus, in order for the workload to exceed x at

$t = 0$, the large job that arrived must be larger than $x + y(1 - \rho)$. Now, in the M/GI/1 setting we have an arrival rate of λ, so we have

$$\begin{aligned} P(W^{FCFS} > x) &\approx \int_{y=0}^{\infty} P(X > x + y(1-\rho))\lambda dy \\ &= \frac{\lambda}{1-\rho}\int_{z=x}^{\infty} P(X > z)dz \\ &= \frac{\rho}{1-\rho}P(\mathcal{E} > x) \end{aligned}$$

Note that the second step follows from the change of variables $z = x + y(1 - \rho)$ and the final step follows from recalling that $f_{\mathcal{E}}(x) = P(X > x)/E[X]$.

Obviously the above is only a sketch of the argument, however this sketch can be made into a rigorous lower bound on $P(W^{FCFS} > x)$ with only a little additional effort. But, deriving a matching upper bound turns out to be much more difficult, and we refer the reader to [40] or [257] for a more comprehensive description.

6.2.2 SRPT

The analysis of the tail of SRPT requires a much different approach than those that we just illustrated for FCFS. Results about the tail behavior of response time under SRPT first emerged in the heavy-tailed setting, so we will begin there and then move to the light-tailed setting.

Heavy-tailed service times

Analyzing the tail behavior of SRPT was an open problem until Nunez-Queija [158] proved that, in the M/GI/1 with $\mathcal{RV}$ service times, SRPT is (what he termed) a "tail-equivalent" policy, i.e. the tail of response times are the tail of service times are equally heavy:

$$P(T^{SRPT} > x) \sim P(X > (1-\rho)x) \quad \text{as } x \to \infty \tag{6.3}$$

This tail behavior is an enormous improvement over the behavior we saw under FCFS, where the tail of response times was as heavy as the tail of the excess of the service times. Further, (6.3) illustrates that SRPT has an asymptotically optimal response time tail in this setting since it is impossible to have a response time tail that is lighter than the tail of the service distribution.

In order to prove (6.3) Nunez-Queija introduced a new technique that reduces results about the tail behavior response time under $\mathcal{RV}$ job sizes to the study of the conditional response time distribution. In particular, Nunez-Queija proved that the following three conditions are enough to characterize the tail-behavior of a scheduling policy any policy P [158].

Condition 6.1 *There exists $g > 0$, $E[T(x)]^P/x \to g$ as $x \to \infty$.*

Condition 6.2 *Let $\overline{F} \in \mathcal{RV}(\alpha)$. There exists $\kappa > \alpha$ such that*

$$P(T(x)^P - E[T(x)]^P > t) \leq \frac{h(x)}{t^\kappa}$$

with $h(x) = o(x^{\kappa-\delta})$ *for some* $\delta > 0$.

Condition 6.3 $T(x)^P$ *is stochastically increasing in* $x \geq 0$.

Theorem 6.2
If $\overline{F} \in \mathcal{RV}(\alpha)$ *and Conditions 6.1-6.3 hold, then*

$$P(T(x)^P > gx) \sim P(X > x) \quad \text{as } x \to \infty$$

Using Theorem 6.2, it is very straightforward to prove (6.3). In fact Condition 6.3 is immediate and Condition 6.1 follows from the following:

$$\begin{aligned}
\lim_{x\to\infty} \frac{E[T(x)]^{SRPT}}{x} = \frac{1}{x}\int_0^x \frac{dt}{1-\rho(t)} + \frac{\lambda\widetilde{m_2}(x)}{2(1-\rho(x))^2} & \\
&= \lim_{x\to\infty} \frac{1}{x}\int_0^x \frac{dt}{1-\rho(t)} \\
&= \lim_{x\to\infty} \frac{1}{1-\rho(x)} \quad \text{(by L'Hopital's rule)} \\
&= \frac{1}{1-\rho}
\end{aligned}$$

Finally, to illustrate the verification of Condition 6.2, we will limit ourselves to $1 < \alpha < 2$ and use Chebyshev's inequality to reduce the condition to the form

$$P(T(x)^P - E[T(x)]^P > t) \leq \frac{Var[T(x)]^P}{t^\kappa}$$

Thus, we need only study the limiting behavior of $Var[T(x)]^{SRPT}$. We will start with the analysis of $Var[R(x)]^{SRPT}$. Let $\epsilon > 0$, then

$$\begin{aligned}
Var[R(x)]^{SRPT} &= \int_0^x \frac{\lambda m_2(t)}{(1-\rho(t))^3} \\
&\leq \frac{\lambda}{(1-\rho)^3}\int_0^x t^2\overline{F}(t) - \widetilde{m_2}(t) \\
&= o(x^{3-\alpha+\epsilon}) \text{as } x \to \infty
\end{aligned}$$

Similarly, it is easy to verify that

$$Var[W(x)]^{SRPT} = o(x^{3-\alpha+\epsilon})$$

Finally, we can complete the proof of (6.3) by applying Theorem 6.2.

The strength of this technique for analyzing the tail behavior of SRPT is that it does not require the availability of the transform of response time, which is difficult to work with under SRPT and many other policies. Further, the technique studies only the behavior of $T(x)$, which is easy to understand under priority based policies such as SRPT. However, the technique is limited by the fact that typically verifying Condition 6.2 involves using Markov's or Chebyshev's inequalities on higher moments of $T(x)$. Thus, in situations

where moments of $T(x)$ cannot be expressed easily (such as outside the M/GI/1 model) Theorem 6.2 is difficult to apply. To remedy this, Guillemin et. al. introduced complementary conditions in [90]. These conditions were later applied by Nuyens, Wierman, and Zwart to show that (6.3) also holds in the more general GI/GI/1 SRPT queue with $\mathcal{IR}$ service times [161].

Condition 6.4 *For some $g > 0$, $T(x)/x \to g$ a.s. as $x \to \infty$.*

Condition 6.5 *There exists a constant k such that*

$$P(T(x) > kx) = o(\overline{F}(x)).$$

Theorem 6.3
If Conditions 6.4 and 6.5 hold, $\overline{F} \in \mathcal{IR}$, and $E[X^p] < \infty$ for some $p > 1$, then

$$P(T > gx) \sim P(X > x) \text{ as } x \to \infty.$$

Notice that Condition 6.4 is a strengthened version of Condition 6.1; however, as we saw, Condition 6.2 is typically the easiest to verify, so strengthening this condition is not typically problematic. Further, the form of Condition 6.5 makes it applicable in situations where Condition 6.2 is difficult to apply (though the reverse is also true). Note that we will illustrate the use of Theorem 6.3 in our analysis of the SMART class later in the thesis.

Light-tailed service times

The analysis of the tail behavior of the response time of SRPT in the light-tailed setting has only emerged very recently. In particular, Zwart and Nuyens were the first to present an analysis [162]. The analysis they present consists of first analyzing the behavior of a 2 class priority queue and then relating the behavior of the low priority class to the behavior of the largest job size in the SRPT queue using sample path arguments. We will not include the analysis here because many of the arguments are generalized later in this section when we derive the tail behavior of the SMART classification. Instead, we will simply state the results.

First, in the case when both Assumptions 6.1 and 6.2 hold (i.e. the service distribution is light-tailed and has no mass at the right endpoint), we have that

$$\log P(T^{SRPT} > x) \sim -\gamma(B)x \quad \text{as } x \to \infty$$

Thus, in this case the tail of SRPT behaves like the tail of a busy period, which we have already seen is the heaviest possible tail among all work conserving policies. This is quite a contrast to the tail behavior of SRPT in the heavy-tailed setting.

If the service distribution has mass in its right endpoint, then the tail of SRPT can be better. For instance, in the M/D/1 queue, SRPT is equivalent to FCFS, thus the tail is equivalent to that of the workload. In other cases, when the right endpoint has mass, the tail behavior of SRPT falls in between the tail of the workload and the tail of the busy period. Define X_1 such that $P(X_1 \leq x) = P(X \leq x | X < x_U)$. Then, under Assumption 6.1, we have that if $0 < P(X = x_U) < 1$,

$$\log P(T^{SRPT} > x) \sim -\gamma(T)^{SRPT} x$$

where

$$\gamma(T)^{SRPT} = \sup_{s\in[0,\gamma(Q)]} \left[s + \mathcal{M}_A^{-1}\left(\frac{1}{\mathcal{M}_{X_1}(s)}\right)\right]$$

Thus, the tail of the response time under SRPT decays like the tail of a busy period including only the jobs smaller than the right endpoint of the service distribution (since SRPT serves jobs with the same remaining size according to FCFS order).

6.2.3 PS

The analysis of the tail behavior of PS can be approached with a wide variety of techniques. The first results for PS were obtained in the heavy-tailed setting, so we will start by surveying heavy-tailed results and then move to the light-tailed setting.

Heavy-tailed service times

The first derivation of the tail behavior of PS used the same transform approach that we described for FCFS. In particular, Zwart and Boxma [256] use the Tauberian theorem for regularly varying distributions (Theorem 2.2) in concert with the transform of T^{PS} (see Section 3.1.4) in order to obtain the asymptotics of the response time distribution of the M/GI/1 PS queue. They prove that

$$P(T^{PS} > x) \sim P(X > (1-\rho)x) \quad \text{as } x \to \infty \tag{6.4}$$

Interestingly, this indicates that the tail behavior of PS is asymptotically equivalent to that of SRPT under regularly varying service distributions, and both are optimal (up to a constant factor).

Following the initial derivation of (6.4) using a transform approach, a number of other analyses of PS in the heavy-tailed setting have emerged. In particular, Nunez-Queija [158] shows that Conditions 6.1 - 6.3 hold in the case of the M/GI/1 PS queue, which provided a far simpler derivation of (6.4). Further, Guillemin et. al. [90] verified Conditions 6.4-6.5 in the case of the GI/GI/1 PS queue, guaranteeing that (6.4) holds in this more general setting. Finally, Jelenkovic and Momcilovic [105] use a sample path approach to show that (6.4) holds even under subexponential service distributions that have the additional property that they are *square-root insensitive*, i.e.

$$P(X > x) \sim P(X > x - \sqrt{x}) \quad \text{as } x \to \infty$$

Note that this condition is always met in the case of regularly varying distributions. The necessity of this condition can be understood intuitively using the central limit theorem as follows. We already know that under PS, the rare event $\{T^{PS} > x\}$ is determined by the event $\{X > x(1-\rho)\}$. If we define $S(t)$ to be the inverse of $T(x)^{PS}$ so that $S(t) = x$ means that $T(x)^{PS} = t$, we know that $S(t)/x \to 1-\rho$ as $x \to \infty$. Further, can be shown using the central limit theorem that $S(t) = (1-\rho)x + O(\sqrt{x})$. Thus, we have that

$$P(T^{PS} > x) = P(X > S(x)) \approx P(X > (1-\rho)x + O(\sqrt{x}))$$

which illustrates the importance of having a service distribution that is square-root insensitive.

Light-tailed service times

The tail behavior of PS under light-tailed service distributions is far more complex than in the heavy-tailed setting. In fact, until very recently only asymptotics for PS in the M/M/1 setting were known. In particular, by relating the behavior of an M/M/1 PS queue to that of an M/M/1 ROS queue, Borst et. al. [39] were able to apply the results of Flatto for ROS [76] in order to prove the following 'exact asymptotics' for the response time of the M/M/1 PS queue:

$$P(T^{PS} > x) \sim cx^{-5/6}e^{-\alpha x^{1/3}}e^{-\gamma x}$$

for some constants c, α, γ.

However, in the more general GI/GI/1 model, only logarithmic asymptotics have been found. Mandjes and Zwart [135] have characterized the response time decay rate under a large class of light-tailed service distributions. In particular, the class of service distributions that satisfy the following assumption:

Assumption 6.3 *For each constant $c > 0$, we have*

$$\lim_{x\to\infty} \frac{1}{x} \log P(X > c \log x) = 0$$

Note that this assumption equivalently requires that e^X is heavy-tailed. This is satisfied by most common light-tailed service distributions, e.g. phase-type and Gamma distributions, but rules out distributions having extremely light tails such as distributions with c.d.f. of the form e^{-e^x}.

The result that Mandjes and Zwart prove about the tail of PS is that under Assumptions 6.1 and 6.3

$$\log P(T^{PS} > x) \sim \log P(B > x) \quad \text{as } x \to \infty$$

Thus, in this setting the tail behavior of PS matches that of a busy period, which is the heaviest possible tail. However, like SRPT, there are other settings where PS can have a lighter tail than a busy period. For example, Egorova, Zwart, and Boxma [72] have shown that in the M/D/1 setting the decay rate of PS falls between that of FCFS and a busy period.

6.2.4 FB

The analysis of the tail behavior of FB can be approached in much the same way as that of SRPT. In fact, the results for FB nearly parallel to those for SRPT.

Heavy-tailed service times

The analysis of the tail of response time under FB was first approached in the heavy-tailed setting. In particular, Nunez-Queija proved that Conditions 6.1 - 6.3 hold for FB in the M/GI/1 queue when the service distribution is regularly varying and $1 < \alpha < 2$ [158]. Thus, it follows from Theorem 6.2 that

$$P(T^{PS} > x) \sim P(X > (1-\rho)x) \quad \text{as } x \to \infty \tag{6.5}$$

This result was then generalized by Nuyens [159], where it was shown to hold for all $\alpha > 1$. Even more recently, Nuyens, Wierman, and Zwart [161] proved that (6.5) holds in the GI/GI/1 queue for all $\mathcal{IR}$ service

distributions. This last result was obtained using Conditions 6.4 and 6.5 in combination with Theorem 6.3 in a way that is general enough to accommodate all SMART policies in addition to just FB. Thus, we defer the proof to Section 6.3.

Light-tailed service times

In the light-tailed setting, the first analysis of FB was performed by Mandjes and Nuyens [134], where it was proven that in the M/GI/1 queue under Assumptions 6.1 and 6.2

$$\log P(T^{FB} > x) \sim \log P(B > x) \quad \text{as } x \to \infty \tag{6.6}$$

that is $\gamma(T)^{FB} = \gamma(B)$. This result was later generalized by Nuyens, Wierman, and Zwart [161], who showed that the same relation holds for the GI/GI/1 queue under Assumption 6.1.

Clearly, the behavior of FB is very similar to that of SRPT in this setting; however it is important to point out that FB performs slightly worse than SRPT with respect to the tail of response time. In particular, if the service distribution has mass in the right endpoint, the decay rate of SRPT is larger than that of the busy period, while the decay rate of FB is the same as that of the busy period. This difference results from the fact that SRPT finishes jobs of the same size in FCFS order while FB finishes all jobs in the system with the same size at the same moment.

6.2.5 LCFS

The last scheduling policy we discuss is LCFS. The tail behavior of response time under LCFS has not received much attention in the literature, but it was used as an example in a recent survey by Borst et. al. [40]. In particular, they show that the tail behavior of response time under LCFS can be derived in a number of different ways. It is easy to analyze using a transform approach, either set of sufficient conditions, or sample path analyses. Though, [40] only presents the analyses in the M/GI/1, it is clear that the analysis can be translated to the GI/GI/1 setting with little difficulty by recalling that $W^{LCFS} = B(\mathcal{E})1_{[\text{busy}]}$. Thus, we have that, in a GI/GI/1 queue with $\mathcal{IR}$ service,

$$P(T^{LCFS} > x) \sim \rho P(\mathcal{E} > (1-\rho)x) \quad \text{as } x \to \infty$$

Further, in a GI/GI/1 queue with a light-tailed service distribution it is clear that the tail behavior of LCFS will match that of a busy period, so we have that

$$\log P(T^{LCFS} > x) \sim \log P(B > x) \quad \text{as } x \to \infty$$

under any distribution that satisfies Assumption 6.1. Thus, LCFS is an example of a policy that performs badly (with respect to the response time tail) under both light-tailed and heavy-tailed service distributions.

6.3 The response time tail under scheduling classifications

Now that we have surveyed the results characterizing the tail behavior of response time under a wide range of individual policies, we can move to a discussion of how scheduling heuristics and techniques affect the tail of response time. Recall that by broadening our focus from individual, idealized policies to scheduling classifications based on heuristics and techniques we are accomplishing two important goals. First, we are deriving results that apply to the hybrid policies that are actually implemented in real systems, and second we are providing structural results that serve to organize and explain many of the results for individual scheduling policies that we have just surveyed.

In this section, we will show that the tail of response time is asymptotically equivalent under a number of different scheduling techniques and heuristics. This is interesting because it shows that the performance of a policy is dominated by the heuristic and technique it uses rather than the details of the policy itself. In particular, we will prove that all non-preemptive policies have asymptotically equivalent response time tails under heavy-tailed service distributions. Further, SMART and FOOLISH policies have asymptotically equivalent response time tails under both light-tailed and heavy-tailed service distributions.

The bulk of the section will focus on the analysis of the SMART class, since this class formalizes the heuristic of "prioritizing small jobs" and is thus the most practical of the scheduling classifications. For this class, we prove that, regardless of how small jobs are given priority, the tail behavior of response time is asymptotically equivalent to that of SRPT. This can be viewed as a theoretical proof that the adjustments made to SRPT in practical settings do not have a large effect on the response times of the resulting policies.

6.3.1 The class of non-preemptive policies

We start our analysis by considering the case of non-preemptive policies. We will first discuss the case of heavy-tailed service demands and then move to the case of light tailed service demands.

In the heavy-tailed setting, the behavior of non-preemptive policies can be understood using one simple observation: all non-preemptive policies must wait behind, at minimum, the excess of the job at the server. When the service distribution is heavy-tailed, this excess dominates the tail-behavior. Thus, all non-preemptive policies are asymptotically equivalent (ignoring constants) to FCFS with respect to the tail of response time.

Theorem 6.4

Consider an M/GI/1 queue governed by a non-preemptive policy P with $\overline{F} \in \mathcal{IR}$. Then

$$\lim_{x\to\infty} \rho P(\mathcal{E} > x) \leq \liminf_{x\to\infty} P(T^P > x) \quad \textit{and} \quad \limsup_{x\to\infty} P(T^P > x) \leq \lim_{x\to\infty} \frac{\rho}{1-\rho} P(\mathcal{E} > (1-\rho)x)$$

This theorem generalizes the results of Anantharam in [14], where the tail behavior of response time under non-preemptive policies is first discussed. Interestingly, many non preemptive policies have tail behavior similar in form to the bounds above. For instance

$$\begin{aligned} P(T^{FCFS} > x) &\sim \frac{\rho}{1-\rho} P(\mathcal{E} > x), \quad x \to \infty \\ P(T^{LCFS} > x) &\sim \rho P(\mathcal{E} > (1-\rho)x), \quad x \to \infty \end{aligned}$$

We are now ready to prove Theorem 6.4

Proof of Theorem 6.4. The lower bound in the theorem is immediate from the observation that if a tagged job arrives and finds the server busy, then the tagged job must wait at minimum job the job at the server to complete, which is an excess. Thus, $P(T^P > x) \geq \rho P(\mathcal{E} > x)$.

To prove the upper bound, we use a similar tagged job argument. First, notice that $P(T^P > x) \leq (1-\rho)P(X > x) + \rho P(B(Q) > x)$ where Q is the steady state workload. Noting that the second term dominates the asymptotics, and can focus our attention on $B(Q)$. Recalling that $P(B(Q) > x) \sim P(Q > (1-\rho)x)$ and $P(Q > x) \sim \frac{1}{1-\rho} P(\mathcal{E} > x)$ completes the proof.
□

Interestingly, though non-preemptive policies have asymptotically equivalent tails in the heavy-tailed setting, they can have very different tails in the light tailed setting. In particular,

$$\begin{aligned} \lim_{x\to\infty} \log P(T^{FCFS} > x) &= \lim_{x\to\infty} \log P(Q > x) \\ \lim_{x\to\infty} \log P(T^{LCFS} > x) &= \lim_{x\to\infty} \log P(B > x) \end{aligned}$$

Where Q is the steady-state workload and B is a standard busy period. As we have already seen, these are the largest and smallest decay rates possible in this setting.

6.3.2 The SMART class

We now move to the analysis of the tail behavior of SMART scheduling policies. We will show that all SMART policies are asymptotically equivalent to SRPT in both the heavy-tailed and light-tailed settings. These results were first proven in Nuyens, Wierman, and Zwart [161]. The technique that we apply to analyze the SMART class turns out to be quite general. It turns out that it can easily be generalized to analyze a variety of other priority-based policies. In order to illustrate this fact, we will use the FB policy as a running example throughout this section.

Heavy-tailed service times

In this section we derive the tail behavior of response time for SMART policies under $\mathcal{IR}$ service distributions. The main result that we prove is the following.

Theorem 6.5
In the GI/GI/1 queue with $\mathsf{P} \in$ SMART, *if* $\overline{F} \in \mathcal{IR}$, *then*

$$P(T^{\mathsf{P}} > x) \sim P(X > (1-\rho)x), \text{ as } x \to \infty. \tag{6.7}$$

To prove the above theorem, we will use Conditions 6.4 and 6.5 in combination with Theorem 6.3.

Note that we have prove that all SMART policies satisfy Condition 6.4 in Section 7.2, thus we will only prove that SMART policies satisfy condition 6.5 in this section.

Before proving that Condition 6.5 holds for SMART policies, we prove an auxiliary result. A similar result has been shown before for the workload in the $M/G/1$ queue [105].

Lemma 6.6
Let X_i be i.i.d. random variables with $E[(X_i^+)^p] < \infty$ for some $p > 1$. Let $S_n(y) = \sum_{i=1}^n (X_i \wedge y)$. Define $M(y) = \sup_n S_n(y)$. For every $\beta > 0$, there exists a $k > 0$ such that $P(M(x) > kx) = o(x^{-\beta})$.

A key ingredient to the proof of this auxiliary result is the following lemma, which is due to Resnick and Samorodnitsky [186].

Lemma 6.7
Let $S_n = X_1 + \cdots + X_n$ be a random walk with i.i.d. step sizes such that $E[X_1] < 0$ and $E[(X_1^+)^p] < \infty$ for some $p > 1$. Then, for any $\alpha < \infty$, there exist $c, k^* > 0$ such that for any n, x and $k > k^*$,

$$P(S_n > kx \mid X_i < x, i \le n) \le cx^{-\alpha}.$$

Using Lemma 6.7, we will now prove Lemma 6.6.

Proof. Let $\beta > 0$. For fixed $y \ge 1$, we write the standard geometric random sum decomposition

$$M(y) \stackrel{d}{=} \sum_{i=1}^{N(y)} H_i(y),$$

with $N(y)$ the number of ladder heights, and $H_i(y)$ the ith overshoot; for details see e.g. Chapter VIII of []. By a sample-path comparison, it follows that

$$M(y) \stackrel{st}{\le} \sum_{i=1}^{N(\infty)} [H_i(\infty) \wedge y].$$

Writing $H_i = H_i(\infty)$, we have for any $k, \gamma > 0$,

$$P(M(x) > kx) \le P(N(\infty) > \lfloor x^\gamma \rfloor) + P\Big(\sum_{i=1}^{\lfloor x^\gamma \rfloor} (H_i \wedge x) > kx \Big), \tag{6.8}$$

where $\lfloor z \rfloor$ is the largest integer smaller than or equal to z. Since the number of overshoots is geometrically distributed, the first term in (6.8) behaves like $\exp(-c\lfloor x^\gamma \rfloor)$ for some $c > 0$. Since this decays faster than any power tail for any $\gamma > 0$, it suffices to consider the second term.

Let $0 < q < \min\{1, p-1\}$. Since the tail of H_i is one degree heavier than that of the X_k (see Theorem 2.1 in Chapter VIII of Asmussen (2003)), we have $EH_i^q < EH_i^{p-1} < \infty$. Hence, $H_i^q - 2E[H_i^q]$ satisfies

the assumption of Lemma 6.7. Take $\gamma \in (0,q)$. Since y^q is a concave function in y, we have

$$\begin{aligned} P\Big(\sum_{i=1}^{\lfloor x^\gamma \rfloor}(H_i \wedge x) > kx\Big) &= P\Big(\big[\sum_{i=1}^{\lfloor x^\gamma \rfloor}(H_i \wedge x)\big]^q > (kx)^q\Big) \\ &\leq P\Big(\sum_{i=1}^{\lfloor x^\gamma \rfloor}(H_i \wedge x)^q > (kx)^q\Big) \\ &= P\Big(\sum_{i=1}^{\lfloor x^\gamma \rfloor}((H_i \wedge x)^q - 2E[H_i^q]) > (kx)^q - 2\lfloor x^\gamma \rfloor E[H_i^q]\Big). \end{aligned}$$

To apply Lemma 6.7, we need conditioned, and not truncated random variables. Choose an integer $l > \beta/(q-\gamma)$. Considering the event that at least l of the H_i are larger than x, and its complement, we find

$$\begin{aligned} &P\Big(\sum_{i=1}^{\lfloor x^\gamma \rfloor}((H_i \wedge x)^q - 2E[H_i^q]) > (kx)^q - 2\lfloor x^\gamma \rfloor E[H_i^q]\Big) \\ &\leq \binom{\lfloor x^\gamma \rfloor}{l} P(H_i > x)^l + P\Big(\sum_{i=1}^{\lfloor x^\gamma \rfloor}(H_i^q - 2E[H_i^q]) > (kx)^q - 2\lfloor x^\gamma \rfloor E[H_i^q] \,\Big|\, \#\{i : H_i > x\} < l\Big) \\ &\leq x^{\gamma l} P(H_i > x)^l + P\Big(\sum_{i=1}^{\lfloor x^\gamma \rfloor - l}(H_i^q - 2E[H_i^q]) > (kx)^q - lx^q - 2\lfloor x^\gamma \rfloor E[H_i^q] \,\Big|\, H_i \leq x\Big). \end{aligned} \tag{6.9}$$

We complete the proof by showing that both terms in (6.9) are $o(x^{-\beta})$.

Since $E[H_i^q] < \infty$, we know that $P(H_i > x) = o(x^{-q})$. Hence, since $0 < \gamma < q$, and $l > \beta/(q-\gamma)$, we have $x^{\gamma l} P(H_i > x)^l = o(x^{\gamma l} x^{-ql}) = o(x^{-\beta})$. Let $\overline{k} > 0$. Since $q > \gamma$, for k large enough, the second term in (6.9) is smaller than

$$P\Big(\sum_{i=1}^{\lfloor x^\gamma \rfloor - l}(H_i^q - 2E[H_i^q]) > \overline{k}\big(x^q - 2E[H_i^q]\big) \,\Big|\, H_i^q - 2E[H_i^q] \leq x^q - 2E[H_i^q]\Big). \tag{6.10}$$

Applying Lemma 6.7 with a suitable choice of $\overline{k}$, there exist $c > 0$ and $\eta > \beta/q$ such that (6.10) is smaller than

$$c\big(x^q - 2E[H_i^q]\big)^{-\eta} \sim c\big(x^q\big)^{-\eta} = o(x^{-\beta}), \qquad x \to \infty.$$

This completes the proof.
□

Consider now a $GI/GI/1$ queue with the same interarrival-time distribution as before, but with generic service time $X \wedge x$. Set $A_x(t) = \sum_{i=1}^{K(t)}(X_i \wedge x)$, with $K(t)$ the number of arrivals in $(0,t]$, so $A_x(t)$ is the work entering the queue in the time interval $(0,t]$. Furthermore, at the beginning of each busy period an initial setup time x is added. Let $\widetilde{B}_x^{*s}$ be the residual busy period after the arrival of a customer of size x.

Then $\widetilde{B}_x^{*s}$ can be represented as follows:

$$\widetilde{B}_x^{*s} = \inf\{t : x + \widetilde{Q}_x^s + A_x(t) - t = 0\},$$

with $\widetilde{Q}_x^s$ the steady-state workload upon customer arrivals in this queue, including the effect of the initial set-up.

Furthermore, let $D_x^{\mathsf{P}}(t)$ be the stochastic processes of work under policy P that would have priority over an arriving job of size x at time t.

Lemma 6.8
For $\mathsf{P} = \mathsf{FB}$ *and all* $\mathsf{P} \in \mathsf{SMART}$, *we have*

$$T(x)^{\mathsf{P}} \leq_{st} \widetilde{B}_x^{*s}.$$

Note that we design this so as to include FB to illustrate that the same proof technique we are using for SMART policies can be easily adjusted to handle many other priority based policies.

Proof. The bound holds for FB, since the residual busy period bounds $T(x)^{\mathsf{FB}}$ if the setup time were not included.

To see that the residual busy period also bounds $T(x)^P$ for $\mathsf{P} \in \mathsf{SMART}$, note that the process $D_x^{\mathsf{P}}(t)$ consists of two types of busy periods: (i) busy periods started by a job of original size $> x$ that now has remaining size $\leq x$ and (ii) busy periods started by a job of original size $\leq x$. In both cases, the Bias Property prevents any job with remaining size $> x$ from receiving service during the busy period; thus only new arrivals of size $\leq x$ can contribute once the busy period is started. Since the initial job under $\mathsf{P} \in \mathsf{SMART}$ is necessarily smaller than the setup x of $\widetilde{B}_x^{*s}$, and the arrivals during the busy period are stochastically larger in $\widetilde{B}_x^{*s}$, the residual length of both of these busy periods is stochastically smaller than $\widetilde{B}_x^{*s}$.
□

The following lemma implies that Condition 6.5 holds for SMART policies.

Lemma 6.9
For every $\beta > 0$, *there exists a constant* k *such that*

$$P(T(x)^{\mathsf{P}} > kx) = o(x^{-\beta}), \qquad x \to \infty \tag{6.11}$$

for all $\mathsf{P} \in \mathsf{SMART}$. *As a consequence, Condition 6.5 holds for* $\mathsf{P} \in \mathsf{SMART}$.

Proof. Let $\mathsf{P} \in \mathsf{SMART}$. We will bound $T(x)^{\mathsf{P}}$ using the residual busy period $\widetilde{B}_x^{*s}$ as per Lemma 6.8. Furthermore, define

$$U_x^c = \sup_{t>0}[A_x(t) - ct + x] = x + \sup_{t>0}[A_x(t) - ct]. \tag{6.12}$$

Then $U_x^1 = \widetilde{Q}_x^s$. For $(1-\rho)/2 < \delta < 1/2$ and $k > 1/\delta$, we have by Lemma 6.8,

$$\begin{aligned} P(T(x)^{\mathsf{P}} > kx) &\leq P(\widetilde{B}_x^{*s} > kx) \\ &\leq P(x + U_x^1 + A_x(kx) - kx > 0) \\ &\leq P(U_x^1 + A_x(kx) - (1-2\delta)kx + x > \delta kx) \\ &\leq P(U_x^1 > \delta kx/2) + P(A_x(kx) - (1-2\delta)kx + x > \delta kx/2) \\ &\leq P(U_x^{1-2\delta} > \delta kx/2) + P(\sup_{t>0}[A_x(t) - (1-2\delta)t + x] > \delta kx/2) \\ &= 2P(U_x^{1-2\delta} > \delta kx/2). \end{aligned}$$

By taking $\overline{k} = k\delta/2$, and $c = 1-2\delta$, it suffices to show that there exists a $\overline{k} > 1$ such that

$$P(U_x^c > \overline{k}x) = P(U_x^c - x > (\overline{k}-1)x) = o(x^{-\beta}). \tag{6.13}$$

We complete the proof by viewing $U_x^c - x$ in terms of a random walk. Since the supremum in (6.12) is attained at arrival instants, we may write

$$U_x^c - x = \sup_n \sum_{i=1}^n (X_i \wedge x) - cA_i \leq \sup_n \sum_{i=1}^n [(X_i - cA_i) \wedge x],$$

where A_i is the time between the $(i-1)$st arrival and the ith arrival. Since $E[X_i - (1-2\delta)A_i] < E[X_i - \rho A_i] = 0$ and $E[((X_i - (1-2\delta)A_i)^+)^p] \leq E[X_i^p] < \infty$, we may apply Lemma 6.6, and (6.13) follows.

To show that $\mathsf{P} \in \mathsf{SMART}$ obey Condition 6.5, note that since $\overline{F} \in \mathcal{IR}$, there exists a $\beta > 0$ such that $x^{-\beta} = o(P(X > x))$. Take this β and choose k as in (6.11). Condition 6.5 now follows.
□

Light-tailed service times

We will now derive the tail behavior of response time for SMART policies under service distributions for which Assumptions 6.1 and 6.2 hold. We prove the following two main theorems.

Theorem 6.10
In the GI/GI/1 queue with $\mathsf{P} \in \mathsf{SMART}$, *if Assumption 6.1 holds, then*

$$\gamma(B) \leq \gamma(T^{\mathsf{P}}) \leq \gamma(T^{\mathsf{SRPT}}).$$

Furthermore, if both Assumptions 6.1 and 6.2 hold, then $\gamma(T^{\mathsf{P}}) = \gamma(T^{\mathsf{FB}}) = \gamma(B)$. *That is,*

$$\log P(T^{\mathsf{P}} > x) \sim \log P(B > x), \text{ as } x \to \infty. \tag{6.14}$$

Theorem 6.11
Suppose $\mathsf{P} \in \mathsf{SMART}$. *Let* y *be such that* $P(X = y) = 0$. *Then*

$$\gamma(T^{\mathsf{P}}(y)) = \gamma(B_y). \tag{6.15}$$

Theorem 6.10 follows from Lemmas 6.14, 6.15, and 6.16; while Theorem 6.11 follows from Lemma 6.17.

We start by upper bounding the tail of T^{P} under all work-conserving disciplines P using the observation that $T^{\mathsf{P}} \leq B^*$, where $B^* \stackrel{\mathrm{d}}{=} B(Q+X)$ is the length of the busy period starting with the amount of work $Q+X$, Q is the steady-state amount of work in the system (upon customer arrivals), and X is a generic service time. Furthermore, $B(\cdot)$, Q and X are independent, i.e., B^* is a residual busy period.

It can be shown that the decay rates of B^* and B coincide. Moreover, we have the following upper bound:

Lemma 6.12
For all work-conserving disciplines P,

$$\limsup_{x\to\infty} \frac{1}{x} \log P(T^{\mathsf{P}} > x) \leq -\gamma(B^*) = -\gamma(B). \tag{6.16}$$

Proof. The first inequality follows from the above observation that $T^{\mathsf{P}} \leq B^*$. The equality $\gamma(B^*) = \gamma(B)$ is trivial in the $M/G/1$ queue, but for the $GI/GI/1$ queue we need additional arguments. Let Q be the steady-state virtual waiting time in the $GI/GI/1$ queue. By Lemma 3.2 in [], B and $B(Q)$ have the same decay rate. However, we are interested in the decay rate of $B(Q+X)$. In the $M/G/1$ case, we could apply PASTA. In the general case, we note that $T \stackrel{d}{=} (Q+X-A^*)^+$, with A^* a residual interarrival time. Therefore, T is stochastically smaller than $Q+X$. Consequently, $\gamma(B^*) = \gamma(B(Q+X)) \leq \gamma(B(T)) = \gamma(B)$. To prove the upper bound, let $A(t)$ be the total amount of work fed into the system between time 0 and t. Using the Chernoff bound, we find

$$P(B(Q+X) > x) \leq P(A(x) - x + Q + X > 0) \leq E[e^{sQ}]E[e^{sX}]E[e^{s(A(x)-x)}].$$

The proof is now completed by minimizing the last factor over s, and showing that for the optimizing argument s^*, we have $E[e^{s^*Q}] < \infty$ and $E[e^{s^*X}] < \infty$. Since this is exactly what is done in Proposition 3.1 of [162], refer to that work for the remaining supporting arguments.
□

The following lemma, which is Proposition 2.2 in [162], will play a key role in our arguments.

Lemma 6.13
For a GI/GI/1 queue under Assumption A, $\gamma(B_x) \downarrow \gamma(B)$ and $\gamma(\widetilde{B_x}) \downarrow \gamma(B)$ *as* $x \uparrow x_F$.

After these two preliminary lemmas, we are now ready to prove Theorems 6.10 and 6.11. We start by analyzing the behavior of SMART . We start by doing the analysis in the simplest case: both Assumptions 6.1 and 6.2 hold, and the service distribution is unbounded.

Lemma 6.14
In the GI/GI/1 queue with $\mathsf{P} \in$ SMART, *if Assumptions 6.1 and 6.2 hold, and* $x_F = \infty$, *then* $\gamma(T^{\mathsf{P}}) = \gamma(B)$. *That is,*

$$\log P(T^{\mathsf{P}} > x) \sim \log P(B > x), \text{ as } x \to \infty.$$

Proof. Let A_1 be the first arrival after that of a tagged customer with size X_0. Let a be such that $P(A < a) > 0$ and $y < x_F - a$. Then for all $P \in \mathsf{SMART}$,

$$\begin{aligned} P(T^{\mathsf{P}} \geq x) &\geq P(T(X_0)^{\mathsf{P}} > x, A_1 < a, X_0 > y + a) \\ &= P(A_1 < a, X_0 > y + a) P(T(X_0)^{\mathsf{P}} > x | A_1 < a, X_0 > y + a). \end{aligned}$$

Conditional on $X_0 > y + a$ and $A_1 < a$, the tagged job has remaining service time larger than y when the new job arrives. The Bias Property implies that this new job has higher priority than the tagged job if its service times is smaller than y. Furthermore, all jobs with service time smaller than y that arrive while the new job is in the system will also have higher priority than the tagged job. Thus, conditional on $X_0 > y + a$ and $A_1 < a$, we have $T(X_0)^{\mathsf{P}} \geq_{st} B_y$. Hence,

$$P(T^{\mathsf{P}} \geq x) \geq P(A_1 < a, X_0 > y + a) P(B_y > x).$$

Since $P(A_1 < a, X_0 > y + a) > 0$, the existence of $\gamma(B_y)$ implies that

$$\liminf_{x \to \infty} \frac{1}{x} \log P(T^{\mathsf{P}} \geq x) \geq \liminf_{x \to \infty} \frac{1}{x} P(B_y > x) = -\gamma(B_y). \tag{6.17}$$

To prove the lemma, it suffices to show that the liminf result corresponding to (6.16) holds. Letting y go to ∞ in (6.17), and applying Lemma 6.13, yields

$$\liminf_{x \to \infty} \frac{1}{x} \log P(T^{\mathsf{P}} \geq x) \geq -\gamma(B).$$

This completes the proof.
□

We now relax the assumption that the service distribution is unbounded. This relaxation results in the need for a more involved argument.

Lemma 6.15
In the GI/GI/1 queue with $\mathsf{P} \in \mathsf{SMART}$*, if Assumptions 6.1 and 6.2 hold, and* $x_F < \infty$*, then* $\gamma(T^{\mathsf{P}}) = \gamma(B)$*.*

Proof. If $P(A < a) > 0$ for all $a > 0$, then the result follows from (6.17) and Lemma 6.13, as in the proof of Lemma 6.14. However, this may not be the case, so we need a different construction.

By definition of x_F, there exists a decreasing sequence $\{\varepsilon_n\}$ such that $\varepsilon_n \to 0$ as $n \to \infty$, and $P(x_F - \varepsilon_n < X < x_F - \varepsilon_n/2) > 0$ for all n. Since $P(X > A) > 0$, we can assume that ε_1 is such that $P(A < x_F - 2\varepsilon_1) > 0$. Let Z_n be the event that the last $\lfloor x_F/\varepsilon_n \rfloor$ customers that arrived before the tagged customer had a service time in the interval $[x_F - \varepsilon_n, x_F - \varepsilon_n/2]$, and that the last $\lfloor x_F/\varepsilon_n \rfloor$ inter-arrival times were smaller than $x_F - 2\varepsilon_n$. By definition of ε_n, we have $P(Z_n) > 0$ for all n.

Furthermore, the Bias Property guarantees that, on the event Z_n, there is a customer with remaining service time larger than $k\varepsilon_n$ after the kth of the inter-arrival times. Hence, at the arrival of the tagged customer (after $k = \lfloor x_F/\varepsilon_n \rfloor$ arrivals), there is a customer in the system with remaining service time in the

interval $[x_F - \varepsilon_n, x_F - \varepsilon_n/2]$. If the tagged customer has service time $X_0 > x_F - \varepsilon_n/2$, his sojourn time satisfies $T^{\mathsf{P}} \geq B_{x_F-\varepsilon_n}$. Consequently, for all $n \in \mathbb{N}$,

$$P(T^{\mathsf{P}} > x) \geq P(Z_n)P(X_0 > x_F - \varepsilon_n/2)P(B_{x_F-\varepsilon_n} > x).$$

Thus, for $\mathsf{P} \in \mathsf{SMART}$, we have

$$\liminf_{x\to\infty} \frac{1}{x} \log P(T^{\mathsf{P}} > x) \geq -\gamma(B_{x_F-\varepsilon_n}).$$

As $n \to \infty$, and hence $\varepsilon_n \downarrow 0$, Lemma 6.13 implies that $\gamma(B_{x_F-\varepsilon_n}) \to \gamma(B)$. Using Lemma 6.12 completes the proof.
□

Finally, we relax Assumption 6.2. In contrast to FB, the sojourn-time tail of SMART policies can improve when there is mass in the endpoint of the service distribution. This is not surprising since many SMART policies, e.g., SRPT, are equivalent to FCFS in the GI/D/1 queue. However, SMART also includes policies where, like FB, jobs of the same size are not served in FCFS order. Thus, the SMART policies have a range of possible sojourn-time tails in this setting.

Lemma 6.16
In the GI/GI/1 queue, under Assumption 6.1, for all $\mathsf{P} \in \mathsf{SMART}$,

$$\gamma(B) \leq \gamma(T^{\mathsf{P}}) \leq \gamma(T^{\mathsf{SRPT}}). \tag{6.18}$$

Proof. By Theorem 6.10, we only need to deal with the case that Assumption 6.2 does not hold, i.e., $P(X = x_F) > 0$. The first inequality follows from Lemma 6.12. For the second inequality, note that $P(T^{\mathsf{P}} > x) \geq P(T(x_F)^{\mathsf{P}} > x)P(X = x_F)$. Thus, since $P(X = x_F) > 0$, $\gamma(T^{\mathsf{P}}) \leq \gamma(T(x_F)^{\mathsf{P}})$. Furthermore, for all $\mathsf{P} \in \mathsf{SMART}$, $T(x_F)^{\mathsf{P}} \geq_{st} W(x_F)^{\mathsf{P}} \geq_{st} W_2(x_F)$, where $W_2(x_F)$ is the waiting time of a low priority job in a 2-class priority queue where the high-priority class includes all jobs smaller than x_F. To complete the proof, we apply Theorems 3.1 and 4.2 of [162], which state that $\gamma(W_2(x_F)) = \gamma(T^{\mathsf{SRPT}})$.
□

We end this section with the analysis of the conditional sojourn time under SMART policies.

Lemma 6.17
In the GI/GI/1 queue with $\mathsf{P} \in \mathsf{SMART}$, *if* $P(X = y) = 0$, *then*

$$\gamma(T(y)^{\mathsf{P}}) = \gamma(B_y). \tag{6.19}$$

Proof. For the lower bound, we remark that $T^{\mathsf{P}}(y) \geq_{st} B_y^*$ for all $\mathsf{P} \in \mathsf{SMART}$. By Lemma 6.12, this residual busy period has decay rate $\gamma(B_y)$.

For the upper bound, we use the fact that, at any point in time, at most one customer with original service time larger than y has remaining service time smaller than y. Denoting by Q_y the stationary workload, upon

arrival instants, made up of customers with service time smaller than y, we can bound

$$T^{\mathsf{P}}(y) \leq_{st} B_y(Q_y + y + y).$$

Denoting the amount of work brought by customers (with size smaller than y) entering the queue in the interval $[0, x]$ by $A_y(x)$, the Chernov bound yields that for all $s \geq 0$,

$$\begin{aligned} P(T^{\mathsf{P}}(y) > x) &\leq P(B_y(Q_y + 2y) > x) \leq P(Q_y + 2y + A_y(x) > x) \\ &= P(\exp(s(Q_y + 2y + A_y(x)) > e^{sx}) \leq e^{-sx} e^{2sy} E e^{sQ_y} E e^{sA_y(x)}. \end{aligned}$$

Hence, for all $s < \gamma(Q_y)$, we have

$$\limsup_{x\to\infty} \frac{1}{x} \log P(T^{\mathsf{P}}(y) > x) \leq -s + \limsup_{x\to\infty} \frac{1}{x} \log E e^{sA_y(x)} = -s - \Phi_A^{\leftarrow}\left(\frac{1}{\Phi_{XI(X<y)}(s)}\right),$$

where the equality follows from Lemma 2.1 in [135]. Taking the infimum over all $s \in [0, \gamma(Q_y))$ yields

$$\limsup_{x\to\infty} \frac{1}{x} \log P(T^{\mathsf{P}}(y) > x) \leq -\sup_{0\leq s<\gamma(Q_y)} \left[s + \Phi_A^{\leftarrow}\left(\frac{1}{\Phi_{XI(X<y)}(s)}\right) \right] = -\gamma(B_y^*),$$

where the equality follows from equation (5.1) in [162]. Noting that B_y and B_y^* have the same decay rate yields the desired upper bound, and completes the proof.
□

6.3.3 The FOOLISH class

We now move to the class of FOOLISH policies. The tail behavior of FOOLISH policies provides a huge contrast to the results for the class of SMART policies we just discussed. Not surprisingly, we will show that the class of FOOLISH policies has the worst possible tail behavior under both light-tailed and heavy-tailed service distributions.

Heavy-tailed service times

The main result we will prove about FOOLISH policies under heavy-tailed service distributions is the following.

Theorem 6.18
In an M/GI/1 queue governed by $P \in$ FOOLISH with $\overline{F} \in \mathcal{IR}$,

$$\lim_{x\to\infty} \frac{\rho}{1-\rho} P(\mathcal{E} > x) \leq \liminf_{x\to\infty} P(T^P > x) \quad \textit{and} \quad \limsup_{x\to\infty} P(T^P > x) \leq \lim_{x\to\infty} \frac{\rho}{1-\rho} P(\mathcal{E} > (1-\rho)x)$$

Though we prove this result in the M/GI/1 setting, it can be extended to the GI/GI/1 setting using arguments parallel to those used in the analysis of SMART.

Proof. We will start by proving the lower bound on $P(T^P > x)$ using a sample path argument. Recall that $T^{PLJF} \leq_{st} T^P$, thus we can use PLJF to lower bound the tail of P. We will show that the critical event leading to the large delay of a tagged job X_0 that arrives at time $t = 0$ is the arrival of a big job X_b at some point $t = -y$ before the arrival of X_0. Formally, define $U^c = \sup_{t\geq 0}\{ct - A(0,t)\}$ where $A(0,t)$ is the amount of work arriving in a time interval $(0,t)$. For any $\epsilon > 0$, $\delta > 0$,

$$\begin{aligned}
P(T^P > x) &\geq P(T^{PLJF} > x) \\
&\geq \int_{y=0}^{\infty} \lambda P(A(0,y) + X_b - y > x) P(X_b > X_0) dy \\
&\geq \lambda \int_{y=0}^{\infty} P(A(0,y) - y(\rho - \delta) \geq -\epsilon x) P(X_b > x(1+\epsilon) + y(1-\rho+\delta)) P(X_b > X_0) dy \\
&\geq \lambda P(\inf_{u\geq 0}(A(0,y) - u(\rho-\delta))) \int_{z=x(1+\epsilon)}^{\infty} P(X_b > z) P(X_b > X_0) \frac{dz}{1-\rho+\delta} \\
&\geq P(U^{\rho-\delta} \leq \epsilon x) \frac{\rho}{1-\rho+\delta} P(\mathcal{E} > x(1+\epsilon)) P(X_0 \leq x(1+\epsilon))
\end{aligned}$$

Note that $P(U^{\rho-\delta} \leq \epsilon x) \to 1$ as $x \to \infty$ because of the law of large numbers. Further, $P(X_0 \leq x(1+\epsilon)) \to 1$ as $x \to \infty$. Thus, the proof of the lower bound is complete.

To prove the upper bound, we recall that $T^P \leq_{st} T^{LRPT}$. Thus, we need only analyze the tail behavior of LRPT. But, in the M/GI/1 setting,

$$T^{LRPT} \stackrel{\mathrm{d}}{=} B(X+Q)$$

where Q is the steady state work in system. Recalling that $P(B(X+Q) > x) \sim P(X+Q > (1-\rho)x)$, $T^{FCFS} \stackrel{\mathrm{d}}{=} X + Q$, and $P(T^{FCFS} > x) \sim \frac{\rho}{1-\rho} P(\mathcal{E} > x)$ completes the proof of the upper bound.
□

Light-tailed service times

We now characterize the response time tail of FOOLISH policies under light-tailed service distributions.

Theorem 6.19

Consider a GI/GI/1 queue governed by $P \in$ FOOLISH under Assumption 6.1 and the assumption that $P(X = x_L) = 0$ where $x_L = \inf_a(F(a) > 0)$. Then, $\gamma(T^P) = \gamma(B)$. That is,

$$\log P(T^P > x) \sim \log P(B > x), \text{ as } x \to \infty.$$

The assumption that $P(X = x_L) = 0$ acts similarly to Assumption 6.2 in the case of SMART policies. Without this assumption, FOOLISH policies are not asymptotically equivalent. For example, PLJF and LRPT differ because LRPT will complete all jobs in the system of size x_L at the same time and PLJF will complete jobs of size x_L in FCFS order, which leads to a smaller decay rate when jobs of size x_L have positive probability mass. For a simple example of this, think of the case of a GI/D/1 queue.

Proof. We have already seen that $\gamma(B)$ is the smallest possible decay rate for work conserving policies, thus we need only prove an upper bound on the decay rate of T^P (i.e. a lower bound on the tail of T^P).

To accomplish this we will construct a critical event that leads to a large delay under all PLJF, which is a stochastic lower bound for all $P \in$ SMART.

To construct a critical event, consider the arrival of a big job X_b, followed by the arrival of a small job X_s after an interarrival time A_0. At the time when the small job arrives, the remaining size of the large job, r, starts a busy period of jobs larger than X_s, $B_{>X_s}(r)$, which are guaranteed to complete before X_b. Let the lower bound of the service distribution be x_L and the upper bound be x_U. For any $\epsilon > 0$, $b > a > 0$,

$$P(T^P > x) \geq P(X_b > b + a, X_s < x_L + \epsilon, A_0 < a)P(B_{>x_L+\epsilon}(b) > x)$$

Finally, letting $\epsilon \to 0$ and recalling that a conditional busy period has the same decay rate as a standard busy period completes the proof.
□

6.4 Concluding remarks

In this chapter, we have moved beyond mean response time and provided a number of results characterizing the distribution of response time both under individual policies and scheduling classifications. Such a study is essential to the applicability of scheduling in practice because users in computer systems demand both response times that are fast on average and response times that are predictable. Further, distributional results about response time are fundamental to QoS, admission control, and capacity planning applications.

But, unfortunately, studying the distribution of response time under scheduling policies is known to be a difficult problem, and thus we cannot hope to attain exact results. Instead, our focus has been on studying the tail behavior of response time in the large buffer large deviations regime. That is, we have characterized the behavior of $P(T > x)$ as $x \to \infty$. Such a regime is of practical importance because it provides bounds on the likelihood of large delays, which are exactly what is needed for QoS and capacity planning applications.

In this chapter, we provided a thorough summary of recent results studying the tail behavior of response time under a wide variety of scheduling policies. In addition, we added to the literature by deriving the tail behavior of response time under SRPT and FB in a heavy-tailed GI/GI/1 queue for the first time, and by deriving the tail behavior of response time in light-tailed GI/GI/1 FB queues for the first time. Further, our main focus in this chapter was on characterizing the response time tail of scheduling classifications, and we attain the first results for the SMART and FOOLISH classifications. These results are essential for the application of scheduling in real-world settings because the classifications include not only idealized policies like SRPT, but also the variations of these policies that are actually implemented in practice. Not only are our results about classifications of practical importance, they also add structure to the space of scheduling policies. For example, all policies in the SMART class have a response time tail that is asymptotically optimal in the heavy-tailed setting but as heavy as possible in the light-tailed setting. This is a behavior that has been observed under many common individual policies, but by showing that the behavior occurs under all policies that "prioritize small jobs" we have provided a formalization of the reason this behavior emerges.

To conclude this chapter, it is important to point out that the large buffer scaling that we used in this chapter is only one scaling that can be used to study the response time distribution. It is the most commonly

used scaling for studying scheduling policies, but recently, the *many sources* large deviations regime has also emerged as a useful tool for analyzing scheduling policies. In the many sources framework, the number of arriving flows, the buffer size, and the service capacity are scaled up proportionally yielding a system with a huge number of arriving flows and huge service capacity. In this asymptotic setting, it is then possible to directly study the delay distribution. Practically speaking, this framework is motivated by applications such as high traffic web servers and routers that have enormous available bandwidth and thousands of simultaneous flows. The analysis of scheduling (beyond FCFS) in the many sources framework is much less mature than the large buffer framework, and has only recently yielded results. In particular, until recently, only results for a handful of individual policies have been able to be attained [42, 64, 209, 123, 249]. But, very recently, we introduced a novel technique for the analysis of both individual scheduling policies and classes of scheduling policies in the many sources regime [248]. This technique, that we refer to as the *two dimensional queueing framework*, adds tie-break rules to policies in ways that do not alter the asymptotic performance of the policies, but greatly simplify their analysis. The strength of this novel framework is that it enables: *(i)* the study of policies that depend on the job state (age and/or remaining size), as opposed to only the queue length; and *(ii)* the study of a *class of policies*, as opposed to only the analysis of individual policies. In [248], we illustrate the generality of this technique by analyzing both FB and the SMART classification in the many sources regime.

Note that the many sources and large buffer regimes provide very different views of the response time distribution. Each regime captures the impact of some practical factors that the other regime does not. For instance, the large buffer scaling provides results that contrast the behavior of response time under heavy-tailed service distributions with infinite support and light-tailed service distributions that may have finite support and may be discrete. In contrast, the many sources regime requires the service distribution to be discrete and have finite support, thus it can only provide results for a small set of light-tailed service distributions. On the other hand, the many sources framework captures the impact of statistical multiplexing between arrival flows and allows the estimation of $P(T(k) > x)$ for all x, neither of which can be captured in the large buffer setting.

As a result of these differences, the two regimes provide complementary descriptions of the "most likely way" (the *critical event* in large deviations parlance) in which large response times occur. Under the large buffer setting, especially when the service distribution is heavy-tailed, it is common that the critical event is the arrival of a handful of very large jobs. In contrast, under the many sources setting, the critical event is the arrival of a large burst of jobs (i.e. a moderate number of arrivals per flow from a large number of flows). Practically, each of these critical events can be important depending on the system that is considered. For instance, if one considers a low-to-moderate traffic web server or router, the arrival of a handful of very large requests is likely to choke the system, and such an event is more likely than a large burst of arrivals. Thus, in this setting the large buffer regime provides useful results. However, in a high traffic web server or router, one with enormous bandwidth (typical large web-servers today handle multi-Gbps traffic) that is accessed by a large number of flows, the arrival of handful of large requests from any single flow is unlikely to choke the system; rather, the critical event is a burst of moderate sized (compared to the scale of the server capacity) arrivals from a large number of flows, which matches the results from the many sources framework.

CHAPTER 7

Fairness

Traditionally, and until this point in this thesis, the performance of scheduling policies has been measured using the mean response time and the tail of response time. Under these measures, we have seen that policies that give priority to small job sizes at the expense of larger job sizes perform quite well. For example, SRPT minimizes the mean response time. As a result, designs based on these policies have been suggested for a variety of computer systems in recent years. However, the adoption of these new designs has been slow due to fears about the fairness of these policies. Specifically, there are worries that large job sizes may be "starved" of service under a policy that gives priority to small job sizes, which would result in large job sizes having response times that are unfairly long and variable [30, 210, 215, 223].

These worries have recurred nearly everywhere size based policies have been suggested. A first example is the case of web servers, where recent designs have illustrated that giving priority to requests for small files can significantly reduce response times [96, 182]. However, it is important that this improvement does not come at the expense of providing large job sizes unfairly large response times, which are typically associated with the important requests. For example, at an online shopping site the large requests are often the shopping cart transactions and at an online music site the large requests are the song and album downloads. The same tradeoff has appeared across diverse application areas. For example, UNIX processes are assigned decreasing priority based on their current age, i.e. CPU usage so far. The worry is that this may create unfairness for old processes [74]. Similar tradeoffs can be found in recent designs for routers [179, 180], wireless networks [102], transport protocols [250], and beyond.

To address these worries, it is important to develop a theoretical framework for studying the fairness of scheduling policies. However, fairness is an amorphous concept, and nearly impossible to define in a universal way. The difficulty in defining the fairness of scheduling policies is best illustrated using a few simple examples:

(i) Suppose jobs a and b are the same size, and a enters the queue slightly before b.

(ii) Suppose jobs c and d are very large and very small respectively, and job c enters the queue slightly before job d.

Most people agree that it is fair to serve job a before job b and to serve job d before job c. Thus, there is a common consensus that it is unfair for small jobs to queue behind larger ones and that it is unfair for a job

that arrived later to bypass jobs that arrived earlier. However, these are clearly competing notions of fairness – when a small job arrives some length of time after a large job, it is unclear which job it is most "fair" to serve.

Further, notice how the fair service order changes depending on the setting being considered. If the queue in question is a ticket box office, then it is more fair to serve job c before job d because, since tickets are a limited resource, it would be unfair for someone who arrived later to get a ticket if an earlier arrival does not. If the setting is a grocery store however, it is quite acceptable to allow small jobs to bypass large jobs (by using the "ten items or less lane"). If the setting is a hospital, however, things change completely. In a hospital, it is more fair to serve the more urgent job, regardless of the sizes or arrival order.

These simple examples illustrate how difficult it is to formalize what is meant by "fairness." As a result of these difficulties, the topic of fairness has not been approached in the queueing literature until this work. In this chapter we will provide an overview of our recent work introducing the first fairness metrics for the M/GI/1 queue. Additionally, we will provide results characterizing the fairness of both individual policies and scheduling classifications with respect to these new metrics. Finally, we will survey some of the newly emerging fairness metrics that other researchers have developed following our work on fairness.

In order to develop a formal definition of fairness, it is important that we first have a clear idea of what is meant by the term "fair" in the context of the applications in which we are interested. In computer applications, there are two notions of fairness that appear quite commonly. We will refer to these two types of fairness as *proportional fairness* and *temporal fairness*. Both of these notions of fairness are illustrated by the simple example that we considered at the opening of this chapter.

- Proportional fairness refers to the idea that all job sizes should receive equitable service, i.e. no job size receives disproportionately large response times.[1]
- Temporal fairness refers to the idea that it is fair to respect the seniority of jobs in the queue, i.e. it is in some sense unfair for a small job that just arrived to the queue to jump in front of the large job.

Each of these notions of fairness is appropriate, to different degrees, in a variety of applications. For instance, in web servers and routers, proportional fairness is important because it is necessary to make sure that no class of jobs is "starved" of service, i.e. all jobs receive an equitable service rate. This is especially relevant when considering priority-based designs. Temporal fairness is fundamental in e-commerce applications where it is important to guarantee that a item gets sold to the first person to request it. Similarly, in databases and other applications where data consistency is important, temporal fairness is very relevant.

Though proportional and temporal fairness are relevant to a wide range of applications, both in computer systems and other application areas, it is important to realize that these are only two possible meaning of the term "fairness." Fairness can take on entirely different meanings in other contexts. For instance, there is a large literature studying the fairness of bandwidth allocations to flows at a network level [34, 100, 114]. But, we will limit ourselves to the notions of proportional and temporal fairness in this chapter.

We will start the chapter by focusing on proportional fairness measures. Over the course of Sections 7.1-7.4, we will develop a novel framework for studying proportional fairness. To begin, we will limit our discussion to proportional fairness in expectation in Section 7.1, where we introduce a new definition of fairness and study the behavior of both individual policies and scheduling classifications. Then, in Sections

[1] Do not confuse this notion of "proportional fairness" with the one introduced in [114]. We are considering only single server queues in this chapter, while [114] considers network resource allocation.

7.2-7.4 we will extend the notion of proportional fairness we develop for the mean to a general framework for studying distributional properties of proportional fairness. Again, we will study the behavior of both individual scheduling policies and scheduling classifications using this new framework. In our exploration of proportional fairness, we find many surprises. Perhaps the biggest surprise is that, for quite a few common policies, proportional fairness is a function of load. That is, at moderate or low loads, these policies are fair to all jobs; yet at higher loads these policies become unfair. SRPT is the most well-known policy that exhibits this behavior. With respect to designing scheduling policies, we find that under high load, almost all scheduling policies are unfair. However under low load one has the opportunity to make a policy fair by sometimes increasing the priority of large jobs, e.g. SRPT allows large jobs to increase their priority as their remaining size drops.

Following our discussion of proportional fairness, we will move to a discussion of temporal fairness in Section 7.5. The metric we develop here is also new to this thesis. Again, we explore the behavior of both common individual policies and scheduling classifications. Our exploration yields many surprises. Probably the biggest surprise is that policies that "prioritize small jobs" are some of the most fair policies in this setting. Further, we prove that no policy can simultaneously provide both temporal and proportional fairness to all job sizes.

Finally, after exploring both proportional and temporal fairness individually, in Section 7.6, we survey a number of fairness metrics that emerged following our work on fairness. These proposals develop hybrid measures that combine the notions of proportional and temporal fairness into one measure.

7.1 Proportional fairness in expectation

The first notion of "fairness" that we will discuss is *proportional fairness*. The concept of proportional fairness derives intuitively from Aristotle's notion of fairness: like cases should be treated alike; different cases should be treated differently; and different cases should be treated differently in proportion to the difference at stake [187]. In the context of scheduling queues, this matches the common intuition that: small jobs should have small response times; large jobs should have large response times; and the differences in response times of small and large jobs should be proportional to the differences between the job sizes. Specifically, the response time for a job of size x, $T(x)$, should be proportional to x.

Proportional fairness arises naturally in many computer applications due to the inherent tradeoff between providing jobs of different sizes "fair" performance and providing "efficiency," which requires biasing towards small job sizes at the expense of large job sizes. This tradeoff between efficiency and fairness is often an important design constraint. For example, in the case of web servers, it has been shown that by giving priority to requests for small files, a Web server can significantly reduce response times; however it is important that this improvement not come at the cost of unfairness to requests for large files [96]. The same tradeoff applies to other application areas; for example, scheduling in supercomputing centers. Here too it is desirable to get small jobs out quickly, while not penalizing the large jobs, which are typically associated with the important customers. The tradeoff also occurs for age based policies. For example, UNIX processes are assigned decreasing priority based on their current age – CPU usage so far. This can create unfairness for old processes. To address the tension between minimizing mean response time and maintaining fairness, hybrid scheduling policies have also been proposed; for example, policies that primarily bias towards young jobs, but give sufficiently old jobs high priority as well.

7.1.1 Defining proportional fairness in expectation

Defining a metric for studying proportional fairness is a difficult task due to the amorphous nature of "fairness." In the first half of this chapter, we will develop a a unified framework for studying the proportional fairness of the entire distribution of $T(x)$; but we begin in this section by studying proportional fairness in expectation only. The following metric for fairness was introduced through a series of papers starting with Bansal and Harchol-Balter [25] and culminating with Wierman and Harchol-Balter [238].[2]

Definition 7.1 *Let $0 < \rho < 1$ in an M/GI/1 system. A job size x is treated **fairly** under policy P, service distribution X, and load ρ if*

$$E[S(x)]^P = \frac{E[T(x)]^P}{x} \leq \frac{x}{1-\rho}$$

*Otherwise, a job size x is treated **unfairly**. A scheduling policy P is **fair** if every job size is treated fairly. Otherwise P is **unfair**.*

Definition 7.1 consists of two pieces: a metric, $E[S(x)] = E[T(x)]/x$, and a criterion, $1/(1-\rho)$. The metric clearly relates to the intuition that the response time of a job should be proportional the size of the job; however the criterion is less intuitive. There are two motivations for this choice of metric and criterion.

1. PS is typically thought of as a fair policy because at every instant every job in the system receives an equal share of the server. This matches Rawls' theory of social justice [183]. Further, PS satisfies the idea that $E[T(x)]$ should be proportional to x, since the slowdown under PS is constant across x: $E[S(x)]^{PS} = 1/(1-\rho)$. In fact, among policies with constant slowdown PS minimizes mean response time (see Theorem 7.9). Thus, a scheduling policy P can intuitively be viewed as unfair if jobs of some size x have $E[T(x)]^P > E[T(x)]^{PS} = x/(1-\rho)$.

2. More formally, when comparing $E[T(x)]^P$ across x, we want a *metric* that scales $E[T(x)]^P$ appropriately to allow for comparison of $E[T(x)]^P$ between small and large x. For $E[T(x)]^P$, it is clear that $1/x$ is an appropriate scaling factor because $E[T(x)]^P = \Theta(x)$ under all work conserving scheduling policies [97], and thus we need to normalize by the growth rate. The *criterion* $1/(1-\rho)$ stems from two formal motivations. First, it provides a min-max notion of fairness: $\min_P \max_x E[T(x)]^P/x = 1/(1-\rho)$ (see Theorem 7.9). Second, $1/(1-\rho)$ provides a criterion that distinguishes between patterns of behavior of policies with respect to the metric $E[T(x)]^P/x$.

With Definition 7.1 in hand, it is now possible to classify scheduling policies based on whether they (i) treat all job sizes fairly or (ii) treat some job sizes unfairly. Curiously, we find that some policies may fall into either type (i) or type (ii) depending on the system load. We therefore define *three classes of unfairness*.

Definition 7.2 *Let $0 < \rho < 1$ in an M/GI/1 queue where X is non-deterministic.[3] A scheduling policy P is: (i) **Always Fair** if P is fair for all such ρ and X; (ii) **Sometimes Fair** if P is fair under some ρ and X and unfair under other ρ and X; or (iii) **Always Unfair** if P is unfair under all loads and service distributions.*

[2] Note that throughout the remainder of this section we will refer to "proportional fairness" as simply "fairness" in order to simplify the exposition. This also matches with the literature where the term fairness has been typically used, [25, 78, 87, 97, 179, 185, 238]. We will use the term "proportional fairness" only when it is necessary to distinguish it from the other notions of fairness discussed in this chapter.

[3] We exclude deterministic distributions because the concept of proportional fairness is only interesting when there exist jobs of different sizes.

Note that Definition 7.2 is a generalization of the definition in [238]. Initially, in [238], only distributions with finite variance were considered. However, more recently Brown [47] was able to extend many of the results to allow the consideration of distributions with *infinite* variance. The extension led to some counter-intuitive results that we will discuss later in the thesis.

In this remainder of this section, we will study the fairness of both individual scheduling policies and the fairness of scheduling classifications. Our goal will be to classify scheduling policies, techniques, and heuristics as either Always Fair, Always Unfair, or Sometimes Fair. Further, we will characterize the degree of unfairness under many common individual policies. The main results are summarized in Figure 7.1. Our aim in providing this taxonomy is, first, to allow researchers to judge the unfairness of existing policies and, second, to provide heuristics for the design of new scheduling policies. Since very little analytical prior work exists on understanding the unfairness of scheduling policies, and what does exist is isolated to a handful of individual policies [96, 179], this section represents the first broad study of the unfairness of scheduling policies.

In our attempts to understand unfairness, we find many surprises. Perhaps the biggest surprise is that for quite a few common policies, unfairness is a function of load and variability. In particular, some policies are fair at moderate or low loads, but become unfair under higher loads. Further, some policies are fair for highly variable service distributions (with $E[X^2] = \infty$), but are unfair if $E[X^2] < \infty$. SRPT is a well-known policy that exhibits both of these behaviors.

These surprises have a direct impact on designing scheduling policies for computer applications. In particular, we find that under high load, almost all scheduling policies are unfair; but that under low load one has the opportunity to make a policy fair by sometimes increasing the priority of large jobs. For example, PSJF and SRPT have very similar behavior and delay characteristics, but because SRPT allows large jobs to increase their priority it is more fair under low loads.

Since so many policies are Always Unfair, and so many others are Sometimes Fair, it is important to ask *who* is being treated unfairly. We present a number of results characterizing who is being treated unfairly and how unfairly they are treated. Initially it may seem that unfairness is an increasing function of job size, with the largest job being treated the most unfairly. This is in fact the case for most bounded job size distributions. However, for unbounded job size distributions, we find that this is usually not the case. Instead, under many policies, unfairness is monotonically increasing with job size up to a particular point; and later is monotonically decreasing with job size. Thus, the job being treated most unfairly is far from the largest (see Figure 7.2). Interestingly, the position of this "hump" changes as a function of load.

7.1.2 The proportional fairness of individual policies

To get a feel for Definition 7.1, it is useful to begin by studying the behavior of common individual policies with respect to fairness.

As a first step, it is easy to see that each of FSP, PS, and PLCFS are Always Fair because

$$E[S(x)]^{FSP} \leq E[S(x)]^{PLCFS} = E[S(x)]^{PS} = \frac{1}{1-\rho}$$

Thus, there are a number of policies that are fair under all service distributions and all loads.

However, most common policies are not Always Fair. For example, we will see that FCFS, and all other non-preemptive blind policies, are Always Unfair (Section 7.1.3.2) because the smallest job size in

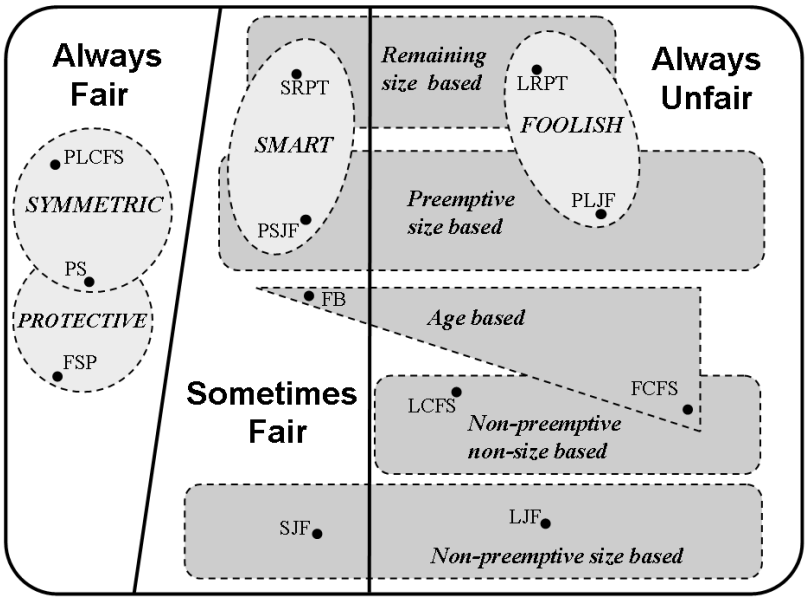


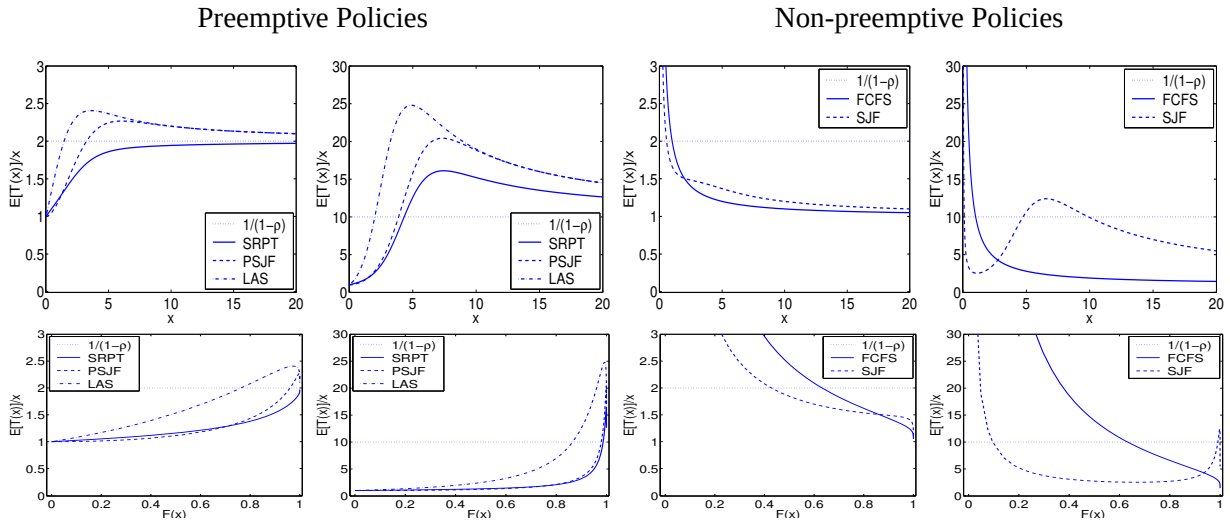
Figure 7.1: *An illustration of the classification of common prioritization techniques and heuristics with respect to proportional fairness.*

the service distribution will always be treated unfairly. Further, we will see that by giving priority to small jobs, it is possible to be Sometimes Fair. In particular, we will see that SRPT and FB are Sometimes Fair. FB is unfair when the service distribution has finite variance, but can be fair when the service distribution has infinite variance and SRPT is unfair if load is high and the service distribution has finite variance, but can be fair if the load is moderate or if the service distribution has infinite variance. Further, we will see that even in the settings where these policies are unfair, the degree of unfairness to large job sizes is not as bad as one might expect.

In this section, we will focus only on FB and SRPT due to their practical importance, however, we have also analyzed the fairness of many other common policies in [238] and we summarize these results in Figure 7.1 and Figure 7.2. We do not include the analyses here because they are similar in tone to the analyses of SRPT and FB; thus it would have become to repetitive for the reader.

7.1.2.1 FB

Given the bias that FB provides for small jobs (since they are always young), it is natural to ask about the performance of the large jobs. Thus, understanding the growth of slowdown as a function of the job size x is important. It turns out that the performance of FB is strongly related to that of PSJF. The first results on the fairness of FB were published simultaneously by Wierman & Harchol-Balter [238] and Rai, Urvoy-Keller, & Biersack [179] under the assumption that $E[X^2] < \infty$. In this setting it was found that FB was always unfair. However, recently Brown found, surprisingly, that FB can be fair when the service distribution has infinite variance [47]. We summarize the major results in the following theorem:



(a) Low load ($\rho = 0.5$) (b) High load ($\rho = 0.9$) (c) Low load ($\rho = 0.5$) (d) High load ($\rho = 0.9$)

Figure 7.2: *The conditional mean is illustrated under a variety of both preemptive and non-preemptive policies. The service distribution is exponential with mean 1. The dotted line shows the criteria for proportional fairness.*

Theorem 7.1

FB *is Sometimes Fair.* FB *is unfair when* $E[X^2] < \infty$. *However,* FB *is fair when the service distribution is regularly varying with rate* $\alpha \in (1, 1.5)$.

Theorem 7.1 illustrates that, surprisingly, in many cases FB is fair to all job sizes. In fact, FB is often fair in practical settings since workloads in many computer applications are thought to be regularly varying with $\alpha \in (1, 1.5)$.

Though Theorem 7.1 says that FB can be fair in some situations, more often than not, FB is unfair. Thus, it is important to understand which job sizes FB is unfair to and how unfairly these job sizes are treated. To provide an answer to this question, let us consider the case of of $E[X^2] < \infty$ in more detail. We will limit our discussion to unbounded service distribution, but case of bounded service distributions can be handled similarly.

Theorem 7.2

In an M/GI/1 FB *queue with* $E[X^2] < \infty$ *there is some job size* y *such that for all* $x > y$, $E[S(x)]^{FB} > 1/(1-\rho)$ *under any unbounded service distribution, for all* ρ. *Furthermore,* $E[S(x)]^{FB}$ *is not monotonic in* x.

Proof. Notice that $\lim_{x\to\infty} E[S(x)]^{FB} = 1/(1-\rho)$ since the service distribution is assumed to have finite variance. As a result, to prove the theorem it is sufficient to show that $\frac{d}{dx}E[S(x)]$ converges to zero from below as $x \to \infty$.

By observing that

$$\begin{aligned}\frac{d}{dx}E[S(x)]^{FB} &= \frac{d}{dx}\frac{E[T(x)]^{FB}}{x} \\ &= \frac{x\frac{d}{dx}E[T(x)]^{FB} - E[T(x)]^{FB}}{x^2},\end{aligned}$$

our goal reduces to showing that as $x \to \infty$

$$x\frac{d}{dx}E[T(x)]^{FB} - E[T(x)]^{FB} < 0 \tag{7.1}$$

Let us begin by differentiating response time:

$$x\frac{d}{dx}E[T(x)]^{FB} = \frac{2\lambda^2\overline{F}(x)x\int_0^x t\overline{F}(t)dt}{(1-\widetilde{\rho}(x))^3} + \frac{2\lambda x^2\overline{F}(x)}{(1-\widetilde{\rho}(x))^2} + \frac{x}{1-\widetilde{\rho}(x)}$$

which gives us

$$x\frac{d}{dx}E[T(x)]^{FB} - E[T(x)]^{FB} = \left(\frac{2\lambda^2\overline{F}(x)x\int_0^x t\overline{F}(t)dt}{(1-\widetilde{\rho}(x))^3}\right) + \left(\frac{2\lambda x^2\overline{F}(x)}{(1-\widetilde{\rho}(x))^2} - \frac{\lambda\int_0^x t\overline{F}(t)dt}{(1-\widetilde{\rho}(x))^2}\right) \tag{7.2}$$

Recall from (7.1) that the above gives us the sign of $\frac{d}{dx}E[S(x)]^{FB}$. There are two terms in (7.2). The first term is clearly positive. Notice that for x such that $\overline{F}(x) \geq \frac{1}{4}$ we have:

$$\begin{aligned}x\frac{d}{dx}E[T(x)]^{FB} - E[T(x)]^{FB} &\geq \frac{\lambda}{(1-\widetilde{\rho}(x))^2}\left(2x^2\overline{F}(x) - \frac{1}{2}x^2\right) \\ &\geq 0\end{aligned}$$

which shows that $E[S(x)]^{FB}$ is monotonically increasing for x such that $F(x) \leq \frac{3}{4}$.

We now prove that the expected slowdown converges to $1/(1-\rho)$ from above as $x \to \infty$. First, we know that $\lim_{x\to\infty} E[S(x)]^{FB} = 1/(1-\rho)$ [97]. Next, (7.2) gives us the sign of $\frac{d}{dx}E[S(x)]^{FB}$. Note that, for any distribution with finite second moment, we know that $\overline{F}(x) = o(x^{-2})$. Using this observation and the fact that $\widetilde{\rho}(x) \to \rho$ as $x \to \infty$,

$$\lim_{x\to\infty} x\frac{d}{dx}E[T(x)]^{FB} - E[T(x)]^{FB} = \frac{-\lambda E[X^2]}{2(1-\rho)^2} < 0$$

Thus, there exists some job size x_0 such that for all $x > x_0$, $E[S(x)]^{FB}$ is monotonically decreasing in x. □

The proof of this theorem shows us that all job sizes greater than a certain size have higher mean response time under FB than under PS. Counter-intuitively however, the job that is treated the most unfairly is not the largest job. Thus, the intuition that by helping the small jobs FB must hurt the biggest jobs is not entirely true.

Interestingly, this theorem is counter to the common portrayal of FB in the literature. When investigating $E[S(x)]^{FB}$, previous literature has used percentile plots, which hide the behavior of the largest one percent of the jobs. When looking at the same plots as a function of job size the presence of a hump becomes evident. This contrast is illustrated in Figure 7.2. In fact, even under bounded distributions this hump exists, regardless of the bound placed on x.

Having shown that some job sizes are treated unfairly under FB scheduling, it is next interesting to understand exactly which job sizes are seeing poor performance. The following theorem places a lower bound on the size of jobs that can be treated unfairly.

Theorem 7.3
In an M/GI/1 queue, for x such that $\widetilde{\rho}(x) \leq 1 - \sqrt{1-\rho}$, $E[T(x)]^{FB} \leq 1/(1-\rho)$

Proof. The proof will proceed by simply manipulating $E[T(x)]^{FB}$.

$$\begin{aligned} E[T(x)]^{FB} &= \frac{\lambda \int_0^x t\overline{F}(t)dt}{(1-\widetilde{\rho}(x))^2} + \frac{x}{1-\widetilde{\rho}(x)} \\ &\leq \frac{\lambda x \int_0^x \overline{F}(t)dt}{(1-\widetilde{\rho}(x))^2} + \frac{x}{1-\widetilde{\rho}(x)} \\ &= \frac{\widetilde{\rho}(x)x}{(1-\widetilde{\rho}(x))^2} + \frac{x(1-\widetilde{\rho}(x))}{(1-\widetilde{\rho}(x))^2} = \frac{x}{(1-\widetilde{\rho}(x))^2} \end{aligned}$$

Letting $\widetilde{\rho}(x) \leq 1 - \sqrt{1-\rho}$ we complete the proof of the theorem.
□

It is important to notice that as ρ increases, so does the lower bound $1 - \sqrt{1-\rho}$ on $\widetilde{\rho}(x)$. In fact, this bound converges to 1 as $\rho \to 1$, which signifies that the size of the smallest job that might be treated unfairly is increasing unboundedly as ρ increases.

7.1.2.2 SRPT

SRPT has long been known to optimize $E[T]$. However, its use in practice has been hindered by the fear that large job sizes experience unfairly long response times. The fairness of SRPT was first studied by Bansal & Harchol-Balter [25] under the assumption that $E[X^2] < \infty$. These initial results were later extended by Wierman & Harchol-Balter [238], and then by Brown [47], who was the first to consider fairness when $E[X^2] = \infty$. We summarize the major results in the following theorem:

Theorem 7.4
SRPT is Sometimes Fair. SRPT *is fair when $\rho \leq 0.5$ or when the service distribution is regularly varying with rate $\alpha \in (1, 1.5)$. However, when $E[X^2] < \infty$, under all service distributions there exists a $\rho_c < 1$ such that for all $\rho > \rho_c$* SRPT *is unfair.*

Theorem 7.4 illustrates that, surprisingly, in many cases SRPT is fair to all job sizes. Thus, in many cases it is possible to optimize $E[T]$ while still providing fair response times to all job sizes. In particular, when the system load is small enough or the tail of the service distribution is heavy enough, SRPT is fair.

In fact, SRPT is often fair in practical settings since workloads in many computer applications are thought to be regularly varying with $\alpha \in (1, 1.5)$.

Though Theorem 7.4 says that SRPT can be fair in many situations, in many cases SRPT is unfair, e.g. under high load. We illustrate this behavior in Figure 7.2. Thus, it is important to understand which job sizes SRPT is unfair to and how unfairly these job sizes are treated. To provide an answer to this question, let us consider the case of of $E[X^2] < \infty$ in more detail.

Theorem 7.5

In an M/GI/1 queue with $E[X^2] < \infty$, for x such that $\rho(x) \leq \frac{1}{2}$, $E[S(x)]^{SRPT}$ is monotonically increasing in x.

Proof. The proof will follow the same technique that we used in proving Theorems 7.2 and 7.1. Begin by noting that

$$m_2(x) = \int_0^x t^2 f(t) = 2\int_0^x t\overline{F}(t)dt - 2x^2\overline{F}(x)$$

Then we can derive

$$x \cdot \frac{d}{dx}E[T(x)]^{SRPT} = \frac{2\lambda^2 f(x)x^2 \int_0^x t\overline{F}(t)dt}{(1-\rho(x))^3} + \frac{\lambda x^2 \overline{F}(x)}{(1-\rho(x))^2} + \frac{x}{1-\rho(x)}$$

which gives us

$$\begin{aligned} x \cdot \frac{d}{dx}E[T(x)]^{SRPT} - E[T(x)]^{SRPT} &= \left(\frac{2\lambda^2 f(x)x^2 \int_0^x t\overline{F}(t)dt}{(1-\rho(x))^3}\right) + \left(\frac{\lambda x^2\overline{F}(x)}{(1-\rho(x))^2} - \frac{\lambda\int_0^x t\overline{F}(t)dt}{(1-\rho(x))^2}\right) \\ &\quad + \left(\frac{x}{1-\rho(x)} - \int_0^x \frac{dt}{1-\rho(t)}\right) \\ &= \left(\frac{2\lambda^2 f(x)x^2 \int_0^x t\overline{F}(t)dt}{(1-\rho(x))^3}\right) - \left(\frac{\lambda m_2(x)}{2(1-\rho(x))^2}\right) \\ &\quad + \left(\frac{x}{1-\rho(x)} - \int_0^x \frac{dt}{1-\rho(t)}\right) \end{aligned}$$

This expression provides us with the sign of the derivative of slowdown. There are 3 terms in the above expression. The first of these terms is clearly positive. The third of these terms is also clearly positive. We will complete the proof by showing that the third term is of larger magnitude than the second term.

To obtain a bound on the third term, we can quickly show that

$$\begin{aligned} \frac{x}{1-\rho(x)} - \int_0^x \frac{dt}{1-\rho(t)} &= \int_0^x \frac{(1-\rho(t)) - (1-\rho(x))}{(1-\rho(t))(1-\rho(x))}dt \\ &\geq \frac{1}{1-\rho(x)}\int_0^x \rho(x) - \rho(t)dt \end{aligned} \tag{7.3}$$

To further specify this bound we can compute

$$\begin{aligned}\int_0^x \rho(t)dt &= \lambda \int_0^x \int_0^t sf(s)dsdt \\ &= \lambda \int_0^x \int_s^x sf(s)dtds \\ &= \lambda \int_0^x sf(s)(x-s)ds \\ &= \rho(x)x - \lambda m_2(x) \end{aligned} \tag{7.4}$$

Finally, putting all three terms back together we see that when $\rho(x) \leq \frac{1}{2}$,

$$\begin{aligned} x \cdot \frac{d}{dx}E[T(x)]^{SRPT} - E[T(x)]^{SRPT} &= \left(\frac{2\lambda^2 f(x)x^2 \int_0^x t\overline{F}(t)dt}{(1-\rho(x))^3}\right) - \left(\frac{\lambda m_2(x)}{2(1-\rho(x))^2}\right) \\ &\quad + \left(\frac{x}{1-\rho(x)} - \int_0^x \frac{dt}{1-\rho(t)}\right) \\ &\geq -\left(\frac{\lambda m_2(x)}{2(1-\rho(x))^2}\right) + \left(\frac{\lambda m_2(x)}{1-\rho(x)}\right) \\ &\geq 0 \end{aligned} \tag{7.5}$$

□

Corollary 7.6
In an M/GI/1 queue with $E[X^2] < \infty$, if $\rho \leq \frac{1}{2}$, $E[S(x)]^{SRPT}$ is monotonically increasing for all x. Furthermore $E[S(x)]^{SRPT} \leq 1/(1-\rho)$ for all x.

Proof. This follows immediately from the above theorem and Theorem 7.17, which gives that: for any work conserving scheduling policy P, $\lim_{x\to\infty} E[S(x)]^P \leq 1/(1-\rho)$.
□

Having seen that SRPT is Sometimes Fair, it is interesting to consider which job sizes are being treated fairly/unfairly. The following theorem shows that as ρ increases, the number of jobs being treated fairly also increases.

Theorem 7.7
In an M/GI/1 queue with $E[X^2] < \infty$, for x such that $\rho(x) \leq \max\{1-\sqrt{1-\rho}, \frac{1}{2}\}$, $E[T(x)]^{SRPT} \leq 1/(1-\rho)$.

The proof follows immediately from Theorem 7.3, Theorem 7.5, and Theorem 3.19, which bounds the performance of SRPT by that of FB.

7.1.3 The proportional fairness of scheduling classifications

The power of definitions 7.1 and 7.2 are that they are simple enough to be tractable for the analysis of the fairness of scheduling classifications, not just individual scheduling policies. For the remainder of this section, we will focus on understanding the impact of common scheduling heuristics and techniques on the fairness of the resulting policies. We will start by studying the techniques and heuristics that lead to Always Fair policies, then we will move through the techniques that are Always Unfair, and finally we will study the techniques and heuristics that are Sometimes Fair.

We provide an overview of the known results in Figure 7.1. However, to avoid repetition, we will not provide the proofs for all the classifications listed in Figure 7.1 since many of the proofs are very similar to one another. Instead, we will illustrate the important proof techniques using a handful of classifications here, and refer the reader to [238] for the proofs not provided.

7.1.3.1 Always Fair

We start our study of scheduling classifications by studying heuristics and techniques that are Always Fair, i.e. policies that are fair to all job sizes under all loads and all service distributions.

Two of our heuristic-based classifications are Always Fair: the SYMMETRIC and the PROTECTIVE classes. This is not surprising. Clearly, all $P \in$ SYMMETRIC are Always Fair since they all have $E[S(x)]^P = E[S(x)]^{PS} = 1/(1-\rho)$. Further, since all PROTECTIVE policies guarantee that no job finishes later than it would under PS, all PROTECTIVE policies are also Always Fair.

Proposition 7.8
In an M/GI/1 queue, all SYMMETRIC *policies are Always Fair. In addition, all* PROTECTIVE *policies are Always Fair.*

Outside of these two classes, we are not aware of any Always Fair policies. However, we can prove a necessary condition for Always Fair policies, which will be useful when showing that policies are *not* Always Fair.

Theorem 7.9
In an M/GI/1 queue with $E[X^2] < \infty$,

$$\min_P \max_x E[S(x)]^P = \frac{1}{1-\rho}$$

Further, if scheduling policy P is Always Fair, then

$$\lim_{x\to\infty} E[S(x)]^P = 1/(1-\rho)$$

Note that in addition to providing a necessary condition for the Always Fair class, Theorem 7.9 serves as a key justification for the notion of proportional fairness defined in Definition 7.1. Once we take $E[S(x)]$ as a metric for fairness, the fact that $1/(1-\rho)$ is the min-max value of $E[S(x)]$ provides a crucial justification for using $1/(1-\rho)$ as a fairness criterion.

Proof. First, because P is Always Fair, $E[S(x)]^P \leq 1/(1-\rho)$ for all x, and therefore $\lim_{x\to\infty} E[S(x)]^P \leq 1/(1-\rho)$. Thus, we need only show that $\lim_{x\to\infty} E[S(x)]^P \geq 1/(1-\rho)$. We accomplish this by bounding

the expected slowdown for a job of size x from below, and then showing that the lower bound converges to $1/(1-\rho)$ as we let $x \to \infty$.

To lower bound the expected slowdown, we consider a modified policy $Q_{x,a}$ that throws away all arrivals whose response time under P is greater than or equal to a and also throws away arrivals with size greater than x. Further, $Q_{x,a}$ works on the remaining jobs at the exact moments that P works on these jobs. We will begin by calculating the load made up of jobs of size less than y (where $y < a < x$) under $Q_{x,a}$, $\rho(y)^{Q_{x,a}}$. By Markov's Inequality we obtain $P(T(y)^P < a) \geq 1 - \frac{y}{a(1-\rho)}$. Thus, we see that

$$\begin{aligned} \rho(y)^{Q_{x,a}} &\geq \lambda \int_0^y \left(1 - \frac{t}{a(1-\rho)}\right) t f(t) dt \\ &= \rho(y)^P - \frac{\lambda m_2(y)}{a(1-\rho)} \end{aligned}$$

where $\rho(y)^P = \lambda \int_0^y t f(t) dt$ is the load made up by jobs of size less than or equal to y in P and $m_2(y) = \int_0^y t^2 f(t) dt$. The intuition behind the remainder of the proof is that as a, y, and x get very large, $\rho(y)^{Q_{x,a}}$ approaches ρ which tells us that the load of jobs that *must* complete before x under P goes to ρ.

We now derive a lower bound on the response time of a job of size x under policy P. We will be interested in large x, with $a < x$. We divide $T(x)^P$ into two parts T_1 and T_2 where T_1 represents the time from when x starts service until it has remaining size a and T_2 represents the time from when x has remaining size a until it completes service. We first note that $T_2 \geq a$. To lower bound T_1 consider the set of jobs, S_y, with size less than y and whose response time under P is less than a. The jobs in S_y are worked on at the same moments under $Q_{x,a}$ and P, and they comprise load $\rho(y)^{Q_{x,a}}$. During time T_1, job x receives service under P at most during the time the system is idle of jobs in S_y, which is $1 - \rho(y)^{Q_{x,a}}$ fraction of the time. Thus

$$E[T_1] \geq \frac{x-a}{1-\rho(y)^{Q_{x,a}}}.$$

It follows that

$$\begin{aligned} E[T(x)]^P &= E[T_1] + E[T_2] \geq \frac{x-a}{1-\rho(y)^{Q_{x,a}}} + a \\ E[S(x)]^P &\geq \frac{x-a}{x\left(1 - \rho(y)^P + \frac{\lambda m_2(y)}{a(1-\rho)}\right)} + \frac{a}{x} \end{aligned}$$

Now, we must set y and a as functions of x such that, as we let $x \to \infty$, we converge as desired. Notice that as $x \to \infty$, we would like $\rho(y)^P \to \rho$, $\frac{\lambda m_2(y)}{a(1-\rho)} \to 0$, and $\frac{a}{x} \to 0$. Thus, we must have $a \ll x$ such that $y \to \infty$ and $a \to \infty$. We can accomplish this by setting $a = 4\sqrt{x}$ and $y = \sqrt{x}$. Notice that

$m_2(\sqrt{x}) \to E[X^2] < \infty$ as $x \to \infty$. Now, looking at expected slowdown we see that as $x \to \infty$:

$$\begin{aligned} E[S(x)]^P &\geq \frac{x - 4\sqrt{x}}{x\left(1 - \rho(\sqrt{x}) + \frac{\lambda m_2(\sqrt{x})}{4\sqrt{x}(1-\rho)}\right)} + \frac{4\sqrt{x}}{x} \\ &= \frac{1 - 4/\sqrt{x}}{1 - \rho(\sqrt{x}) + \frac{\lambda m_2(\sqrt{x})}{4\sqrt{x}(1-\rho)}} + \frac{4}{\sqrt{x}} \\ &\to \frac{1}{1-\rho} \end{aligned}$$

□

7.1.3.2 Always Unfair

We will now prove that a large number of common scheduling techniques and heuristics are Always Unfair, i.e. are guaranteed to treat some job size unfairly under all system loads and service distributions. For an overview of scheduling techniques that are Always Unfair, see Figure 7.1. The policies in the Always Unfair class exhibit fundamentally different behavior with respect to $E[S(x)]$ than those in the Always Fair class. While policies in the Always Fair class have either monotonically increasing or constant $E[S(x)]$, the policies we study here typically exhibit decreasing behavior (see Figure 7.2).

Non-preemptive blind policies

The analysis in this section is based on the simple observation that any policy where a small job cannot preempt the job in service will be unfair to small jobs. In fact, the analysis holds for all non-preemptive policies as long as the service distribution includes jobs of arbitrarily small sizes.

Lemma 7.10

In an M/GI/1 queue with $E[X^2] < \infty$[4], any non-preemptive policy P is unfair for all loads under any service distribution defined on a neighborhood of zero.

Proof. We can bound the performance of P by noticing that, at a minimum, an arriving job of size x must take x time plus the excess of the job that is serving. Thus, $E[T(x)]^P \geq x + \frac{\rho E[X^2]}{2E[X]}$. Notice that $\lim_{x\to 0} E[S(x)]^P = \infty$. Thus, there exists some job size y such that $E[S(y)]^P > 1/(1-\rho)$, for all $\rho < 1$. □

Under service distributions with non zero lower bounds on the smallest job size, not all non-preemptive policies are Always Unfair. However, all non-preemptive blind policies can be classified as Always Unfair. (Note that the remainder of the possible non-preemptive policies are explored in Section 7.1.3.3.)

Theorem 7.11

In an M/GI/1 queue with $E[X^2] < \infty$, all non-preemptive blind policies P are Always Unfair.

[4]Note that non-preemptive policies require that $E[X^2] < \infty$ in order for $E[T] < \infty$

Proof. Assume that the service time distribution has lower bound $C > 0$ (we have already dealt with the case of $C = 0$). We will show that jobs of size C are treated unfairly. Recall that all non-preemptive, blind based policies have the same expected response time for a job of size x

$$\begin{aligned} E[T(C)]^P &= C + \frac{\lambda E[X^2]}{2(1-\rho)} \\ &= \frac{C(1-\rho) + \lambda \int_0^\infty (t+C)\overline{F}(t+C)dt}{1-\rho} \\ &= \frac{C - C\rho + C\rho + \lambda \int_0^\infty t\overline{F}(t+C)dt}{1-\rho} \\ &> \frac{C}{1-\rho} \end{aligned}$$

where the last inequality follows since the service distribution is required to be non-deterministic.
□

FOOLISH scheduling

We now analyze the class of FOOLISH policies (i.e. policies that prioritize large job sizes). The two most common examples of such policies are Preemptive-Longest-Job-First (PLJF) and Longest-Remaining-Time-First (LRPT). It is not surprising that FOOLISH policies will always be unfair to small job sizes, since large job sizes have preemptive priority.

Theorem 7.12
In an M/GI/1 queue with $E[X^2] < \infty$[5], all FOOLISH *policies are Always Unfair.*

Proof. Let x_L be the lower bound of the service distribution. Then, as $x \to x_L$, $E[T(x)]^{PLJF} \to E[B(x_L + W)]$ since the service distribution is continuous. Thus, under all $P \in$ FOOLISH, as $x \to x_L$, $E[T(x)]^P \to E[B(x_L + W)]$. So, all $P \in$ FOOLISH are Always Unfair since $E[B(x_L + W)] > E[B(x_L)]$ for $\rho > 0$.
□

7.1.3.3 Sometimes Fair

We now move to the class of Sometimes Fair policies – policies that for some ρ, and X treat all job sizes fairly, but for other ρ and X treat some job size unfairly. For an overview of scheduling techniques that are Sometimes Fair, see Figure 7.1. The policies that are Sometimes Fair have more complicated behavior with respect to $E[S(x)]$ than we observed in the cases of the Always Fair and Always Unfair classes. For instance, SRPT maintains monotonic $E[S(x)]$ for low loads similarly to policies in the Always Fair class, but SRPT exhibits non-monotonic behavior under high enough load. This behavior is illustrated in Figure 7.2.

[5]Note that FOOLISH policies require that $E[X^2] < \infty$ in order for $E[T] < \infty$

Non-preemptive, size-based policies

This section completes the analysis of non-preemptive policies begun in Section 7.1.3.2. It is based on the observation that if there is a lower bound on the smallest job size in the service distribution, then it is possible for a non-preemptive policy to avoid being Always Unfair.

Theorem 7.13

In an M/GI/1 queue with $E[X^2] < \infty$, any non-preemptive, size-based policy P is either Sometimes Fair or Always Unfair.

Proof. We will prove in Section 7.2 that $\lim_{x\to\infty} E[S(x)]^P = 1$ for all non-preemptive policies P. Thus, we can apply Theorem 7.9 to conclude that a non-preemptive policy P cannot attain Always Fair, so P must be either Always Unfair or Sometimes Fair.

Further, observe there are examples of size based, non-preemptive policies in each of the two classes. For instance, it can easily be shown that the Longest-Job-First (LJF) policy is Always Unfair. However, Shortest-Job-First (SJF) is only Sometimes Fair – that is, there exist service distributions and loads such that $E[S(x)]^{SJF} \leq 1/(1-\rho)$ for all x. One example of such a distribution and load is $(X-2) \sim Exp(1)$ with $\rho = 0.2$.
□

SMART Scheduling

The fairness of SMART policies is of particular interest due to the intrinsic bias against large job sizes under these policies. It is well known that policies that bias towards small job sizes or jobs with small remaining service times perform well with respect to mean response time and mean slowdown, but the concerns about fairness of such policies has led to their limited acceptance in computer applications.

We have already seen one example of a SMART policy that is Always Unfair. However, it is possible for SMART policies to be fair. In fact we find that many SMART policies (e.g. SRPT) can be fair under low loads regardless of the service distribution. Further, if the service distribution is highly variable, SMART policies can be fair at even high loads. However, under every SMART policy, under every service distribution with $E[X^2] < \infty$, there exists some load that is high enough to cause the policy to be unfair. Interestingly though, as we saw in PSJF, it is not the largest job size that is treated the most unfairly.

Theorem 7.14

In an M/GI/1 queue, all SMART *policies are Sometimes Fair.*

In the remainder of this section, we will prove the above result for the class of SMART policies. Then, we will focus on the specific case of SRPT in the next section.

Before we can prove Theorem 7.14, we first need to develop a few technical lemmas about the behavior of $E[T(x)]^{SRPT}$.

Lemma 7.15

Let $h(x) > 0$ be a continuous, increasing function of x such that $\int_0^x h(x) = H(x)$.

$$\int_0^x \frac{h(t)}{(1-\rho(t))^i} dt \quad \geq \quad \frac{H(x)}{(1-\rho(x)+\lambda m_2(x)/x)^i}$$

Proof. We prove this using Chebyshev's Integral Inequality [89]. The following holds for all $i \geq 1$.

$$\left(\int_0^x 1-\rho(t)dt\right)\left(\int_0^x \frac{h(t)}{(1-\rho(t))^i}dt\right) \quad \geq \quad x\int_0^x \frac{h(t)}{(1-\rho(t))^{i-1}}dt$$

Thus,

$$\begin{aligned}\int_0^x \frac{h(t)}{(1-\rho(t))^i}dt &\geq \frac{x^i H(x)}{\left(\int_0^x 1-\rho(t)dt\right)^i} \\ &= \frac{H(x)}{(1-\rho(x)+\lambda m_2(x)/x)^i}\end{aligned}$$

□

Lemma 7.16
Define $\delta_x = \lambda m_2(x)/x$ and let $\epsilon > 0$. Then

$$\frac{x(1+\epsilon)}{1-\rho(x)} - \int_0^x \frac{dt}{1-\rho(t)} \quad \leq \quad \frac{\lambda m_2(x)(1+\epsilon)+(1-\rho(x))x\epsilon}{(1-\rho(x)+\delta_x)(1-\rho(x))}$$

Proof. We will use the bound in Lemma 7.15 in the first step, and then calculate directly.

$$\begin{aligned}\frac{x(1+\epsilon)}{1-\rho(x)} - \int_0^x \frac{dt}{1-\rho(t)} &\leq \frac{x(1+\epsilon)}{1-\rho(x)} - \frac{x}{1-\rho(x)+\lambda m_2(x)/x} \\ &= \frac{x}{1-\rho(x)+\delta_x}\left((1+\epsilon)\frac{1-\rho(x)+\delta_x}{1-\rho(x)} - 1\right) \\ &= \frac{x\delta_x(1+\epsilon)+(1-\rho(x))x\epsilon}{(1-\rho(x))(1-\rho(x)+\delta_x)}\end{aligned}$$

□

Now, we are now ready to prove the main result.

Proof of Theorem 7.14. Note that it follows immediately from Brown's results for FB and SRPT [47] that all SMART policies are fair if job sizes are regularly varying with $\alpha \in (1, 1.5)$. Thus, we need only show that all SMART policies are unfair if $E[X^2] < \infty$ for large enough ρ to complete the proof.

Let $P \in$ SMART and $\delta_x = \lambda m_2(x)/x$. We will start by proving the result in the case of an unbounded service distribution. Let

$$\epsilon_x = \frac{\rho - \rho(x)}{1-\rho}$$

such that $1+\epsilon_x = (1-\rho(x))/(1-\rho)$. Note that $x(\rho-\rho(x)) = \lambda x\int_x^\infty tf(t)dt \leq \lambda\int_x^\infty t^2 f(t)dt = o(1)$ since $E[X^2] < \infty$. Thus, $\epsilon_x = o(1/x)$.

Now, we can calculate using Lemma 7.16:

$$\begin{aligned}
\frac{x}{1-\rho} - E[T(x)]^P &= \frac{x(1+\epsilon_x)}{1-\rho(x)} - E[T(x)]^P \\
&\leq \frac{x(1+\epsilon_x)}{1-\rho(x)} - \int_0^x \frac{dt}{1-\rho(t)} - \frac{\lambda m_2(x)}{2(1-\rho(x))^2} \\
&\leq \frac{\lambda m_2(x)(1+\epsilon_x) + (1-\rho(x))x\epsilon_x}{(1-\rho(x))(1-\rho(x)+\delta_x)} - \frac{\lambda m_2(x)}{2(1-\rho(x))^2} \\
&= \frac{\lambda m_2(x)}{1-\rho(x)} \left(\frac{1+\epsilon_x + (1-\rho(x))\epsilon_x/\delta_x}{1-\rho(x)+\delta_x} - \frac{1}{2(1-\rho(x))} \right)
\end{aligned}$$

Thus, P is unfair to x when the following equivalent statements hold

$$\begin{aligned}
2(1-\rho(x))(1+\epsilon_x + (1-\rho(x))\epsilon_x/\delta_x) - (1-\rho(x)+\delta_x) &< 0 \qquad (7.6) \\
(1-\rho(x))(1+2\epsilon_x + 2(1-\rho(x))\epsilon_x/\delta_x) - \delta_x &< 0 \\
(1-\rho(x))(1+o(1)) - \delta_x &< 0 \\
(1-\rho)(1+o(1)) + (\rho-\rho(x))(1+o(1)) - \delta_x &< 0 \qquad (7.7)
\end{aligned}$$

We now show that (7.7) holds for large enough x and ρ. Note that $\rho - \rho(x) = o(1/x)$. Let $\gamma > 0$ such that there exists an x_γ large enough that both $(1+o(1)) < 2$ and $(\rho-\rho(x))2 - \delta_x < -\gamma$. We can find such a γ since $\delta_x = \lambda m_2(x)/x$. Choose $\rho_\gamma < 1$ large enough that $2(1-\rho_\gamma) - \gamma < 0$. Thus, P treats x_γ unfairly under load ρ_γ, which completes the proof in the case of an unbounded service distribution.

Note that in the case of a service distribution with upper bound x_U, we can plug $x = x_U$ into (7.6) and obtain

$$2(1-\rho) - (1-\rho+\delta_{x_U}) = 1 - \rho + \frac{\lambda E[X^2]}{x_U} < 0$$

which holds for large enough ρ.
□

7.2 Proportional fairness to large jobs

In the previous section, we began our study of proportional fairness by studying proportional fairness in *expectation* across *all job sizes*. We now move from a discussion of proportional fairness with respect to all jobs sizes to a discussion about only *large job sizes*. By focusing on only the behavior of large job sizes, we will be able to study the *distributional behavior* of proportional unfairness instead of being limited to studying proportional fairness in expectation. Characterizing the distribution of proportional fairness experienced by large job sizes is especially important because it has often been cited that the superior performance of scheduling policies that bias towards small jobs may come at the cost of starving large jobs, resulting in both larger and *more variable* response times [30, 215, 223, 210]. In addition to characterizing the fairness experienced by *large job sizes*, the results in this section are a necessary building block towards developing a framework for studying the distributional behavior of proportional fairness across *all job sizes*.

In fact, the results in this section provide theoretical justification for the generalized framework for studying proportional fairness presented in Section 7.3.

In order to study the distributional behavior of proportional unfairness we need to understand how to generalize the metric in Definition 7.1, $E[S(x)] = E[T(x)]/x$. There are many different possibilities, each with there own strengths and weaknesses. The most natural generalization of Definition 7.1 is to study the behavior of $S(x) = T(x)/x$. This matches the motivation for proportional fairness that response times should be proportional to job sizes. We study the behavior of $S(x)$ for large job sizes in detail in Section 7.2.1, where we prove that $S(x)^P$ converges almost surely as $x \to \infty$ under a wide range of common policies. In fact, we show that all work conserving scheduling policies have slowdown no worse than that of PS with respect to large jobs. In particular, we prove that the slowdown as job size tends to infinity under any work conserving policy is a.s. bounded by $1/(1-\rho)$; even under policies that clearly bias against large jobs.

Though using $S(x)$ to characterize the distributional behavior of proportional fairness is natural, the fact that $S(x)$ converges almost surely as $x \to \infty$ hints that other metrics with weaker scaling factors may provide more information about the distribution of proportional fairness. Specifically, the normalization factor $1/x$ in $S(x)$ hides information about the variability of the response times of large job sizes; thus it is important to consider other scaling factors. In Section 7.2.2, we illustrate that cumulant moments[6] provide a useful characterization of the the limiting response times of large job sizes. Further, we prove results that parallel the results attained when studying $S(x)$ as $x \to \infty$: we show that all work conserving policies have smaller asymptotic cumulant moments than PS.

7.2.1 Asymptotic behavior of slowdown

In order to study the distributional behavior of proportional fairness, the most natural metric to begin with is the slowdown for a job of size x, $S(x) = T(x)/x$. Slowdown clearly captures the idea that the response time of a job should be proportional to the size of the job, in addition it is a simple enough metric to allow the analysis of a wide range of scheduling policies. Further, as we saw in Chapter 6, the asymptotic behavior of slowdown is important for the study of the tail behavior of T in addition to its importance as a fairness metric.

In this section, we will show that many preemptive *work conserving* scheduling policies have the same performance as PS with respect to large jobs. In particular, we show that the slowdown as job size tends to infinity under any work conserving policy is at most $\frac{1}{1-\rho}$; even for policies that clearly bias against large jobs and that this limit is attained under many common preemptive policies. We will first prove the bound on all work conserving policies, and then we will analyze the performance of a range of individual policies, scheduling techniques, and scheduling heuristics.

Theorem 7.17

In a M/GI/1 with $E[X^2] < \infty$, under any work conserving policy P it holds a.s. (assuming the limit exists) that

$$\lim_{x\to\infty} S(x)^P \leq \frac{1}{1-\rho}.$$

If the policy is also non-preemptive, then the limit does exists and $S(x)^P \to 1$ a.s. as $x \to \infty$.

[6] For a brief overview of cumulants see Section 2.3.1.

Proof. The proof for *non-preemptive*, work conserving policies is quick. Start with the observation that

$$P(S(x)^P \geq 1) \quad = 1 \qquad \forall x, \forall \text{ policies } P$$

This follows simply by definition of slowdown. Thus, by taking limits, a.s. it holds that

$$\liminf_{x\to\infty} S(x)^P \geq 1, \forall \text{ policies } P$$

Further, we can upper bound the response time $T(x)$ under any non-preemptive policy with

$$T(x)^P \leq_{st} x + B(Q)$$

where Q is the work in the system. Thus, we have a.s. that

$$S(x)^P \quad \leq \quad 1 + \frac{B(Q)}{x} \quad \forall x, \forall \text{work conserving, non-preemptive policies } P$$

Taking limits we have a.s. that

$$\limsup_{x\to\infty} S(x)^P \leq 1, \forall \text{work conserving, non-preemptive policies P}$$

It follows that for all work conserving, non-preemptive policies P the limit does exists and

$$S(x)^P \to 1 \text{a.s. as } x \to \infty.$$

The remainder of the proof will concentrate on work conserving policies that allow preemption. We know that a.s.

$$T(x) \leq B(x+Q).$$

Thus

$$\lim_{x\to\infty} T(x)/x \leq \lim_{x\to\infty} \frac{B(x+Q)}{x}.$$

We will complete the proof by showing that

$$\lim_{x\to\infty} \frac{B(x+Q)}{x} = \frac{1}{1-\rho} \text{ a.s.} \tag{7.8}$$

If we let $\{B_i : i \geq 1\}$ denote an i.i.d. sequence of regular busy periods (non-exceptional), then $B(x)$ can be expressed as

$$B(x) = x + \sum_{i=1}^{N(x)} B_i$$

where $\{N(x) : x \geq 1\}$ is a Poisson process of rate λ independent of $\{B_i : i \geq 1\}$. We conclude that this version of $\{B(x) : x \geq 0\}$ is a compound Poisson process with a linear x term added on, so it has stationary

and independent increments. Thus, almost surely,

$$\lim_{x\to\infty} \frac{B(x)}{x} = E[B(1)] \quad \text{(by S.L.L.N)}$$
$$= \frac{1}{1-\rho}$$

Finally, notice that replacing x by $x + Q$ does not change this limit.
□

Remark 7.1 *Theorem 7.17 can be easily extended to the GI/GI/1 setting. However, Theorem 7.17 does not extend to policies that are not work conserving. In fact, for every $z \in [1, \infty)$ there is a non work conserving policy such that $\lim_{x\to\infty} S(x) = z$. To see this, consider the policy that makes each job wait $(z-1)x$ time before it is allowed to enter the queue of a non-preemptive, work conserving system.*

We now prove that the upper bound on $S(x)$ as $x \to \infty$ under all work conserving policies matches the limit under PS. Thus, no work conserving policy can treat large job sizes too much worse than PS does. Since PS is typically taken as a benchmark for fairness, this means that one never need to worry too much about the behavior of very large jobs. However, as we saw in Section 7.1, there is often some range of large (but not the largest) jobs that are treated unfairly under policies that bias towards small jobs.

Theorem 7.18
In a GI/GI/1 queue, $S(x)^{PS} \to 1/(1-\rho)$ a.s. as $x \to \infty$.

Proof. Define $G(t)$ to be the service given in time t to a permanent customer arriving to a stationary queue at time zero. Note that $P(G(t) > x) = P(T(x) > t)$. Because the PS queue is stationary, we know that $(t - G(t))/t$ converges to ρ a.s. as $t \to \infty$. That is, the amount of fraction service given to non-permanent customers must converge to ρ, otherwise the system would not be stable. Thus, we have that $G(t)/t \to 1-\rho$ almost surely, or equivalently, $S(x) = T(x)/x \to 1/(1-\rho)$ a.s.
□

We complete this section by proving that many other preemptive policies have $S(x)$ that converges to $1/(1-\rho)$ as $x \to \infty$. In fact, almost all common preemptive policies have the same performance with respect to limiting slowdown.

Theorem 7.19
In a GI/GI/1 queue, for $P \in \{\mathsf{SMART}, \mathsf{FB}, \mathsf{PLCFS}\}$, $S(x)^P \to 1/(1-\rho)$ a.s. as $x \to \infty$.

Proof. Note that Theorem 7.17 gives an upper bound, so all we need to show is a lower bound.

To prove the lower bound, we first derive a stochastic lower bound for the sojourn time of $\mathsf{P} = \mathsf{FB}$ and $\mathsf{P} \in \mathsf{SMART}$ in terms of a single busy period. For FB we have

$$T(x)^{\mathsf{FB}} \geq_{st} \widetilde{B_x}(x) \geq_{st} B_x(x) \geq_{st} B_{\epsilon x}((1-\epsilon)x), \qquad 0 < \epsilon < 1. \tag{7.9}$$

Furthermore, for $\mathsf{P} \in \mathsf{SMART}$, the Bias property guarantees that until the tagged job has received $(1-\epsilon)x$ units of service, all arriving jobs smaller than ϵx receive priority. Hence,

$$T(x)^{\mathsf{P}} \geq_{st} B_{\epsilon x}((1-\epsilon)x), \qquad 0 < \epsilon < 1. \tag{7.10}$$

To understand the length of this busy period, we will analyze a PLCFS system. Define $G_y(t)$ to be the service given in time t to a permanent customer arriving in an empty queue at time 0 when the generic service time is $XI_{[X<y]}$. Denoting the inverse of B_y by B_y^{-1}, we have for all x and t,

$$P(G_y(t) > x) = P(B_y(x) < t) = P(B_y^{-1}(t) > x).$$

Hence, G_y is stochastically equal to B_y^{-1}, so that

$$\lim_{x\to\infty} \frac{B_y(x)}{x} = \lim_{x\to\infty} \frac{x}{B_y^{-1}(x)} = \lim_{x\to\infty} \frac{x}{G_y(x)} \qquad a.s.$$

Note that this also holds if y is a function of x. Furthermore, define $G(t) = \lim_{y\to\infty} G_y(t)$. Then

$$\lim_{t\to\infty} \frac{G(t)}{t} = 1-\rho \text{ a.s.} \tag{7.11}$$

Set $z = (1-\epsilon)x$. From (7.9), (7.10) and (7.11), it follows that for all $0 < \varepsilon < 1$,

$$\begin{aligned}
\lim_{x\to\infty} \frac{T(x)^{\mathsf{P}}}{x} &\geq \lim_{x\to\infty} \frac{B_{\varepsilon x}((1-\varepsilon)x)}{x} \\
&= (1-\varepsilon) \lim_{z\to\infty} \frac{B_{\varepsilon z/(1-\varepsilon)}(z)}{z} \\
&= (1-\varepsilon) \lim_{z\to\infty} \frac{z}{G_{\varepsilon z/(1-\varepsilon)}(z)} \\
&= \frac{1-\varepsilon}{1-\rho} \qquad a.s.
\end{aligned}$$

The final equality follows because for any constant c, there exists a $z(c)$ such that for all $z > z(c)$, $G_c(z) \leq_{st} G_{\epsilon z/(1-\epsilon)}(z) \leq_{st} G(z)$. This completes the proof of the lower bound.
□

Before we move away from studying the convergence of slowdown, it is important to point out that for every $z \in [1, \frac{1}{1-\rho}]$ there is a work conserving policy such that $S(x) \to z$, a.s. as $x \to \infty$.

To see this, consider the Jump-To-Front (JTF) policy. JTF is linear combination of FCFS and PLCFS. More specifically, under JTF, with probability q an arriving job preempts the job being serviced and with probability $1-q$ an arriving job is placed at the back of a FCFS queue to await service.

We can quickly analyze this policy to find $S(x)^P$. Consider an arrival that gets placed at the front of the queue. This arrival can only be bothered by other jobs that are allowed to preempt. Thus, for this job $T(x) = B(x)|_{\lambda'}$, where $\lambda' = q\lambda$ for $q \in [0,1]$. That is, $T(x)$ is the length of a busy period started by a job of size x where the arrival rate is λ'.

Now consider a job that gets placed in the back of the queue. If the system is idle when the job arrives, we again see that $T(x) = B(x)|_{\lambda'}$. However, if the system is busy at the time of the arrival $T(x) = B(x + Q|busy))|_{\lambda'}$, where Q is the amount of work in system seen by an arbitrary arrival, and $Q|busy$ is the work seen by an arrival which finds the system busy. As in the analysis above, the effect of Q disappears in the limit.

Let $\rho' = \lambda' E[X]$. Then, putting these three pieces together, we see that $S(x)^P \to \frac{1}{1-\rho'}$ a.s. Since ρ' is an arbitrary number in $[0, \rho]$, we can make $\frac{1}{1-\rho'}$ any number in $[1, \frac{1}{1-\rho}]$.

7.2.2 Scaling response times

The fact that $S(x)$ converges almost surely as $x \to \infty$ provides an interesting perspective on the unfairness experienced by large job sizes, but it provides very little information about the variability of $T(x)$ as $x \to \infty$. In order to characterize the variability of response times experienced by large job sizes, we need to consider weaker scalings of $T(x)$ than $S(x) = T(x)/x$. In this section we will contrast the behavior of common scalings of $T(x)$ as $x \to \infty$ in order to illustrate that cumulant moments are unique in the sense that they have a scaling factor that retains information about the variability of the limiting distribution of $T(x)$ as $x \to \infty$.

In order to illustrate the issues in finding an appropriate scaling factor for the limit as $x \to \infty$, we will begin by looking at the asymptotic behavior of a busy period started by a job of size x, $B(x)$. Busy periods are fundamental to the analysis of many size based scheduling policies, and we will find that the correct scaling factor for $B(x)$ will match the scaling necessary for response times under many policies. Recall that the Laplace transform of $B(x)$, $\mathcal{L}_{B(x)}(s)$, is:

$$\mathcal{L}_{B(x)}(s) = e^{-x(s+\lambda-\lambda\mathcal{L}_B(s))}$$

where $\mathcal{L}_B(s)$ is the Laplace transform of a standard M/GI/1 busy period.

The most natural possibility for an appropriate metric for studying $B(x)$ as $x \to \infty$ is to scale the raw moments of $B(x)$. We can calculate the moments of $B(x)$ using $h(s) = -x(s + \lambda - \lambda\mathcal{L}_B(s))$. Thus, $h'(0) = -\frac{x}{1-\rho}$ and $h^{(i)}(0) = (-1)^i \lambda x E[B^i]$ for $i > 1$. It is important to notice that in each of these terms, x has degree one since $E[B^i]$ does not depend on x. Thus, we can determine the growth of $E[T(x)^i]$ as $x \to \infty$:

$$\begin{aligned} E[B(x)] &= \frac{x}{1-\rho} \\ E[B(x)^2] &= \left(\frac{x}{1-\rho}\right)^2 + \lambda x E[B^i] \\ E[B(x)^i] &= \left(\frac{x}{1-\rho}\right)^i + o(x^i) \end{aligned}$$

So, we must scale the ith raw moment by x^i in order to obtain a limit. This is equivalent to considering the slowdown of $B(x)$ as done in Section 7.2.1, and thus is too heavy handed for the current purpose since it hides the behavior of the higher moments of $B(x)$. Specifically, normalizing by x^i leads to a degenerate limiting distribution: $\lim_{x\to\infty} Var[B(x)]/x^2 = 0$.

Another natural suggestion for an appropriate scaling factor is to consider the central moments of $B(x)$, $E[(B(x)-E[B(x)])^i]$. Up until the third central moment, it seems that central moments can be scaled using a linear factor:

$$\begin{aligned} E[(B(x) - E[B(x)])^2] &= \lambda x E[B^2] \\ E[(B(x) - E[B(x)])^3] &= \lambda x E[B^3] \end{aligned}$$

However, beyond the third central moment the scaling becomes more convoluted and it becomes apparent that there is no simple scaling factor for the central moments that will capture the complete behavior of the higher moments. That is, any scaling factor will hide the effect of lower order variability terms, e.g.

$$E[(B(X) - E[B(X)])^4] = \lambda x \left(E[B^4]\right) + 3(\lambda x E[B^2])^2$$

The observation that the first three central moments are well behaved is important however. It hints that cumulants might provide the correct asymptotic metric. Define $\kappa_i[Y]$ as the ith cumulant of X and $\mathcal{K}_X(s) = \log(\mathcal{L}_X(s))$ as the cumulant generating function of X. For a brief overview of cumulants see Section 2.3.1.

In contrast to raw and central moments, the cumulants of $B(x)$ have a very simple form.

$$\mathcal{K}_{B(x)}(s) = \log(\mathcal{L}_{B(x)}(s)) = -x(s + \lambda - \lambda \mathcal{L}_B(s))$$

Calculating the cumulant moments through differentiation:

$$\kappa_i[B(x)] = \begin{cases} x/(1-\rho) & \text{for } i = 1 \\ \lambda x E[B^i] & \text{for } i > 1 \end{cases}$$

Thus, using κ_i/x, it is possible to capture all the variability in the limiting distribution of response time. In fact, $\kappa_i[T(x)]/x$ is an appropriate metric across a wide range of scheduling policies.

Theorem 7.20
In an M/GI/1 queue, let $E[X^{i+1}] < \infty$. Under any work conserving policy P,

$$\lim_{x\to\infty} \frac{\kappa_i[T(x)]^P}{x} \leq \begin{cases} 1/(1-\rho) & \text{for } i = 1 \\ \lambda E[B^i] & \text{for } i > 1 \end{cases} \tag{7.12}$$

Equality holds for $P \in \{SMART, FB, PLCFS\}$. Further, under any non-preemptive work conserving policy P,

$$\lim_{x\to\infty} \frac{\kappa_i[T(x)]^P}{x} = \begin{cases} 1 & \text{for } i = 1 \\ 0 & \text{for } i > 1 \end{cases}$$

Proof. Let P be a work conserving policy and Q be the time average work in system. Then, $T(x)^P \leq B(x+Q)$ because $B(x+Q) = B(x) + B(Q)$ corresponds to the time it would take to finish all the work

in the system when x arrived in addition to all the arriving work while x is in the system. Thus, as $x \to \infty$

$$\begin{aligned}\mathcal{K}_{B(x+Q)}(s)/x &= \log(\mathcal{L}_{B(x+Q)}(s))/x \\ &= \log(\mathcal{L}_{B(x)}(s))/x + \log(\mathcal{L}_{B(Q)}(s))/x \\ &\to s + \lambda - \lambda\mathcal{L}_B(s)\end{aligned}$$

which yields $\lim_{x\to\infty} \kappa_1[B(x+Q)]/x = \frac{1}{1-\rho}$ and $\lim_{x\to\infty} \kappa_i[B(x+Q)]/x = \lambda E[B^i]$ for $i > 1$; from which the result follows.

To prove that equality holds for $P \in \{\mathsf{SMART}, FB, PLCFS\}$, we can use a sequence of straightforward calculations using the cumulant generating functions (c.g.f.) for each policy and the bounds for SMART. Normalizing the c.g.f. by x and letting $x \to \infty$ shows that the c.g.f. converges to the c.g.f of $B(x)$ in each case.

Finally, to prove the limit in the non-preemptive case, we note that $T(x)^P \leq x + B(Q)$ for any non-preemptive policy P. Thus, as $x \to \infty$

$$\mathcal{K}_{1+B(Q)/x}(s) = s + \log(\mathcal{L}_{B(Q)}(s))/x \to s$$

which yields $\lim_{x\to\infty} \kappa_1[B(x+Q)]/x \leq 1$. We can observe that, by definition, we also have that $\lim_{x\to\infty} \kappa_1[T(x)^P]/x \geq 1$. Further, for $i > 1$ differentiation yields $\lim_{x\to\infty} \kappa_i[T(x)^P]/x = 0$.
□

Now that we understand the behavior of $\kappa_i[T(x)]$ under a range of policies, it is important to compare this behavior to that of PS. In [240] it was conjectured that equality holds for the limit in (7.12) under PS. However, known asymptotics are only tight enough to show the convergence of the first and second cumulants. In particular, it is known that [256]: $E[T(x)^i]^{PS} = \frac{x^i}{(1-\rho)^i} + \frac{\lambda x^{i-1} E[X^2] i(i-1)}{2(1-\rho)^{i+1}} + o(x^{i-1})$, which proves the result for $\kappa_1[T(x)]^{PS}$ and $\kappa_2[T(x)]^{PS}$. However, information about higher cumulants is lost in the $o(x^{i-1})$ term.

We now present an algorithm for computing the asymptotic ith cumulant under PS and show that equality holds in (7.12) for at least the first 10 cumulants of PS.

Theorem 7.21
Consider an M/GI/1 queue. For positive integer $i \leq 10$, let $E[X^{i+1}] < \infty$. Then, $\kappa_i[T(x)]^{PS}/x \to \lambda E[B^i]$ as $x \to \infty$.

The moments of response time in an M/GI/1/PS queue have a complex form. Let $\alpha_0(x) = 1$, $\delta_0(x) = \delta_1(x) = 0$, and for $i \geq 1$

$$\alpha_i(x) = \left(\frac{x}{1-\rho}\right)^i - \delta_i(x) \tag{7.13}$$

$$\delta_i(x) = \frac{i}{(1-\rho)^i} \int_0^x (x-t)^{i-1} \overline{R}^{(i-1)*}(t)dt \tag{7.14}$$

where $\overline{R}^{n*}(t)$ is the n-fold convolution of the waiting time distribution in the M/GI/1/FCFS queue, $W^{FCFS} =$

W. Then, we can write the moments of $T(x)^{PS}$ recursively as follows [256]

$$E[T(x)^k]^{PS} = -\sum_{i=1}^{k} \binom{k}{i} (-1)^i E[T(x)^{k-i}]^{PS} \alpha_i(x) \tag{7.15}$$

where the $E[T(x)] = 1/(1-\rho)$.

To calculate the moments of the convolution of waiting time, $\mu_i[W^{n*}]$ we will make use of the additivity of the cumulants and then use (2.2) to calculate the moments of the convolution in terms of the moments of W:

$$\begin{aligned}
\mu_1[W^{n*}] &= n\kappa_1[W] \\
\mu_i[W^{n*}] &= n\left(\kappa_i[W] - \sum_{j=1}^{i-1} \binom{i-1}{j} \mu_j[W^{n*}]\kappa_{i-j}[W]\right) \qquad (7.16) \\
\kappa_1[W] &= \mu_1[W] \\
\kappa_i[W] &= \mu_i[W] - \sum_{j=1}^{i-1} \binom{i-1}{j} \mu_j[W]\kappa_{i-j}[W] \qquad (7.17)
\end{aligned}$$

Finally, we can use Takács recursive formula for the moments of W to finish the calculation [221]:

$$\begin{aligned}
\mu_0[W] &= 1 \\
\mu_i[W] &= \frac{\lambda}{1-\rho} \sum_{j=1}^{i} \binom{i}{j} \frac{E[X^{j+1}]}{j+1} \mu_{i-j}[W] \qquad (7.18)
\end{aligned}$$

Proof of Theorem 7.21.

We will present an algorithm for the computation of $\lim_{x\to\infty} \kappa_i[T(x)]^{PS}/x$ and illustrate the computation for $i \leq 4$. We have carried out the calculations for $i \leq 10$ using Mathematica.

We begin by using (7.13), (7.14), (7.15), and (2.2) to derive formulas for $\kappa_i[T(x)]^{PS}$ in terms of $\delta_n(x)$.

$$\begin{aligned}
\kappa_1[T(x)]^{PS} &= \frac{x}{1-\rho} \qquad (7.19) \\
\kappa_2[T(x)]^{PS} &= \delta_2(x) \qquad (7.20) \\
\kappa_3[T(x)]^{PS} &= \frac{3x\delta_2(x)}{1-\rho} - \delta_3(x) \qquad (7.21) \\
\kappa_4[T(x)]^{PS} &= \frac{6x^2\delta_2(x)}{(1-\rho)^2} - \frac{4x\delta_3(x)}{1-\rho} + 3\delta_2(x)^2 + \delta_4(x) \qquad (7.22)
\end{aligned}$$

To see that $\kappa_i[T(x)]^{PS}/x \to \lambda E[B^i]$ as $x \to \infty$, we need to understand the asymptotic behavior of

$\delta_n(x)$. Note that we can rewrite $\delta_n(x)$ as

$$\delta_n(x) = \frac{n}{(1-\rho)^n} \sum_{i=0}^{n-1} \binom{n-1}{i} (-1)^{n-1-i} x^i \int_0^x t^{n-1-i} \overline{R}^{(n-1)*}(t) dt \tag{7.23}$$

Further, $\int_0^x t^{n-1-i} \overline{R}^{(n-1)*}(t) = \mu_{n-i}[W_x^{(n-1)*}]/(n-i)$ where $W_x = \min(W, x)$.

We now combine (7.23) with (7.19) - (7.22) and note that $W_x \to W$ as $x \to \infty$ to obtain the limit of $\kappa_i[T(x)]^{PS}/x$ in terms of $\kappa_i[W]$.

$$\begin{aligned}
\lim_{x\to\infty} \frac{\kappa_1[T(x)]^{PS}}{x} &= \frac{1}{1-\rho} \\
\lim_{x\to\infty} \frac{\kappa_2[T(x)]^{PS}}{x} &= \frac{2\kappa_1[W]}{(1-\rho)^2} \\
\lim_{x\to\infty} \frac{\kappa_3[T(x)]^{PS}}{x} &= \frac{9\kappa_1[W]^2 + 3\kappa_2[W]}{(1-\rho)^3} \\
\lim_{x\to\infty} \frac{\kappa_4[T(x)]^{PS}}{x} &= \frac{64\kappa_1[W]^3 + 48\kappa_1[W]\kappa_2[W] + 4\kappa_3[W]}{(1-\rho)^4}
\end{aligned}$$

To complete the proof, we calculate the cumulants of W using (7.17) and (7.18) and derive the final expressions for the limits.

$$\begin{aligned}
\lim_{x\to\infty} \frac{\kappa_1[T(x)]^{PS}}{x} &= \frac{1}{1-\rho} \\
\lim_{x\to\infty} \frac{\kappa_2[T(x)]^{PS}}{x} &= \frac{\lambda E[X^2]}{2(1-\rho)^3} \\
\lim_{x\to\infty} \frac{\kappa_3[T(x)]^{PS}}{x} &= \frac{\lambda E[X^3]}{(1-\rho)^4} + \frac{3\lambda^2 E[X^2]^2}{(1-\rho)^5} \\
\lim_{x\to\infty} \frac{\kappa_4[T(x)]^{PS}}{x} &= \frac{\lambda E[X^4]}{(1-\rho)^5} + \frac{10\lambda^2 E[X^2]E[X^3]}{(1-\rho)^6} + \frac{15\lambda^3 E[X^2]^3}{(1-\rho)^7}
\end{aligned}$$

Then, noting that the busy period moments can be derived from the Laplace transform $\widetilde{B}(s) = \widetilde{X}(s + \lambda - \lambda\widetilde{B}(s))$ or more efficiently using an algorithm such as [70], we can verify that the limits indeed match $\lambda E[B^i]$.

□

7.3 A unified framework for proportional fairness

As we have seen, providing proportional fairness is an important design constraint under a variety of applications where users desire small mean response times, but also want to be treated "fairly," e.g. web servers and routers. In many of these settings, system designers are hesitant to use policies that provide small mean

response times by prioritizing small job sizes (at the expense of large job sizes) due to worries that large job sizes will be starved of service. Specifically, there are worries that large job sizes will receive disproportionately large response times and that large job sizes will receive disproportionately variable response times. Thus, it is important to study both proportional fairness in expectation and the distributional behavior of proportional fairness.

In this section, we will extend the approach used to study proportional fairness in expectation (Section 7.1) using the results characterizing the distributional behavior of the response times of large jobs (Section 7.2) in order to motivate a framework of metrics for studying the distributional behavior of proportional fairness. We will accomplish this by generalizing Definition 7.1 to higher moments using the cumulant moments of $T(x)$ as follows. For a brief overview of cumulants see Section 2.3.1. Recall that B is the length of a busy period.

Definition 7.3 *Let $0 < \rho < 1$ and $E[X^i] < \infty$ in an M/GI/1. A job size x is treated **fairly** under policy P, service distribution X, and load ρ if*

$$\frac{\kappa_i[T(x)]^P}{x} \leq 1_{[i=1]} + \lambda E[B^i]$$

*Otherwise a job size x is treated **unfairly**. A scheduling policy P is **fair** if every job size is treated fairly. Otherwise P is **unfair**.*

The above definition was first introduced by Wierman and Harchol-Balter in [240]. It is worth pointing out that, though the $1_{[i=1]}$ may appear strange at first, it is a fundamental result of the fact that the first cumulant is shift-equivariant while all others are shift-invariant: letting c be a constant, $\kappa_1[Y + c] = \kappa_1[Y] + c$ but for $i \geq 2$, $\kappa_i[Y + c] = \kappa_i[Y]$.

Notice that there are many parallels between Definition 7.1, for studying proportional fairness in expectation, and Definition 7.3. Both definitions have two pieces: a metric and a criterion. Further, the metric and criterion in Definition 7.1 match the metric and criterion for $i = 1$ in Definition 7.3: $\kappa_1[T(x)]/x = E[T(x)]/x$ and $1 + \lambda E[B] = 1/(1 - \rho)$. Additionally, the motivation for the metric and criterion parallel the motivations for the metric $E[T(x)]/x$ and the criterion $1/(1 - \rho)$ in Definition 7.1.

In both Definition 7.1 and Definition 7.3, the *metric* is motivated by the behavior of jobs of size $x \to \infty$. Specifically, the metric must scale moments of $T(x)$ appropriately to allow for comparison of moments of $T(x)$ between small and large x. For $E[T(x)]^P$, it is clear that $1/x$ is an appropriate scaling factor because $E[T(x)]^P = \Theta(x)$ under all work conserving scheduling policies, and thus we need to normalize by the growth rate. For higher moments of $T(x)$, the correct scaling factor is not obvious; however in Section 7.2 we illustrated that $\kappa_i[T(x)]$ is $\Theta(x)$ for common preemptive policies and $O(x)$ for all work conserving policies (Theorems 7.20 and 7.21). Hence, scaling by $1/x$ makes sense; whereas using a stronger scaling would hide the variability in the distribution of $T(x)$ as $x \to \infty$.

The motivation for the criteria in Definitions 7.1 and 7.3 is more involved. The criterion $1/(1 - \rho)$ in Definition 7.1 stems from two formal motivations. First, it provides a min-max notion of fairness: $\min_P \max_x E[T(x)]^P/x = 1/(1 - \rho)$. Second, the criterion $1/(1 - \rho)$ has the property that $\lim_{x\to\infty} E[T(x)]^P/x = 1/(1-\rho)$. This property is key to the derivation a broad classification of scheduling policies as one of Always Fair, Sometimes Fair, or Always Unfair (see Section 7.1). The classification is perhaps the strongest motivation for the criterion $1/(1 - \rho)$ because it illustrates that the criterion distinguishes between patterns of behavior of policies with respect to the metric $E[T(x)]^P/x$.

The motivation for the criterion in Definition 7.3 parallels that for the criterion in Definition 7.1; however it is not as cut-and-dry. Just as the criterion $1/(1-\rho)$ used in Definition 7.1 has the property that $\lim_{x\to\infty} E[T(x)]^P/x = 1/(1-\rho)$ under many common policies, Theorems 7.20 and 7.21 illustrate that the criterion in Definition 7.3 also serves as the limit for $\kappa_i[T(x)]/x$ under many common scheduling policies. However, a priori, it is not clear whether this limiting behavior distinguishes between patterns of behaviors with respect to $\kappa_i[T(x)]$ in the same way $1/(1-\rho)$ did for $E[T(x)]/x$. All we can do to provide this justification is to illustrate that in the case of $i = 2$, when $\kappa_2[T(x)]/x = Var[T(x)]/x$, the criterion $\lambda E[B^2]$ does indeed differentiate between contrasting $Var[T(x)]^P/x$ behaviors. This is what we will do in Section 7.4. Specifically, we will illustrate that Definition 7.3 distinguishes between between non-monotonic "hump" behaviors – where some mid-range job sizes are treated the most unfairly – and monotonically increasing behaviors in the case of $i = 2$ just as it does in the case of $i = 1$. Further, the classifications that result from Definition 7.3 in the cases of $i = 1$ and $i = 2$ parallel each other; and thus we conjecture that for $i > 2$ similar classifications will emerge. However, deriving classifications for these higher order moments will be a difficult task.

7.4 Predictability

In this section, we will provide an illustration of the generalized framework for studying proportional fairness presented in Definition 7.3. We specifically consider the case of $\kappa_2[T(x)]/x = Var[T(x)]/x$. In studying the behavior of $Var[T(x)]$ across x, we are characterizing the "predictability" of response times, which is an important metric in its own right. In fact, in many modern computer systems improving the predictability of response times is of fundamental importance: it can be even more urgent than improving the response times on average. This is because users expect certain response times based on past experience and become frustrated if they must wait longer than expected. So, an important goal for a scheduling policy is to provide identical jobs nearly identical response times.

Though there has been a significant amount of prior literature *deriving* $Var[T(x)]$ and higher moments of $T(x)$ under many common policies [222, 253, 119, 120]; little work has studied the *behavior* of $Var[T(x)]$ and higher moments of $T(x)$ across x, possibly due to the complicated nature of the derived formulas. Understanding the behavior of $T(x)$ beyond the mean is key to many applications where users know the size of the job they are submitting and would like to minimize the difference between their *experienced* response time, $T(x)$, and their *expected* response time, $E[T(x)]$; thus maximizing "predictability." Reducing "unpredictability" in response times can be more important to users than reducing the response times themselves because waiting much longer than expected causes far more user frustration than simply waiting longer on average [65, 255]. Note that higher moments of $T(x)$ provide a better measure of user-perceived "predictability" than do higher moments of T in the situation where the size of the job is known by the user. Further, many QoS guarantees are of the form "90% of the time a job of size x will have response time $< g(x)$," for some function $g(\cdot)$. Such guarantees can be phrased as bounding higher moments of $T(x)$ by applying tail inequalities such as Chebyshev's Inequality.

7.4.1 Defining predictability

The notion of predictability that we define in this section is a special case of the framework for proportional fairness in Definition 7.3 that we have developed in this chapter; thus we have already provided motivation for the metric and criteria in this definition in Section 7.3. Specifically, we consider the case of $\kappa_2[T(x)]/x = Var[T(x)]/x$ and introduce the following definition:

Definition 7.4 *Let $0 < \rho < 1$ and $E[X^2] < \infty$ in an M/GI/1 system. A job size x is treated **predictably** under policy P, service distribution X, and load ρ if*

$$\frac{Var[T(x)]^P}{x} \leq \lambda E[B^2] = \frac{\lambda E[X^2]}{(1-\rho)^3}$$

*Otherwise a job size x is treated **unpredictably**. A scheduling policy P is **predictable** if every job size is treated predictably. Otherwise P is **unpredictable**.*

As with proportional fairness in expectation in Section 7.1, we build on Definition 7.4 to develop a classification of predictability.

Definition 7.5 *Let $0 < \rho < 1$ and $E[X^2] < \infty$ in an M/GI/1 queue where X is non-deterministic.[7] A scheduling policy P is: (i) **Always Predictable** if P is predictable for all such ρ and X; (ii) **Sometimes Predictable** if P is predictable under some ρ and X and unpredictable under other ρ and X; or (iii) **Always Unpredictable** if P is unpredictable under all loads and service distributions.*

The above definitions were introduced by Wierman and Harchol-Balter in [240].

Remark 7.2 *In addition to the fact that Definition 7.4 is a special case of the generalized framework in Definition 7.3, the definition of predictability is also motivated by the task of providing QoS guarantees. Many QoS guarantees take the form "90% of the time $T(x) - E[T(x)] < g(x)$," or equivalently $P(T(x) - E[T(x)] \geq g(x)) \leq 10\%$. Chebyshev's Inequality [193] gives us a bound of the form*

$$P(T(x)^P - E[T(x)]^P \geq g(x))) \leq \frac{Var[T(x)]^P}{g(x)^2} \tag{7.24}$$

Thus, we can provide the desired QoS guarantee by ensuring that $Var[T(x)]/g(x)^2$ is not too large.[8] Looking more closely at (7.24), to determine an appropriate metric for predictability, we need to ask "what is the smallest value of $g(x)$ that allows $Var[T(x)]/g(x)^2$ to be bounded by a constant (10% in the above example) for all x?"

Suppose that $g(x) = kx^i$ for some k independent of x and some constant i. Then, we need to choose the smallest i that allows $Var[T(x)]/g(x)^2$ to be bounded by a constant. Notice that we can immediately rule out $i > 1$ because $T(x)^P$ and $E[T(x)]^P$ grow linearly in x for all P; thus it does not make sense to

[7] We exclude deterministic distributions because the concept of proportional fairness is only interesting when there exist jobs of different sizes.

[8] Note that a more complex bound including other information about the distribution of $T(x)$ could be used to provide QoS guarantees in practice. However, the simple calculation of (7.24) provides intuition for an appropriate metric with which to study $Var[T(x)]$.

bound $T(x)^P - E[T(x)]^P$ by something growing super-linearly. We can also rule out $i < 1/2$ because for such i, $Var[T(x)]^P/x^{2i} \to \infty$ as $x \to \infty$ under all P. This leaves $i \in [1/2, 1]$, where $i = 1/2$ is the most desirable because it provides the tightest bound on $T(x) - E[T(x)]$ as x grows.

Definition 7.4 uses the metric $Var[T(x)]^P/x$, which corresponds to choosing $i = 1/2$. This choice makes sense because $Var[T(x)]^P/x$ is $O(1)$ under all work conserving policies P. Thus, any policy that is predictable will allow a QoS bound that is constant across x. Note that choosing $i \in (1/2, 1]$ is also reasonable; however the results are less interesting.[9]

In the remainder of this section we will classify individual policies, scheduling techniques, and scheduling heuristics as one of Always Predictable (Section 7.4.3.1), Always Unpredictable (Section 7.4.3.2), or Sometimes Predictable (Section 7.4.3.3). This classification is illustrated in Figure 7.3. Interestingly, the classification of predictability has many parallels to the classification of proportional fairness (see Figure 7.1). For instance, PS and PLCFS are both Always Fair and Always Predictable. Similarly, SRPT is both Sometimes Fair and Sometimes Predictable and exhibits the same interesting non-monotonic (hump shaped) behavior under both measures. In fact, the almost all technique-based and heuristic-based classifications receive exhibit parallel behavior under the two measures, the only exception being non-preemptive non-size based policies.

In classifying scheduling policies with respect to predictability, we find that $Var[T(x)]^P/x$ can exhibit four different patterns of functional behavior (see Figure 7.4). Some policies, e.g. PS, have $Var[T(x)]^P/x$ that grows monotonically and is bounded by a constant across x; whereas other policies, e.g. FCFS, have $Var[T(x)]^P$ that decreases monotonically in x and is unbounded as $x \to 0$. Further, it seems that prioritization, be it age based, size based or remaining size based, leads to non-monotonic behavior in normalized conditional response times. In particular, under PSJF, FB, and SRPT mid-range job sizes have the largest $Var[T(x)]^P/x$. Further, SJF has a similar hump behavior for mid-range jobs; however the smallest job sizes still receive unbounded $Var[T(x)]^P/x$. Our work illustrates that the criterion $\lambda E[X^2]/(1-\rho)^3$ in Definition 7.4 for predictability distinguishes between these functional behaviors. If a policy has monotonically increasing, bounded $Var[T(x)]^P/x$ under some service distributions and loads then the policy is Always or Sometimes Predictable; otherwise the policy is Always Unpredictable because under all service distributions and loads either $Var[T(x)]^P/x$ is unbounded or some mid-range job sizes receive significantly worse $Var[T(x)]/x$ than other job sizes.

The parallels between the classifications of fairness and predictability beg the question of whether similar classifications exist for higher conditional moments. We conjecture that using the generalized framework in Definition 7.3 to study higher moments will lead to classifications that parallel the results presented here; however the task of deriving such classifications seems difficult.

7.4.2 The predictability of individual policies

To get a feel for Definition 7.4, it is important to begin by studying the behavior of common individual policies with respect to predictability.

[9] For $i \in (1/2, 1]$, $Var[T(x)]^P/x^{2i} \to 0$ as $x \to \infty$ under all P. As a result, it can quickly be seen that policies fall into one of two classes based the behavior of $Var[T(x)]^P$ as $x \to 0$, i.e whether $\lim_{x\to 0} Var[T(x)]^P/x^{2i} < \infty$. This makes intuitive sense because the bound on $T(x)^P - E[T(x)]^P$ is much looser as x grows and thus the performance of the small jobs dominates the QoS bound.

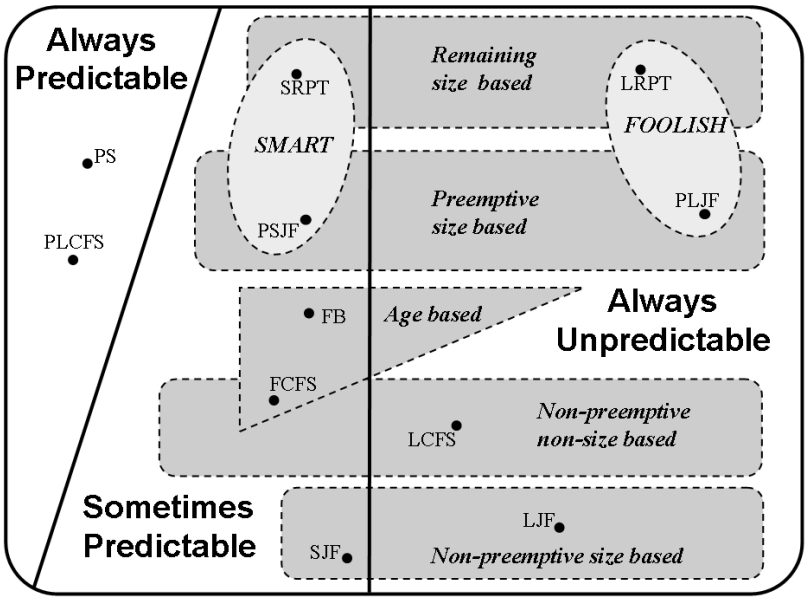


Figure 7.3: *An illustration of the classification of common prioritization techniques and heuristics with respect to predictability.*

As a first step, it is easy to see that

$$E[T(x)]^{PLCFS} = \frac{\lambda E[X^2]}{(1-\rho)^3}$$

Thus, PLCFS is Always Predictable. However, beyond PLCFS, the only other common policy that is Always Predictable is PS, which we will analyze in Section 7.4.2.1 below. Most policies fall into the Sometimes Predictable or Always Predictable classifications. In fact, as is illustrated in Figure 7.3, the behavior of many policies with respect to $Var[T(x)]/x$ mimics the behavior of $E[T(x)]/x$. We will illustrate this in this section using the examples of PS, PSJF, FB, and SRPT.

7.4.2.1 PS

To analyze the behavior of $Var[T(x)]^{PS}$, we begin with the following useful representation of $Var[T(x)]^{PS}$:

$$Var[T(x)]^{PS} = \frac{2}{(1-\rho)^2}\int_0^x (x-t)\overline{R}(t)dt$$

where $\overline{R}(t) = 1 - R(t)$ and $R(t) = (1-\rho)\sum_{n=0}^{\infty}\rho^n F^{*n}(t)$ with $F^{*n}(t) = \int_0^{\infty} F^{*(n-1)}(t-s)dF^{*1}(s)$, $F^{*1}(t) = \frac{1}{E[X]}\int_0^t (1-F(s))ds$, and $F^{*0}(t) = \begin{cases} 1, & x \geq 0 \\ 0, & x < 0 \end{cases}$.

The complexity of this formula has led to mainly asymptotic analysis of the conditional variance of PS. However, we can to exploit this asymptotic information in order to show that PS is predictable for all x.

Theorem 7.22
In an M/GI/1 queue with $E[X^2] < \infty$ PS is Always Predictable. Further, $Var[T(x)]^{PS}/x$ is strictly monotonically increasing in x.

Proof. We will prove the result by showing that $\frac{d}{dx}\left(Var[T(x)]^{PS}/x\right) > 0$ for all x. In combination with the fact that $Var[T(x)]^{PS}/x \to \lambda E[B^2]$, this will complete the proof.

$$\begin{aligned}\frac{d}{dx}\frac{Var[T(x)]^{PS}}{x} &= \frac{2}{(1-\rho)^2}\frac{d}{dx}\left(\int_0^x \overline{R}(t)dt - \frac{1}{x}\int_0^x t\overline{R}(t)dt\right) \\ &= \frac{2}{(1-\rho)^2}\left(\overline{R}(x) - \overline{R}(x) + \frac{1}{x^2}\int_0^x t\overline{R}(t)dt\right) > 0\end{aligned}$$

□

It is interesting that $Var[T(x)]^{PS}/x$ is monotonically increasing in x under all service distributions because this is different than $E[T(x)]^{PS}/x = 1/(1-\rho)$, which is constant across x. This illustrates why $Var[T(x)]^{PS}/x$ is not an appropriate criterion for a definition of predictability. It is important to point out that the predictability of PS has been studied in much more detail by Ward and Whitt [233]. While we assume no knowledge of the system state in order to study how well response times will match with prior user experience, Ward and Whitt study how well $T(x)^{PS}$ can be predicted given knowledge of the system state (e.g. the number of jobs in the system upon arrival, N). They look at the question analytically as $N \to \infty$ and $x \to \infty$ and prove that predictions can be made quite accurately when either x or N is large.

7.4.2.2 PSJF

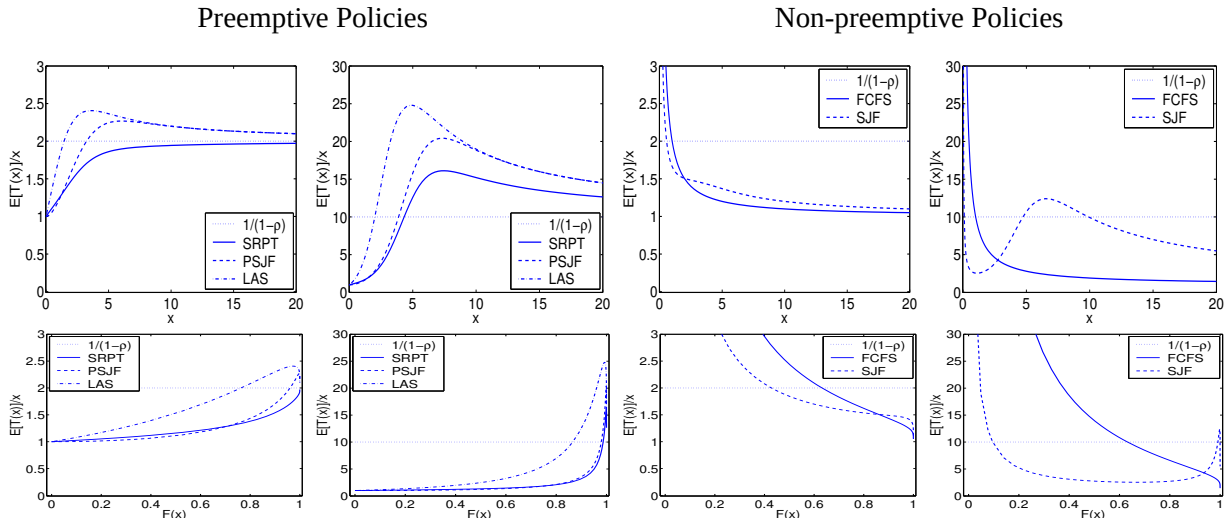
We now move to another important individual policy, Preemptive-Shortest-Job-First (PSJF). PSJF is the canonical example of a policy that prioritizes based on size, and it will serve as the building block for the analysis of all size based policies. PSJF significantly improves on the mean response time of PS, and has is near optimal with respect to mean response time as we saw in Theorem 4.2.

In this section, we will first prove that under distributions with $E[X^3] < \infty$ PSJF exhibits non-monotonic behavior in $Var[T(x)]^{PSJF}/x$, where mid-range job sizes are treated the most unpredictably. Then, we will bound the position and size of this "hump."

Theorem 7.23
In an M/GI/1 queue with $E[X^3] < \infty$, PSJF is unpredictable. Further, there exists some L such that all $x \geq L$ are treated unpredictably.

Note that the above result only discusses the case that $E[X^3] < \infty$. No published work has appeared for PSJF in the case that $E[X^3] = \infty$, but Brown has an unpublished manuscript in which he proves that PSJF can be predictable in this setting. Thus, we classify PSJF as Sometimes Predictable.

Proof. We will start by proving the result for the case when the service distribution has some finite upper bound L and then move to the case when the service distribution has no upper bound.



(a) Low load ($\rho = 0.5$) (b) High load ($\rho = 0.9$) (c) Low load ($\rho = 0.5$) (d) High load ($\rho = 0.9$)

Figure 7.4: *The conditional variance is illustrated under a variety of both preemptive and non-preemptive policies. The service distribution is exponential with mean 1. The dotted line shows the criteria for predictability.*

In the case where the service distribution has some finite upper bound L the result is immediate.

$$\begin{aligned} Var[T(L)]^{PSJF} &= \frac{\lambda L E[X^2]}{(1-\rho)^3} + \frac{\lambda E[X^3]}{3(1-\rho)^3} + \frac{3}{4}\left(\frac{\lambda E[X^2]}{(1-\rho)^2}\right)^2 \\ &> \frac{\lambda L E[X^2]}{(1-\rho)^3} \end{aligned}$$

The case of unbounded service distributions is more complicated. Observe that $Var[T(x)]^{PSJF}/x$ is increasing in x for small x. Also, recall that from Theorem 7.20 that $Var[T(x)]^{PSJF}/x \to \lambda x E[X^2]/(1-\rho)^3$ as $x \to \infty$. Hence, if we can show that the limit is approached from above, rather than below, we will have exhibited non-monotonic behavior. We accomplish this by showing that $\frac{d}{dx}\left(Var[T(x)]^{PSJF}/x\right)$ approaches 0 from below as $x \to \infty$. By observing that

$$\frac{d}{dx}\frac{Var[T(x)]^{PSJF}}{x} = \frac{x\frac{d}{dx}Var[T(x)]^{PSJF} - Var[T(x)]^{PSJF}}{x^2}$$

our goal reduces to showing that as $x \to \infty$

$$x\frac{d}{dx}Var[T(x)]^{PSJF} - Var[T(x)]^{PSJF} < 0 \tag{7.25}$$

Computation yields that for any distribution with finite third moment:

$$x\frac{d}{dx}Var[T(x)]^{PSJF} = \frac{\lambda x m_2(x)}{(1-\rho(x))^3} + O(x^4 f(x))$$

So,

$$\begin{aligned} x\frac{d}{dx}Var[T(x)]^{PSJF} - Var[T(x)]^{PSJF} &= \frac{\lambda x m_2(x)}{(1-\rho(x))^3} + O(x^4 f(x)) - Var[T(x)]^{PSJF} \\ &< 0 \text{ as } x \to \infty \end{aligned}$$

Thus, PSJF is unpredictable for all loads and all unbounded service distributions.
□

Although there are always some sizes that are treated unpredictably under PSJF, most sizes receive predictable response times.

Theorem 7.24
In an M/GI/1 queue, let K_1 be a constant such that $m_3(x) \leq K_1 x m_2(x)$. Then

$$Var[T(x)]^{PSJF} \leq Var[B(x)]h_1(\rho, x)^{PSJF}$$

where

$$h_1(\rho,x)^{PSJF} = \frac{(1-\rho)^3}{(1-\rho(x))^4}\left\{\left(1+\frac{K_1}{3}\right) + \left(\frac{5K_1}{12}-1\right)\rho(x)\right\}$$

Further, noting that $K_1 \leq 1$ for all service distributions, we have that $h_1(\rho,x) \leq \frac{(1-\rho)^3}{(1-\rho(x))^4}\left\{\frac{4}{3} - \frac{7}{12}\rho(x)\right\}$.

Proof. The proof follows from direct calculation.

$$\begin{aligned} Var[T(x)]^{PSJF} &\leq \frac{\lambda x m_2(x)}{(1-\rho(x))^3} + \frac{\lambda m_3(x)}{3(1-\rho(x))^3} + \frac{3\lambda^2 m_3(x) m_1(x)}{4(1-\rho(x))^4} \\ &\leq Var[B(x)]\frac{(1-\rho)^3}{(1-\rho(x))^4}\frac{m_2(x)}{E[X^2]}\left\{\left(1+\frac{K_1}{3}\right)(1-\rho(x)) + \frac{3K_1\rho(x)}{4}\right\} \\ &\leq Var[B(x)]\frac{(1-\rho)^3}{(1-\rho(x))^4}\left\{\left(1+\frac{K_1}{3}\right) + \left(\frac{5K_1}{12}-1\right)\rho(x)\right\} \end{aligned}$$

□

Notice that this bound guarantees that a large percentage of job sizes will be treated predictably. In particular, all job sizes such that $\rho(x) \leq 1 - \left(\frac{4}{3}(1-\rho)^3\right)^{1/4}$. For example, if the load is 0.8, all job sizes x such that $\rho(x) \leq 0.678$ will be treated predictably. If the job size distribution is highly variable, this is nearly all jobs (since a small percentage of the largest jobs make up half the load).

Example

Consider $X \sim Exp(1)$. Thus, $f(x) = e^{-x}$. Then, $\rho(x) = \rho(1 - e^{-x} - xe^{-x})$. So, $\rho(x) \leq 1 - \left(\frac{4}{3}(1-\rho)^3\right)^{1/4}$ when $e^{-x} + xe^{-x} \geq 1 - \frac{1-\left(\frac{4}{3}(1-\rho)^3\right)^{1/4}}{\rho}$. This says that when $\rho = 0.8$, PSJF will be predictable for at least jobs of size $x \leq 3.3$. Thus, PSJF will be predictable for at least 96.3% of the jobs.
□

Further, an even larger percentage of job sizes can be shown to be treated predictably if K_1 is bounded below 1 using Theorem 3.8. For instance, if $f(x)$ is decreasing, K_1 can be set to 3/4.

Theorem 7.24 shows that small (and in fact most) job sizes receive predictable service, but the question still remains as to how unpredictably the large jobs can be treated. The dependence of Theorem 7.24 on the bound $m_3(x) \leq K_1 x m_2(x)$ leads to an overestimate of $Var[T(x)]^{PSJF}$ for large job sizes. Thus, we must take a different approach in order to obtain a tighter bound for the large jobs.

Define $M_2[X] = \frac{E[X^2]}{E[X^2]}$ and $M_3[X] = \frac{E[X^3]}{E[X]E[X^2]}$.

Theorem 7.25

In an M/GI/1 queue with $E[X^3] < \infty$, for jobs of size $x > K_2 E[X]$

$$Var[T(x)]^{PSJF} \leq Var[B(x)]h_2(\rho)$$

where

$$h_2(\rho) = \left(1 + \frac{M_3[X]}{3K_2}\right) + \frac{3\rho M_2[X]}{4K_2(1-\rho)}$$

Proof. We again proceed with direct calculation on $Var[T(x)]$

$$\begin{aligned} Var[T(x)]^{PSJF} &\leq \frac{\lambda x E[X^2]}{(1-\rho)^3} + \frac{\lambda E[X^3]}{3(1-\rho)^3} + \frac{3\lambda^2 E[X^2]^2}{4(1-\rho)^4} \\ &\leq \frac{Var[B(x)]}{1-\rho}\left\{\left(1 + \frac{M_3[X]}{3K_2}\right)(1-\rho) + \frac{3\rho M_2[X]}{4K_2}\right\} \end{aligned}$$

□

The combination of the Theorems 7.24 and 7.25 provides a technique for determining both (i) which job sizes are treated unpredictably and (ii) how unpredictably they can be treated. We illustrate this process in the next example.

Example

Returning to the case of $X \sim Exp(1)$ we can use our prior calculation to set $K_2 = 3.3$ in the case where $\rho = 0.8$ in our PSJF system. Now, noting that $M_3[X] = 3$ and $M_2[X] = 2$ in the case of the exponential, we have $Var[T(x)]^{PSJF} \leq 3.1 Var[B(x)]$. Thus, although PSJF is Always Unpredictable, even in the case of an exponential service distribution with $\rho = 0.8$, PSJF is only unpredictable for at most 4% of jobs and this small fraction of jobs only receives a factor of 3.1 higher variance. This agrees with the behavior shown in Figure 7.4.
□

7.4.2.3 FB

We next turn to another important priority-based policy: FB. In this section, we will first prove that FB exhibits non-monotonic behavior in $Var[T(x)]^{FB}/x$, where large, but not the largest, job sizes are treated the most unpredictably. We will then bound the position and size of this "hump" through bounds on $Var[T(x)]^{FB}$.

Lemma 7.26
For all x, $Var[T(x)]^{PSJF} \leq Var[T(x)]^{FB}$

This Lemma is a special case of Theorem 3.19. Combining Lemma 7.26 with Theorem 7.23, we have:

Corollary 7.27
In an M/GI/1 with $E[X^3] < \infty$, FB is unpredictable. Further, there exists some L such that all $x > L$ are treated unpredictably.

Note that the above result only discusses the case when $E[X^3] < \infty$. As with PSJF, no published work has appeared for FB in the case that $E[X^3] = \infty$, but Brown has an unpublished manuscript in which he proves that FB can be predictable in this setting. Thus, we classify FB as Sometimes Predictable.

Now, let us return to the case of $E[X^3] < \infty$. Although there are always some job sizes that are treated unpredictably under FB when $E[X^3] < \infty$, most job sizes receive predictable response times.

Theorem 7.28
In an M/GI/1 queue, let K_1 be a constant such that $m_3(x) \leq K_1 x m_2(x)$. Then

$$Var[T(x)]^{FB} \leq Var[B(x)]h_1(\rho, x)^{FB}$$

where

$$h_1(\rho, x)^{FB} = \frac{(1-\rho)^3}{(1-\widetilde{\rho}(x))^4}\left\{\left(1+\frac{K_1}{3}\right)+\left(\frac{2K_1}{3}-1\right)\widetilde{\rho}(x)\right\}$$

Further, noting that $K_1 \leq 1$ for all service distributions we have that $h_1(\rho, x)^{FB} \leq \frac{(1-\rho)^3}{(1-\widetilde{\rho}(x))^4}\left\{\frac{4}{3}-\frac{1}{3}\widetilde{\rho}(x)\right\}$.

The proof follows using Lemmas 7.29 and 7.30 which are stated below.

Lemma 7.29
$\lambda^2\widetilde{m}_2(x)^2 \leq \frac{4}{3}\lambda\widetilde{m}_3(x)\widetilde{\rho}(x)$

Proof.

$$\begin{aligned}\lambda^2\widetilde{m}_2(x)^2 &\leq 4\lambda^2\left(\int_0^x (t\overline{F}(t)^{1/2})^2 dt\right)\left(\int_0^x (\overline{F}(t)^{1/2})^2 dt\right)\\ &= \frac{4}{3}\lambda\widetilde{m}_3(x)\widetilde{\rho}(x)\end{aligned}$$

□

Lemma 7.30
Let K_1 be such that $m_3(x) \leq K_1 x m_2(x)$. Then $\widetilde{m}_3(x) \leq K_1 x E[X^2]$.

Proof.

$$\begin{aligned}
\widetilde{m}_3(x) &= m_3(x) + x^3\overline{F}(x) \\
&\leq K_1 x m_2(x)\left(1 + \frac{x^2\int_x^\infty f(t)dt}{m_2(x)}\right) \\
&\leq K_1 x m_2(x)\left(1 + \frac{\int_x^\infty t^2 f(t)dt}{m_2(x)}\right) \\
&= K_1 x m_2(x)\left(1 + \frac{E[X^2] - m_2(x)}{m_2(x)}\right) = K_1 x E[X^2]
\end{aligned}$$

□

The bound in Theorem 7.28 guarantees that a large percentage of job sizes will be treated predictably. In particular, all job sizes such that $\widetilde{\rho}(x) \leq 1 - \left(\frac{4}{3}(1-\rho)^3\right)^{1/4}$. Thus, if $\rho = 0.8$, all jobs such that $\widetilde{\rho}(x) \leq 0.678$ will be treated predictably. However, the question still remains as to how unpredictably the large jobs can be treated.

Theorem 7.31
In an M/GI/1 queue with $E[X^3] < \infty$, for jobs of size $x > K_2 E[X]$,

$$Var[T(x)]^{FB} \leq E[B(x)]h_2(\rho)$$

where

$$h_2(\rho) = \left(1 + \frac{M_3[X]}{3K_2}\right) + \frac{3\rho M_2[X]}{4K_2(1-\rho)}$$

Note that this is the same bound on the hump size as under PSJF. The difference will come in the application because the bound on the position of the hump is in terms of $\widetilde{\rho}(x)$ under FB instead of $\rho(x)$ as under PSJF, so K_2 will be smaller. We illustrate this using our running example.

Example
Again consider $X \sim Exp(1)$. Then, $\widetilde{\rho}(x) = \rho\left(1 - e^{-x}\right)$. So, $\widetilde{\rho}(x) \leq 1 - \left(\frac{4}{3}(1-\rho)^3\right)^{1/4}$ when $e^{-x} \geq 1 - \frac{1-\left(\frac{4}{3}(1-\rho)^3\right)^{1/4}}{\rho}$. This says that when $\rho = 0.8$, FB will be predictable for at least jobs of size $x \leq 1.8$. Thus, FB will be predictable for at least 83.4% of the jobs. We can use this result to set $K_2 = 1.8$ in the case where $\rho = 0.8$, which gives $Var[T(x)]^{PSJF} \leq 4.9 Var[B(x)]$. Thus, although FB is Always Unpredictable, when $\rho = 0.8$, FB is only unpredictable for at most 17% of jobs and this fraction of jobs only receives at most a factor of 5 higher variance. Note that although this is not nearly as good as what we saw under PSJF, FB is operating without knowledge of job sizes. This agrees with the behavior shown in Figure 7.4.
□

7.4.2.4 SRPT

SRPT is perhaps the most important priority-based policy due to the fact that it has been shown to be optimal with respect to mean response time. We will start the section by showing that SRPT provides predictable response times for all job sizes at low load, regardless of the service distribution. Then, we will show that, even when SRPT might not provide predictable response times for all job sizes, only a tiny percentage of the jobs receive unpredictable response times, and this unpredictability is not too bad.

We will first prove one technical lemma.

Lemma 7.32

In an M/GI/1 queue,

$$Var[R(x)]^{SRPT} = \int_0^x \frac{\lambda m_2(t)}{(1-\rho(t))^3}dt \quad \leq \quad \frac{\lambda x E[X^2]}{(1-\rho(x))^3} - \frac{\lambda \widetilde{m}_3(x)}{(1-\rho(x))^3}$$

Proof.

$$\begin{aligned}
\int_0^x \frac{\lambda m_2(t)}{(1-\rho(t))^3}dt &\leq \frac{\int_0^x \lambda m_2(t)dt}{(1-\rho(x))^3} = \frac{\lambda x m_2(x) - \lambda m_3(x)}{(1-\rho(x))^3} \\
&\leq \frac{\lambda x E[X^2]}{(1-\rho(x))^3} - \frac{\lambda x^3 \overline{F}(x)}{(1-\rho(x))^3} - \frac{\lambda m_3(x)}{(1-\rho(x))^3} \\
&= \frac{\lambda x E[X^2]}{(1-\rho(x))^3} - \frac{\lambda \widetilde{m}_3(x)}{(1-\rho(x))^3}
\end{aligned}$$

□

We can now prove that SRPT behaves predictably under low load.

Theorem 7.33

In an M/GI/1 queue, let K_1 be a constant such that $m_3(x) \leq K_1 x m_2(x)$. Under all service distributions SRPT *is predictable when $\rho \leq 0.4$. Further, all x such that $\rho(x) \leq 0.4$ are treated predictably under all service distributions, and for x such that $\rho(x) > 0.4$,*

$$Var[T(x)]^{SRPT} \leq Var[B(x)]h_1(\rho, x)^{SRPT}$$

where

$$h_1(\rho, x)^{SRPT} = \frac{(1-\rho)^3}{(1-\rho(x))^4}\left\{\left(1 - \frac{2}{3}K_1\right) + \left(\frac{5}{3}K_1 - 1\right)\rho(x)\right\}$$

Noting that for all distributions $m_3(x) \leq x m_2(x)$, we can set $K_1 = 1$ and obtain $h_1(\rho, x) \leq \frac{(1-\rho)^3}{(1-\rho(x))^4}\left\{\frac{1}{3} + \frac{2}{3}\rho(x)\right\}$.

Note that the above result only discuss the case that $E[X^3] < \infty$. As with PSJF and FB, no published work has appeared for SRPT in the case that $E[X^3] = \infty$, but Brown has an unpublished manuscript in which he proves that SRPT can be predictable in this setting.

Proof. First, we upper bound $Var[T(x)]^{SRPT}$ using Lemmas 7.29 and 7.32

$$\begin{aligned} Var[T(x)]^{SRPT} &\leq \frac{\lambda x E[X^2]}{(1-\rho(x))^3} - \frac{\lambda \widetilde{m}_3(x)}{(1-\rho(x))^3} + \frac{\lambda \widetilde{m}_3(x)}{3(1-\rho(x))^3} \\ &\quad + \frac{3}{4}\frac{\frac{4}{3}\lambda \widetilde{m}_3(x)\widetilde{\rho}(x)}{(1-\rho(x))^4} - \frac{\lambda^2 x \widetilde{m}_3(x)\overline{F}(x)}{(1-\rho(x))^4} \\ &= \frac{\lambda x E[X^2]}{(1-\rho(x))^3} - \frac{2}{3}\frac{\lambda \widetilde{m}_3(x)}{(1-\rho(x))^3} + \frac{\lambda \widetilde{m}_3(x)\rho(x)}{(1-\rho(x))^4} \\ &= \frac{\lambda x E[X^2]}{(1-\rho(x))^3}\left(1 + \frac{\widetilde{m}_3(x)(5\rho(x)-2)}{3xE[X^2](1-\rho(x))}\right) \end{aligned}$$

From this, we see that $Var[T(x)]^{SRPT} \leq Var[B(x)]$ for all x such that $5\rho(x) - 2 < 0$, i.e. $\rho(x) \leq 0.4$. Then, we apply Lemma 7.30 in the case when $\rho(x) > 0.4$ to finish the proof.

$$\begin{aligned} Var[T(x)]^{SRPT} &\leq \frac{\lambda x E[X^2]}{(1-\rho(x))^3}\left(1 + \frac{K_1 x E[X^2](5\rho(x)-2)}{3xE[X^2](1-\rho(x))}\right) \\ &= \frac{\lambda x E[X^2]}{(1-\rho(x))^4}\left(\left(1 - \frac{2}{3}K_1\right) + \left(\frac{5}{3}K_1 - 1\right)\rho(x)\right) \end{aligned}$$

□

Using this theorem, we can see that most job sizes will be treated predictably under SRPT even under high load. For x such that $\rho(x) > 0.4$, $Var[T(x)]^{SRPT} \leq Var[B(x)]$ whenever $\rho(x) \leq 1 - (1-\rho)^{3/4}$. Notice that this gives a much better range than the $\rho(x) < 0.4$ when ρ is high. When $\rho = 0.8$, SRPT is predictable for all job sizes x that have $\rho(x) \leq 0.7$ regardless of the service distribution.

We now show that, though SRPT can provide predictable response times for all job sizes under low loads, SRPT will be unpredictable for some job size under high enough load. This result follows immediately from Theorems 7.37 (which holds for the entire SMART class) and 7.33.

Theorem 7.34

In an M/GI/1 queue with $E[X^3] < \infty$, SRPT *is Sometimes Predictable. For every service distribution, there exists some ρ_{crit} and L such that, for all $\rho > \rho_{crit}$,* SRPT *is unpredictable for all jobs of size $x \geq L$.*

The prior theorems give bounds on the position and existence of the hump in $Var[T(x)]^{SRPT}/x$; to bound the height of the hump it turns out to be effective to use the same bound that we have used for PSJF and FB.

Lemma 7.35

In an M/GI/1 queue, for all x $Var[T(x)]^{SRPT} \leq Var[T(x)]^{FB}$

This lemma is a special case of Theorem 3.19.

Lemma 7.35 allows us to use the bound already derived for FB in Theorem 7.31. As in the cases of PSJF and FB, the combination of the above theorems provides tight bounds on the position and size of the hump in $Var[T(x)]^{SRPT}/x$.

Example

Again consider $X \sim Exp(1)$. $\rho(x) \leq 1 - (1-\rho)^{3/4}$ *when* $e^{-x} + xe^{-x} \geq 1 - \frac{1-(1-\rho)^{3/4}}{\rho}$. *This says that when* $\rho = 0.8$, SRPT *will be predictable for at least jobs of size* $x \leq 3.6$, *which is at least 97.2% of the jobs. We can use this result to set* $K_2 = 3.6$ *in the case where* $\rho = 0.8$ *which gives* $Var[T(x)]^{SRPT} \leq 2.9Var[B(x)]$. *Thus, although* SRPT *can be unpredictable, in the case of an exponential service distribution with* $\rho = 0.8$, SRPT *is only unpredictable for at most* 3% *of jobs and this fraction of jobs only receives at most a factor of 3 higher variance. Note both of these bounds are better than were obtained for either* PSJF *or* FB.

□

7.4.3 The predictability of scheduling classifications

The power of Definitions 7.4 and 7.5 is that they are simple enough to be tractable for the analysis of scheduling classifications in addition to the analysis of individual policies. For the remainder of this section, we will focus on understanding the impact of common scheduling heuristics and techniques on the predictability of the resulting policies. We will start by discussing the Always Predictable classifications, and then we will move to the Always Unpredictable and Sometimes Predictable classifications. We provide an overview of all the known results in Figure 7.3. However, to avoid repetition, we will not provide the proofs for all the classifications listed in Figure 7.3 since many of the proofs are very similar to one another and to the proofs for the case of mean proportional fairness. Instead, we will limit our focus to a few of the most interesting classifications.

7.4.3.1 Always Predictable

We begin by studying the class of Always Predictable policies, policies where every job size is treated predictably under all service distributions and system loads. One might expect that, since PS and PLCFS are both Always Predictable, it might be possible to also show that all SYMMETRIC policies are Always Predictable. Unfortunately, little has been proven about the variance of response times of other SYMMETRIC policies. A rare exception is the work of Avi-Itzhak and Halfin [9], where $Var[T(x)]$ under PLCFS, PS, and one other less common SYMMETRIC policy are compared. However, intuitively, $Var[T(x)]^P$ for all SYMMETRIC policies should fall between $Var[T(x)]^{PS}$ and $Var[T(x)]^{PLCFS}$, thus we conjecture that in fact all $P \in$ SYMMETRIC are Always Predictable.

Similarly, since PS is Always Predictable, one might conjecture that all PROTECTIVE policies are Always Predictable. Again though, the analysis of PROTECTIVE policies is difficult and no analytic results have been obtained about the variance of response times of policies in this class. There have been some simulation results studying $Var[T(x)]^{FSP}$ [86], but it is intuitively unclear whether PROTECTIVE policies will turn out to be Always Predictable because it seems possible that the variance of response times may be larger than that of PS since $E[T(x)]$ is guaranteed to be smaller.

7.4.3.2 Always Unpredictable

We now move to a discussion of Always Unpredictable policies, i.e. policies guaranteed under all system loads and all service distributions to treat some job size unpredictably. The only scheduling classification that turns out to be Always Unpredictable is the FOOLISH classification (see Figure 7.4).

FOOLISH Scheduling

We start with the class of FOOLISH policies, i.e. policies that bias towards large job sizes. As one would expect, FOOLISH policies will always be unpredictable for small job sizes, since the large job sizes are allowed to preempt small ones upon arrival.

Theorem 7.36
In an M/GI/1 queue, all FOOLISH *policies are Always Unfair.*

Proof. Let x_L be the lower bound of the service distribution. Then, as $x \to x_L$, $T(x)^{PLJF} \xrightarrow{\text{d}} B(x_L+W)$ since the service distribution is continuous. Further, $T(x)^{LRPT} \stackrel{\text{d}}{=} B(x+W)$. Thus, under all $P \in$ FOOLISH, as $x \to x_L$, $T(x)^P \xrightarrow{\text{d}} B(x_L+W)$. Thus, all $P \in$ FOOLISH are Always Unpredictable since for all i, $\kappa_i[B(x_L+W)] \geq \kappa_i[B(x_L)]$ for $\rho > 0$.
□

7.4.3.3 Sometimes Predictable

We will now show that many classes of policies are neither Always Predictable or Always Unpredictable. Instead, many classes have include policies that fall into the Sometimes Predictable class. That is many policies can be predictable for all job sizes under some loads and service distributions and unpredictable for some job size under other loads and service distributions. The policies that are Sometimes Predictable have more complicated behavior with respect to $Var[T(x)]$ than we observed in the cases of the Always Predictable and Always Unpredictable classes. These more complicated behaviors are illustrated in Figure 7.4.

SMART scheduling

The predictability of SMART policies is of particular interest due to the intrinsic bias against large job sizes under these policies. We have already seen one example of a SMART policy that is Always Unpredictable. However, it is possible for SMART policies to be predictable. In fact, many SMART policies (e.g. SRPT) can be predictable under low loads regardless of the service distribution. Further, if the service distribution is highly variable, SMART policies can be predictable at even high loads. However, under every SMART policy, under every service distribution, there exists some load that is high enough to cause the policy to be unpredictable.

Theorem 7.37
In an M/GI/1 queue with $E[X^3] < \infty$ there exist $\rho_c < 1$ such that all SMART *policies are unpredictable $\rho > \rho_c$.*

By combining Theorem 7.37 with the soon to be published work of Brown (which extends [47]) that shows that SMART policies are predictable in under some distributions having $E[X^3] < \infty$, it follows that all SMART policies are Sometimes Predictable

In proving Theorem 7.37, we need to prove a tighter bound on $Var[T(x)]$ than Theorem 4.4 gives directly. Thus, we first prove tighter bounds on the behavior of $Var[R(x)]^P$ for $P \in$ SMART in the following corollary.

Corollary 7.38
In an M/GI/1 queue, for $P \in$ SMART,

$$\int_0^x \frac{\lambda m_2(t)}{(1-\rho(t))^3}dt + \frac{\lambda m_3(x)}{3(1-\rho(x))^3} + \left(\frac{\lambda m_2(x)}{(1-\rho(x))^2}\right)^2 - \left(\frac{\lambda \widetilde{m_2}(x)}{2(1-\rho(x))^2}\right)^2 \tag{7.26}$$

$$\leq Var[T(x)]^P \leq$$

$$\frac{\lambda x m_2(x)}{(1-\rho(x))^3} + \frac{\lambda \widetilde{m_3}(x)}{3(1-\rho(x))^3} + \frac{\lambda m_2(x)\widetilde{m_2}(x)}{(1-\rho(x))^4} - \left(\frac{\lambda m_2(x)}{2(1-\rho(x))^2}\right)^2 \tag{7.27}$$

Proof. Let $P \in$ SMART. To bound $Var[W(x)]^P$, we use the bounds on $E[W(x)^2]$ and $E[W(x)]^2$ provided by Theorem 4.4. However, to bound $Var[T(x)]^P$ we need a better bound on $Var[R(x)]^P$ than is provided by the bounds on $E[R(x)^2]$ and $E[R(x)]^2$. We will obtain better bounds by deriving the cumulant generating function of $R(x)^P$.

View $R(x)^P$ as a special type of busy period $B_*(x)$ started by the tagged job j_x. We begin by partitioning the job size into k small intervals of length $\Delta = x/k$. The partition defines a set of $k+1$ points $0 = x_0 < x_1 < \ldots < x_k = x$. We will be taking $k \to \infty$ to obtain $\kappa_{B_*(x)}(s)$, as in [202].

While the remaining size of j_x is in $[x_i, x_{i+1})$ we will say j_x is in class i. During $B_*(x)$, j_x must remain in class i for a time T_i during which Δ work is performed on j_x and all arriving work with higher priority is served.

To see which arrivals will have higher priority, recall that the Consistency Property guarantees that once a tagged job j_x of size x begins service, only newer arrivals can preempt j_x. Further, once a new arrival j_y of size y preempts j_x, the Transitivity Property guarantees that all new arrivals of size $< y$ will have priority over j_x until j_x next receives service (which can only happen when all new arrivals of size $< y$ have completed). Thus, the length of the interruption j_y begins is the same as the length of busy period including all arrivals of size $< y$, i.e. B_y. Note that if the remaining size of j_x is r when j_y arrives, then it must be that $r \leq y \leq x$.

Under any SMART policy (for large enough k), there is some cutoff size y_i for class i jobs with $x_i \leq y_i \leq x$ such that all arriving jobs of size $< y_i$ have priority over j_x while j_x is in class i. Thus, $T_i \stackrel{\mathrm{d}}{=} B_{y_i}(\Delta)$ which gives

$$\kappa_{T_i}(s) = -\Delta(s + \lambda F(y_i) - \lambda F(y_i)\kappa_{B_{y_i}}(s))$$

Note that y_i may depend on the current system state, past history, or even an external random choice. However, though the y_is may be dependent, the length of the corresponding busy periods are conditionally independent given the choice of the y_is. Thus we have

$$\kappa_{B_{k*}(x)}(x|y_1, \ldots, y_k) \quad = \quad -\sum_{i=0}^{k} \Delta(s + \lambda F(y_i) - \lambda F(y_i)\kappa_{B_{y_i}}(s))$$

Taking derivatives to obtain the variance, we have

$$Var[B_{k*}(x|y_1, \ldots, y_k)] \quad = \quad \sum_{i=0}^{k} \Delta\lambda F(y_i) E[B_{y_i}^2]$$

Recalling that $x_i \leq y_i \leq x$ gives

$$\sum_{i=0}^{k} \Delta\lambda F(x_i) E[B_{x_i}^2] \leq Var[B_{k*}(x)] \leq \sum_{i=0}^{k} \Delta\lambda F(x) E[B_x^2]$$

Finally, letting $k \to \infty$ gives

$$\int_0^x \lambda F(t) E[B_t^2] dt \leq Var[B_*(x)] \leq \lambda x F(x) E[B_x^2]$$

which simplifies to

$$\int_0^x \frac{\lambda m_2(t)}{(1-\rho(t))^3} dt \leq Var[B_*(x)] \leq \frac{\lambda x m_2(x)}{(1-\rho(x))^3}$$

□

Before starting the proof of Theorem 7.37 we need two other technical lemmas.

Lemma 7.39
Define $\delta_x = \lambda m_2(x)/x$ and let $\epsilon > 0$.

$$\frac{\lambda x(1+\epsilon) m_2(x)}{(1-\rho(x))^3} - \int_0^x \frac{\lambda m_2(t)}{(1-\rho(t))^3} dt \quad \leq \quad \frac{\lambda^2 m_2(x)^2}{(1-\rho(x))^3} h(x) + \frac{\lambda m_3(x)}{(1-\rho(x)+\delta_x)^3}$$

where

$$h(x) = \left\{ \epsilon/\delta_x + \frac{3}{(1-\rho(x)+\delta_x)} + \frac{3\delta_x}{(1-\rho(x)+\delta_x)^2} + \frac{\delta_x^2}{(1-\rho(x)+\delta_x)^3} \right\}$$

Proof. We will use the bound in Lemma 7.15 in the first step, and then calculate directly.

$$\begin{aligned}
\frac{\lambda x(1+\epsilon) m_2(x)}{(1-\rho(x))^3} \quad - \quad & \int_0^x \frac{\lambda m_2(t)}{(1-\rho(t))^3} dt \\
\leq \quad & \frac{\lambda x(1+\epsilon) m_2(x)}{(1-\rho(x))^3} - \frac{\lambda x m_2(x) - \lambda m_3(x)}{(1-\rho(x)+\delta_x)^3} \\
= \quad & \frac{\lambda x m_2(x)}{(1-\rho(x))^3} \left\{ (1+\epsilon) - \frac{(1-\rho(x))^3}{(1-\rho(x)+\delta_x)^3} \right\} + \frac{\lambda m_3(x)}{(1-\rho(x)+\delta_x)^3} \\
= \quad & \frac{\lambda x m_2(x)}{(1-\rho(x))^3} \left\{ \epsilon + \frac{3(1-\rho(x))^2\delta_x + 3(1-\rho(x))\delta_x^2 + \delta_x^3}{(1-\rho(x)+\delta_x)^3} \right\} + \frac{\lambda m_3(x)}{(1-\rho(x)+\delta_x)^3} \\
\leq \quad & \frac{\lambda x m_2(x)}{(1-\rho(x))^3} \left\{ \epsilon + \frac{3\delta_x}{(1-\rho(x)+\delta_x)} + \frac{3\delta_x^2}{(1-\rho(x)+\delta_x)^2} + \frac{\delta_x^3}{(1-\rho(x)+\delta_x)^3} \right\} \\
& + \frac{\lambda m_3(x)}{(1-\rho(x)+\delta_x)^3}
\end{aligned}$$

□

Lemma 7.40
Let $E[X^3] < \infty$ and define $\epsilon_x = \frac{E[X^2]}{m_2(x)} \left(\frac{1-\rho(x)}{1-\rho}\right)^3$. Then, $\epsilon_x = o(1/x)$. That is $x\epsilon_x \to 0$ as $x \to \infty$.

Proof.
It is sufficient to prove that $(1-\rho(x))^3 \frac{E[X^2]}{m_2(x)} - (1-\rho)^3 = o(1/x)$. First, note that

$$\frac{E[X^2]}{m_2(x)} = 1 + \frac{\int_x^\infty t^2 f(t)dt}{\int_0^x t^2 f(t)dt} = 1 + o(1/x)$$

Thus, we have

$$(1-\rho(x))^3 \frac{E[X^2]}{m_2(x)} - (1-\rho)^3 = \sum_{i=0}^{3} \binom{3}{i} (-1)^i \left((1+o(1/x))\rho(x)^i - \rho^i\right)$$

Now, looking term by term, clearly the $i = 0$ term is $o(1/x)$. For $i \geq 1$,

$$\begin{aligned} \binom{3}{i} (-1)^i \left((1+o(1/x))\rho(x)^i - \rho^i\right) &\leq 3(\rho^i - \rho(x)^i(1+o(1/x))) \\ &= 3(\rho^i - \rho(x)^i) + o(1/x) \\ &\leq 3i(\rho - \rho(x)) + o(1/x) = o(1/x) \end{aligned}$$

where the last inequality follows from the fact that for $0 \leq a < b \leq 1$ and positive integer i:

$$b^i - a^i = (b-a)(b^{i-1} + ab^{i-2} + \ldots + a^{i-2}b + a^{i-1}) \leq i(b-a)$$

□

We are now ready to prove Theorem 7.37.

Proof of Theorem 7.37. The proof will mimic the proof of Theorem 7.14, with added complexity due to the form of (7.26). Let $P \in$ SMART and $\delta_x = \lambda m_2(x)/x$.

Define ϵ_x as

$$\epsilon_x = \frac{E[X^2]}{m_2(x)} \left(\frac{1-\rho(x)}{1-\rho}\right)^3 - 1$$

In Lemma 7.40 we show that $\epsilon_x = o(1/x)$.

Now, we can calculate

$$\begin{aligned} \frac{\lambda x E[X^2]}{(1-\rho)^3} - Var[T(x)]^P &= \frac{\lambda m_2(x) x(1+\epsilon_x)}{(1-\rho(x))^3} - Var[T(x)]^P \\ &\leq \frac{\lambda m_2(x) x(1+\epsilon_x)}{(1-\rho(x))^3} - \left(\int_0^x \frac{dt}{1-\rho(t)}\right)^2 - \frac{\lambda m_3(x)}{3(1-\rho(x))^3} - \frac{1}{2}\left(\frac{\lambda m_2(x)}{(1-\rho(x))^2}\right)^2 \end{aligned} \quad (7.28)$$

where we use the fact that $E[X^3] < \infty$ implies $\overline{F}(x) = o(x^3)$ to bound $\left(\frac{\lambda \widetilde{m}_2(x)}{2(1-\rho(x))^2}\right)^2 < \frac{1}{2}\left(\frac{\lambda m_2(x)}{(1-\rho(x))^2}\right)^2$ for large enough x in the final step.

Looking at the pieces in (7.28) using Lemmas 7.15, 7.16, 7.39, and the calculations in the proof of Theorem 7.14, we have

$$\frac{\lambda m_2(x) x(1+\epsilon_x)}{(1-\rho(x))^3} - \int_0^x \frac{\lambda m_2(t)}{(1-\rho(t))^3} dt \quad \leq \quad \frac{\lambda m_3(x)}{(1-\rho(x)+\delta_x)^3} + \frac{\lambda^2 m_2(x)^2}{(1-\rho(x))^3} h(x)$$

where $h(x) = \left\{ \epsilon_x/\delta_x + \frac{3}{(1-\rho(x)+\delta_x)} + \frac{3\delta_x}{(1-\rho(x)+\delta_x)^2} + \frac{\delta_x^2}{(1-\rho(x)+\delta_x)^3} \right\}$

Plugging this bound into (7.28) we obtain

$$\begin{aligned} \frac{\lambda x E[X^2]}{(1-\rho)^3} - Var[T(x)]^{SMART} \quad \leq \quad & \left(\frac{\lambda m_3(x)}{(1-\rho(x)+\delta_x)^3} - \frac{\lambda m_3(x)}{3(1-\rho(x))^3} \right) \\ & + \left(\frac{\lambda^2 m_2(x)^2}{(1-\rho(x))^3} h(x) - \frac{1}{2} \frac{\lambda^2 m_2(x)^2}{(1-\rho(x))^4} \right) \end{aligned} \tag{7.29}$$

We now show that for x and ρ large enough, each term in (7.29) can be made negative. We will start with the first term and then move to the second term.

Working with the first term, we have

$$\frac{\lambda m_3(x)}{(1-\rho(x)+\delta_x)^3} - \frac{\lambda m_3(x)}{3(1-\rho(x))^3} \quad = \quad \lambda m_3(x) \left(\frac{1}{(1-\rho(x)+\delta_x)^3} - \frac{1}{3(1-\rho(x))^3} \right)$$

which is negative when $3(1-\rho(x))^3 < (1-\rho(x)+\delta_x)^3$. Noting that $\sqrt[3]{3} < 2$, we can simplify this condition to

$$\begin{aligned} 2(1-\rho(x)) - (1-\rho(x)+\delta_x) \quad &< \quad 0 \\ (1-\rho) + (\rho - \rho(x)) - \delta_x \quad &< \quad 0 \end{aligned} \tag{7.30}$$

As in (7.7), this inequality holds for large enough x and ρ.

We now move to the second term in (7.29). Simplifying, we have

$$\frac{\lambda^2 m_2(x)^2}{(1-\rho(x))^3} \left(\epsilon_x/\delta_x + \frac{3}{(1-\rho(x)+\delta_x)} + \frac{3\delta_x}{(1-\rho(x)+\delta_x)^2} + \frac{\delta_x^2}{(1-\rho(x)+\delta_x)^3} - \frac{1}{2(1-\rho(x))} \right) \tag{7.31}$$

We will compare each of the positive terms to $\frac{1}{8(1-\rho(x))}$ in order to show that (7.31) is negative.

First, note that $\epsilon_x/\delta_x = o(1)$, therefore $\epsilon_x/\delta_x - \frac{1}{8(1-\rho(x))} < 0$ for large enough x.

For the remainder of the positive terms in (7.31), we argue as follows. Let $c > 0$ and $i \geq 0$ be constants. The following are equivalent

$$\begin{aligned} \frac{c\delta_x^i}{(1-\rho(x)+\delta_x)^{i+1}} \quad &< \quad \frac{1}{8(1-\rho(x))} \\ 8c\delta_x^i (1-\rho(x)) \quad &< \quad (1-\rho(x)+\delta_x)^{i+1} \end{aligned}$$

Noting that $(1-\rho(x)+\delta_x)^{i+1} > \delta_x^{i+i}$, it is sufficient to show that

$$8c(1-\rho(x)) < \delta_x \tag{7.32}$$

which holds for large enough ρ and x by a parallel argument to what was used for (7.7) and (7.30). This completes the proof for the case of an unbounded service distribution.

In the case of a service distribution with upper bound x_U, it is sufficient to look at the performance of the largest job size. Thus, by noting that $\epsilon_{x_U} = 1$, $x_U^2\overline{F}(x_U) = 0$, and plugging $x = x_U$ into (7.30) and (7.32), we obtain the result.
□

Non-preemptive policies

We now move to a discussion of the predictability under non-preemptive policies.

Non-preemptive policies have very different behavior than preemptive policies. We have seen in Section 7.2 that large job sizes see nearly deterministic response times under non-preemptive policies, because once they begin service they cannot be interrupted. However, one result of this bias towards large job sizes is that small job sizes can receive extremely variable service because they may have to wait behind the excess of a much larger job.

In fact, whenever the service distribution includes arbitrarily small jobs, these small jobs will receive unpredictable response times under non-preemptive policies.

Theorem 7.41

In an M/GI/1 queue with $E[X^3] < \infty$,[10] all non-preemptive policies are either Sometimes Predictable or Always Unpredictable. All non-preemptive policies are unpredictable for all loads if the service distribution includes arbitrarily small job sizes.

Proof. Let P be a work conserving non-preemptive policy. The response time of a job j_x of size x under P is the sum of the work in the system that will serve ahead of j_x, W_{j_x}, and all arrivals while j_x is in the system that serve ahead of j_x. This second piece can be viewed as a busy period, $B_{j_x}(W_{j_x})$. We can bound W_{j_x} from below by the excess of the job at the server upon the arrival of j_x, $\mathcal{E}$. Further, we can bound $Var[B_{j_x}(W_{j_x})] \geq Var[W_{j_x}] \geq Var[\mathcal{E}]$. Finally, we can complete the proof by observing that $\lim_{x\to 0} \frac{Var[T(x)]^P}{x} \geq \lim_{x\to 0} \frac{Var[\mathcal{E}]}{x} = \infty$.
□

However, in many real world cases there is some lower bound that can be placed on the size of a service request. In this case, non-preemptive policies *can* provide predictable service. We illustrate this using the

[10] Note that non-preemptive policies require that $E[X^3] < \infty$ in order for $Var[T(x)] < \infty$.

examples of FCFS and non-preemptive Shortest-Job-First (SJF). Note that [222]

$$\begin{aligned} Var[T(x)]^{FCFS} &= \frac{\lambda E[X^3]}{3(1-\rho)} + \frac{\lambda^2 E[X^2]^2}{4(1-\rho)^2} \\ Var[T(x)]^{SJF} &= \frac{\lambda E[X^3]}{3(1-\rho(x))^3} + \frac{\lambda^2 m_2(x) E[X^2]}{(1-\rho(x))^4} - \frac{\lambda^2 E[X^2]^2}{4(1-\rho(x))^4} \end{aligned}$$

Theorem 7.42

In an M/GI/1 queue with $E[X^3] < \infty$, FCFS *is Sometimes Predictable. (i) For all service distributions with no non-zero lower bound,* FCFS *is unpredictable. (ii) For all service distributions with lower bound $L \neq 0$, there exists a ρ_{crit} such that for all $\rho \in (\rho_{crit}, 1)$* FCFS *is predictable.*

Theorem 7.43

In an M/GI/1 queue with $E[X^3] < \infty$, SJF *is Sometimes Predictable. (i) For all service distribution with no non-zero lower bound,* SJF *is unpredictable. (ii) For service distributions with lower bound $L \neq 0$,* SJF *is predictable when $\frac{M_3[X]}{3} + \frac{3\rho M_2[X]}{4(1-\rho)} \leq \frac{L}{E[X]}$.*

The proofs of these theorems are straightforward manipulations of $Var[T(x)]$, and are thus omitted.

These two examples illustrate the strange effects of size based prioritization. While FCFS and all blind based non-preemptive policies have $Var[T(x)]/x$ that is strictly decreasing in x, size based non-preemptive policies, such as SJF, exhibit non-monotonic behavior similar to that seen under preemptive policies such as SRPT, FB, and PSJF. This contrast is illustrated in Figure 7.4.

7.5 Temporal Fairness

To this point, we have considered only *proportional fairness* measures, which are motivated by the idea that it is fair for jobs to receive response times proportional to their service times. Thus, under proportional fairness measures, it is unfair to force a small job to queue behind a large job because the response time of the small job will become unfairly large. However, proportional fairness is not the most appropriate form of fairness for every application. For instance, if a large job has been waiting in the queue for a long time, it is in some sense "unfair" for a small job that just arrived to the queue to jump in front of the large job. In this section, we consider an alternative to the concept of proportional fairness called *temporal fairness*. Temporal fairness refers to the idea that it is "fair" to serve jobs in the order in which they arrive, i.e. the order of *seniority*.

Like proportional fairness, temporal fairness arises naturally in many computer applications due to the inherent tradeoff between providing jobs of different sizes "fair" performance and providing a small overall mean response time, which requires allowing small job sizes to violate the seniority of large job sizes. This tradeoff between minimizing mean response time and maintaining temporal fairness is often an important design constraint. For example, in applications such as scheduling flows in routers, there is a tension between providing flows small overall mean response times and ensuring that streaming flows (which tend to be large) do not experience jitter as a result of being interrupted by smaller flows [179, 180]. Similarly, in designing web servers there is a tension between prioritizing small files such as .html files over larger files such as

image files in order to improve overall mean response time because if large files are always interrupted by small files web sites (which depend on both large and small files) will load more slowly [96, 182].

The tradeoff between efficiency (e.g. mean response time) and temporal fairness is perhaps best illustrated by the fact that the only *strictly* temporally fair policy is FCFS, which serves jobs in the order they arrive; however, FCFS can have extremely large mean response times under highly variable service distributions. Thus, in designing a policy for an application like web servers and routers where both temporal fairness and mean response times are important, one must strike a balance between allowing small jobs to preempt large jobs and respecting the seniority of large jobs.

In the remainder of this section, we introduce a measure of temporal fairness called *politeness*, which helps to characterize this tradeoff (Section 7.5.1). Then, in Section 7.5.2, we analyze the politeness individual scheduling policies. Finally, in Section 7.5.3 we study the politeness of scheduling techniques and heuristics.

7.5.1 Defining politeness

Informally, the idea behind the notion of politeness is that a job is treated "politely" if the fraction of time that the seniority of the job is violated is small. More formally, we have the following definition.

Definition 7.6 *We denote the **politeness experienced by a job of size** x under policy P as $Pol(x)^P$ where $Pol(x)^P$ is the fraction of the response time (of a job of size x) during which the seniority of the job is respected. The **impoliteness experienced by a job of size** x under policy P is $1 - Pol(x)^P$.*

Clearly, the politeness of FCFS is $Pol(x)^{FCFS} = 1$, which is the "most polite" a policy can be. To see how "impolite" a policy can be, let us consider PLCFS. PLCFS is the worst case for $Pol(x)$ among work conserving policies. To see this, notice that a work conserving policy can serve at most $B(\epsilon) - \epsilon$ work with lower seniority for every ϵ work with higher seniority served, which is exactly what happens under PLCFS.

Proposition 7.44
In an M/GI/1 queue, $E[Pol(x)]^{PLCFS} = 1 - \rho$.

Proof. Under PLCFS, the only time during which the seniority of the tagged job is not violated is when the tagged job is being served. We can use renewal-reward to calculate $Pol(x)$ as follows. Consider a sequence of response times $T(x)_i^{PLCFS}$ where in each renewal, reward is earned with rate 1 whenever the tagged job (of size x) is being served. Thus, x reward is earned in each renewal. Further, the expected length of each renewal is $E[T(x)]^{PLCFS} = x/(1-\rho)$. Thus,

$$E[Pol(x)]^{PLCFS} = \frac{x}{x/(1-\rho)} = 1 - \rho$$

□

7.5.2 The politeness of individual policies

We have already analyzed the politeness of two policies that provide upper an lower bounds on the politeness of work conserving policies: FCFS and PLCFS. In this section, we will analyze a wide range of policies with respect to politeness.

Though the politeness of FCFS and PLCFS are independent of x, this is not always the case. For example, consider the cases of Preemptive-Shortest-Job-First (PSJF) and Non-preemptive Shortest-Job-First (SJF).

Proposition 7.45
In an M/GI/1 queue with $E[X^2] < \infty$, we have that $E[Pol(x)]^{PSJF} = 1 - \rho(x)$ and $E[Pol(x)]^{SJF} = 1 - \rho(x)C_x^{SJF}$ where

$$C_x^{SJF} = \frac{1}{1 + \frac{x}{E[W(x)]^{SJF}}} = \frac{1}{1 + \frac{2x(1-\rho(x))^2}{\lambda E[X^2]}}$$

Note that Proposition 7.45 immediately gives that

$$E[Pol(x)]^{PLCFS} \leq E[Pol(x)]^{PSJF} \leq E[Pol(x)]^{SJF} \leq E[Pol(x)]^{FCFS},$$

which matches with intuition for how these four policies should be ordered. For an illustration of the behavior of $E[Pol(x)]$ under these and other common policies, see Figure 7.5.

Proof. Let us begin with PSJF . Consider a tagged job of size x entering a PSJF system. The only times the seniority of the tagged job is not being violated is while tagged job is being served and while the work in the system seen by the tagged job upon arrival completes. This is exactly $x + \frac{\lambda m_2(x)}{2(1-\rho(x))}$ work. Thus, the time when seniority *is* being violated is

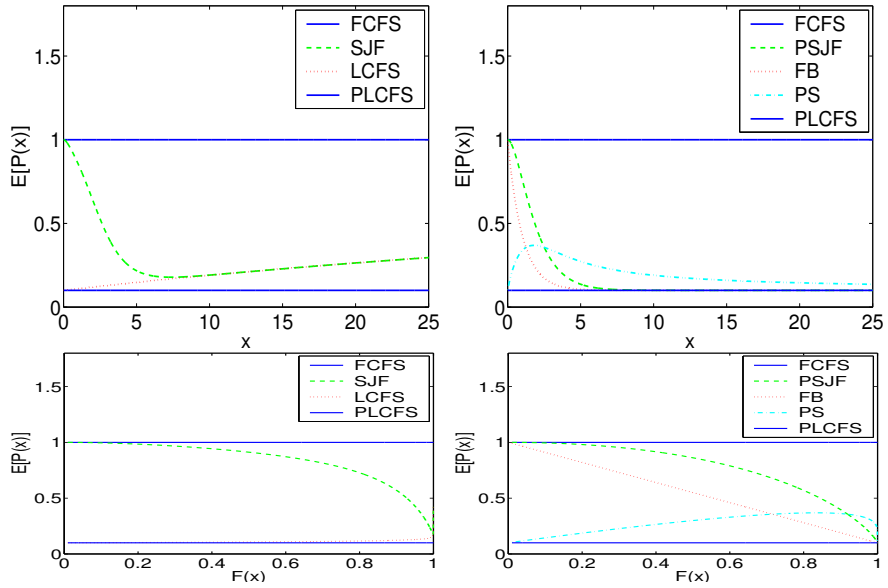
$$\begin{aligned} E[T(x)]^{PSJF} - x - \frac{\lambda E[X^2]}{2(1-\rho(x))} &= \frac{x}{1-\rho(x)} - x + \frac{\lambda m_2(x)}{2(1-\rho(x))^2} - \frac{\lambda m_2(x)}{2(1-\rho(x))} \\ &= \frac{x\rho(x)}{1-\rho(x)} + \frac{\lambda m_2(x)\rho(x)}{2(1-\rho(x))^2} \\ &= \rho(x)E[T(x)]^{PSJF} \end{aligned}$$

Using renewal-reward, as in Proposition 7.44, we obtain

$$\begin{aligned} 1 - E[Pol(x)]^{PSJF} &= \frac{\rho(x)E[T(x)]^{PSJF}}{E[T(x)]^{PSJF}} \\ &= \rho(x) \end{aligned}$$

from which the proposition follows.

Now consider a tagged job of size x entering a SJF system. The only times when the seniority of the tagged job is not being violated is while the tagged job is being served and while the work in the system seen by the tagged job upon arrival is being served. This is exactly $x + \frac{\lambda E[X^2]}{2(1-\rho(x))}$ work. Thus, the time when



(a) Non-preemptive policies (b) Preemptive policies

Figure 7.5: *An illustration of the politeness of common preemptive and non-preemptive policies. The service distribution is Exponential with mean 1 and the load is fixed at $\rho = 0.9$. All plots show* FCFS *and* PLCFS*, which act as upper and lower bounds on the possible values of $E[Pol(x)]$. The top row shows $E[Pol(x)]$ as a function of jobs size, x, and the bottom row shows $E[Pol(x)]$ as a function of the percentile of the job, $F(x)$.*

seniority is being violated is

$$\begin{aligned} E[T(x)]^{SJF} - x - \frac{\lambda E[X^2]}{2(1-\rho(x))} &= \frac{\lambda E[X^2]}{2(1-\rho(x))^2} - \frac{\lambda E[X^2]}{2(1-\rho(x))} \\ &= \frac{\lambda E[X^2]\rho(x)}{2(1-\rho(x))^2} \\ &= \rho(x)E[W(x)]^{SJF} \end{aligned}$$

Again using renewal-reward, we obtain

$$\begin{aligned} 1 - E[Pol(x)]^{SJF} &= \frac{\rho(x)E[W(x)]^{SJF}}{x + E[W(x)]^{SJF}} \\ &= \rho(x)C_x^{SJF} \end{aligned}$$

from which the proposition follows.
□

Using arguments that parallel the above, we can derive $E[Pol(x)]$ under a wide range of common

policies. Notice that the form of $E[Pol(x)]$ we saw under PSJF and SJF extends throughout many other common policies, with a notable exception being PS.

Theorem 7.46
In an M/GI/1 queue with $E[X^2] < \infty$,

$$\begin{aligned} E[Pol(x)]^{LCFS} &= 1 - \rho C_x^{LCFS} \\ E[Pol(x)]^{SRPT} &= 1 - \rho(x) C_x^{SRPT} \\ E[Pol(x)]^{LRPT} &= 1 - \rho \\ E[Pol(x)]^{FB} &= 1 - \widetilde{\rho}(x) \\ E[Pol(x)]^{PS} &= 1 - \rho + \frac{\lambda \widetilde{m_2}(x)}{2x} \end{aligned}$$

where

$$\begin{aligned} C_x^{LCFS} &= \frac{1}{1 + x/E[W(x)]^{LCFS}} = \frac{1}{1 + \frac{2x(1-\rho)}{\lambda E[X^2]}} \\ C_x^{SRPT} &= \frac{\int_0^x \frac{\rho(t)}{\rho(x)} \frac{dt}{1-\rho(t)} + E[W(x)]^{SRPT}}{E[T(x)]^{SRPT}} \end{aligned}$$

Proof. We will only prove the result for the case of PS since the other arguments all mimic the proof of Proposition 7.45.

To prove the result for PS, recall Theorem 4.16, which characterizes the system state under SYMMETRIC policies. Consider the experience of a tagged job of size x. We can calculate the amount of work done on jobs in the system when the tagged job arrived using the fact that all jobs in the system upon the arrival of the tagged job have i.i.d. remaining sizes distributed as the equilibrium distribution, $\mathcal{E}$. Further, the number of jobs in the system, N^{PS}, is geometric with mean $\rho/(1-\rho)$. Thus, the work done on jobs in the system when the tagged job arrived is

$$\begin{aligned} E\left[\sum_{i=1}^{\infty} \mathcal{E} I(\mathcal{E} < x)\right] &= \sum_{i=1}^{\infty} iE[\mathcal{E} I(\mathcal{E} < x)] P(N^{PS} = i) \\ &= E[N^{PS}] E[\mathcal{E} I(\mathcal{E} < x)] \\ &= \frac{\rho}{1-\rho} \left(\int_0^x \frac{t\overline{F}(t)}{E[X]} dt \right) \\ &= \frac{\lambda \widetilde{m_2}(x)}{2(1-\rho)} \end{aligned}$$

Thus, we have that

$$\begin{aligned} E[Pol(x)]^{PS} &= \frac{x + \frac{\lambda \widetilde{m_2}(x)}{2(1-\rho)}}{x/(1-\rho)} \\ &= 1 - \rho + \frac{\lambda \widetilde{m_2}(x)}{2x} \end{aligned}$$

□

The politeness of the policies in Theorem 7.46 and Proposition 7.45 is illustrated in Figure 7.5. Clearly, many of these policies are polite to most job sizes; however, under all the policies so far (except FCFS) there are cases where some particular job size is treated as impolite as possible, i.e. $E[Pol(x)] = 1 - \rho$. For instance, PSJF, FB, and PS have $E[Pol(x)] \to 1 - \rho$ as $x \to \infty$ for all ρ, and SJF will treat some medium-large sized job as impolitely as possible as $\rho \to 1$. This is unsatisfactory, and so our goal now is to present a family of policies that have "bounded impoliteness". To formalize this concept, we use the following definition.

Definition 7.7 *Let $0 < \rho < 1$ and $E[X^2] < \infty$ in an M/GI/1 system. A policy P is **k-polite** if*

$$\inf_x E[Pol(x)]^P > 1 - k\rho.$$

*For $k = 1$, we simply say that scheduling policy P is **polite**. If $\inf_x E[Pol(x)]^P = 1 - \rho$, we say that P is **impolite**.*

All the policies we have seen so far have some service distribution where some job size is treated impolitely. However, the following simple family of policies provides "bounded impoliteness" in the sense that the family provides a policy that is k-polite for arbitrary k.

Definition 7.8 *Under the **Jump-To-Front(q)** (JTF(q)) policy an arriving job joins the back of the queue with probability $1 - q$ and goes immediately into service (pushing every other job back one position in the queue) with probability q.*

Before we can analyze the politeness of JTF policies, we will first derive the mean response time of these policies.

Proposition 7.47
In an M/GI/1 queue with $E[X^2] < \infty$,

$$E[T(x)]^{JTF(q)} = \frac{x + \frac{\lambda(1-q)E[X^2]}{2(1-\rho)}}{1 - q\rho}$$

Proof. Consider a tagged job of size x. With probability q the job will wait only behind other arriving jobs that "jump to the front." These jobs form a busy period with load $q\rho$. With probability $1-q$ the job will wait behind all the work in the queue, $\frac{\lambda E[X^2]}{2(1-\rho)}$, and the busy period of arriving jobs that jump to the front.

Again the load of the arriving jobs that jump to the front is $q\rho$. The proof is completed by noting that for any random variable Y, $E[B(Y)] = E[Y]/(1-\gamma)$, where γ is the load of the arriving jobs:

$$\begin{aligned} E[T(x)]^{JTF(q)} &= q\left(\frac{x}{1-q\rho}\right) + (1-q)\left(\frac{x + \frac{\lambda E[X^2]}{2(1-\rho)}}{1-q\rho}\right) \\ &= \frac{x + \frac{\lambda(1-q)E[X^2]}{2(1-\rho)}}{1-q\rho} \end{aligned}$$

□

Using the Proposition 7.47, it is now straightforward to analyze the politeness of JTF(q) policies.

Theorem 7.48
In an M/GI/1 queue with $E[X^2] < \infty$, JTF(q) is q-polite.

Proof. Consider the politeness experienced by a tagged job of size x. With probability $1-q$ all the work the tagged job sees upon arrival will complete before the tagged job. Otherwise, none of the work the tagged job sees upon arrival will complete before the tagged job. Thus,

$$\begin{aligned} E[Pol(x)]^{JTF(q)} &= \frac{x + (1-q)\frac{\lambda E[X^2]}{2(1-\rho)}}{\frac{x + \frac{\lambda(1-q)E[X^2]}{2(1-\rho)}}{1-q\rho}} \\ &= 1 - q\rho \end{aligned}$$

□

7.5.3 The politeness of scheduling classifications

Using the understanding of the politeness of individual policies that we developed in the previous section, we now move to the task of classifying the politeness of scheduling techniques and heuristics. Parallel to proportional fairness and predictability, we introduce the following classes.

Definition 7.9 *Further, we say that a policy P is **Always Polite** if P is polite under all loads and service distributions. A policy P is **Sometimes Polite** if P is polite under some loads and service distributions and impolite under other loads and service distributions. A policy P is **Always Impolite** if P is impolite under all loads and service distributions.*

These three classes identify three distinct behaviors with respect to politeness, and are simple enough to allow us to analyze the classes of scheduling techniques and mechanisms that are the focus of this thesis.

Using Theorem 7.46, we can already classify many common policies according to politeness. For example, FCFS is Always Polite, PSJF is Always Impolite, and SJF is Sometimes Polite. In the remainder

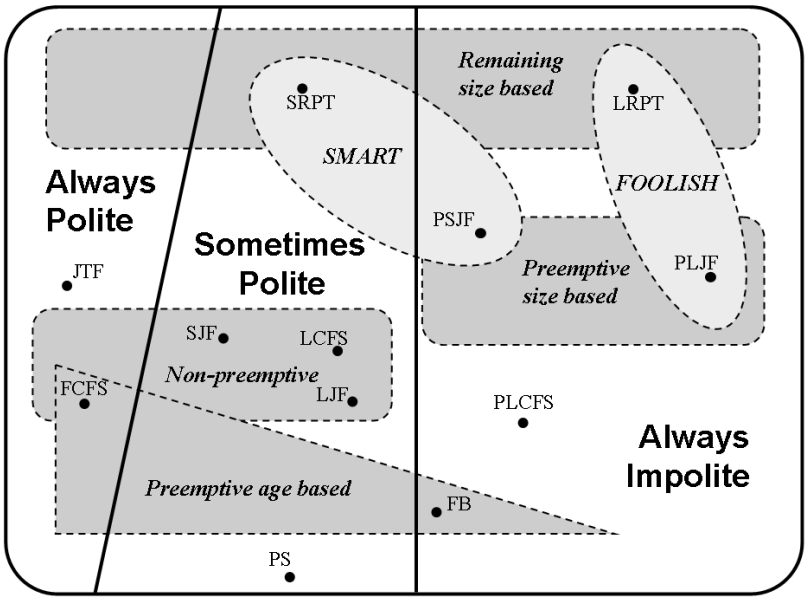


Figure 7.6: *An illustration of the classification of common prioritization techniques and heuristics with respect to politeness.*

of this section, we will extend the analyses of individual policies in order to classify the behavior of classes of scheduling techniques and heuristics with respect to politeness. The results are summarized in Figure 7.6. Figure 7.6 illustrates that most common scheduling techniques and heuristics lead to policies that are impolite in many situations. However, if we restrict our discussion to service distributions with finite upper bounds, many of these techniques and heuristics can be polite. For example, PS and SRPT are polite in such settings.

A useful tool in the analysis of scheduling techniques and heuristics is the following necessary condition for an Always Polite policy. This theorem provides a simple technique for showing that a class of policies is *not* Always Polite.

Theorem 7.49

In an M/GI/1 queue with $E[X^2] < \infty$, if a policy is Always Polite then

$$\lim_{x \to \infty} \frac{E[T(x)]}{x} < \frac{1}{1-\rho}$$

In particular, any policy P with $E[T(x)]^P/x \to 1/(1-\rho)$ as $x \to \infty$ is impolite when the service distribution has no finite upper bound.

This theorem provides an interesting view of the tradeoff between temporal fairness and proportional fairness. In particular, combining Theorem 7.49 with Theorem 7.9 yields the following corollary.

Corollary 7.50
In an M/GI/1 queue, no policy can be Always Polite and Always Fair.

Now, let us prove Theorem 7.49.

Proof of Theorem 7.49. Consider a the politeness experienced by a tagged job of size x. At most, the work done (before the tagged job completes) on jobs with higher seniority is $x + \frac{\lambda E[X^2]}{2(1-\rho)}$. Thus,

$$E[Pol(x)]^P \leq \frac{x + \frac{\lambda E[X^2]}{2(1-\rho)}}{E[T(x)]^P}$$

Now, consider a policy P such that $E[T(x)]^P/x \to 1/(1-\rho)$ as $x \to \infty$. Under such a P,

$$\begin{aligned} \lim_{x\to\infty} E[Pol(x)]^P &\leq \lim_{x\to\infty} \frac{1 + \frac{\lambda E[X^2]}{2x(1-\rho)}}{E[T(x)]^P/x} \\ &= (1-\rho) \end{aligned}$$

Thus, no such P is Always Polite.
□

7.5.3.1 Scheduling techniques

We will start the analysis of the politeness of scheduling classes by analyzing the politeness of various scheduling techniques.

Non-preemptive policies

Clearly, we expect non-preemptive policies to be polite since once a job gets to the server no one else can violate job's seniority. This turns out to be the case, even under policies such a SJF where small jobs can violate the seniority of large jobs until they begin to receive service. However, not all non-preemptive policies are Always Polite, for instance as $\rho \to 1$, $E[Pol(x)]^{SJF} \to 1 - \rho = 0$ for some x.

Theorem 7.51
In an M/GI/1 queue, all non-preemptive policies are either Always Polite or Sometimes Polite

Note that we have already seen that FCFS is Always Polite and SJF is Sometimes Polite, so this classification is as tight as possible.

Proof. Consider the politeness of a tagged job having size x. Let Q be the amount of work that is in the system upon the arrival of the tagged job that completes before the tagged job. Then, at worst, the impoliteness experienced by the tagged job is

$$1 - E[Pol(x)] \leq \frac{E[B(Q)] - E[Q]}{x + E[B(Q)]}$$

since at worst all later arrivals will violate the seniority of the tagged job. Now, we need only show that $1 - E[Pol(x)] < \rho$ for some some load and service distribution regardless of Q.

$$\begin{aligned} 1 - E[Pol(x)] &\leq \frac{E[B(Q)] - E[Q]}{x + E[B(Q)]} \\ &= \frac{\frac{E[Q]}{1-\rho} - E[Q]}{x + \frac{E[Q]}{1-\rho}} \\ &= \rho \frac{E[Q]1 - \rho}{x + \frac{E[Q]}{1-\rho}} \end{aligned}$$

So, clearly $E[Pol(x)] < 1 - \rho$ for all x under any distribution with non-zero lower bound.
□

Preemptive size based policies

As one would expect, all preemptive size based policies turn out to be Always Impolite. These policies must give some job size the lowest priority, and this job size is then treated impolitely.

Theorem 7.52
In an M/GI/1 queue, all preemptive size based policies are Always Impolite.

Proof. First, consider the case when there is a size y that receives the lowest priority. Then, since all other job sizes have priority over y, the response time of y is

$$E[T(y)] = \frac{x}{1-\rho} + \frac{\lambda E[X^2]}{2(1-\rho)^2}$$

Further, all the work in the system when y arrives will complete before y. Thus, the politeness of y is

$$\begin{aligned} E[Pol(y)] &= \frac{x + \frac{\lambda E[X^2]}{2(1-\rho)}}{\frac{x}{1-\rho} + \frac{\lambda E[X^2]}{2(1-\rho)^2}} \\ &= (1 - \rho) \end{aligned}$$

So, the policy is impolite.

To handle the case when no finite size y receives the lowest priority, simply create an infinite sequence of job sizes $\{y_i\}$ such that the mass of job sizes with lower priority than y_i monotonically converges to zero. Then, using the same argument as above, $E[Pol(y_i)] \to (1 - \rho)$.
□

7.5.3.2 Scheduling heuristics

We now move to the analysis of the politeness of various scheduling techniques. Unfortunately, we can only present results for the SMART and FOOLISH classes. The analysis of the PROTECTIVE and SYMMETRIC classes is more difficult, though we have already analyzed the politeness of two SYMMETRIC policies: PS and PLCFS.

SMART policies

One might expect that SMART policies are Always Impolite to large job sizes because SMART policies prioritize small jobs at the expense of larger ones. However, it turns out that many SMART policies can be polite under when the service distribution is upper bounded, e.g. SRPT and RS.

Theorem 7.53

In an M/GI/1 queue, all SMART *policies are Sometimes Polite or Always Impolite.*

Proof. We will prove the result under the assumption that $E[X^2] < \infty$, but the same arguments can easily be extended to the case when $E[X^2] = \infty$. Consider the experience of a job of size x. We will show that any job of size $x \to \infty$ must be treated impolitely. Using the bounds in Theorem 4.4, we have that Then

$$\begin{aligned} \lim_{x\to\infty} E[Pol(x)] &= \frac{x\frac{\lambda E[X^2]}{2(1-\rho)}}{\frac{x}{1-\rho} + \frac{\lambda E[X^2]}{2(1-\rho)^2}} \\ &= (1-\rho) \end{aligned}$$

□

FOOLISH policies

As expected, it turns out that all FOOLISH policies are Always Impolite because small job sizes are guaranteed to be biased against and have no hope of increasing their priority as the wait in the queue. This leads to an interesting contrast between SMART and FOOLISH policies. SMART policies can treat large job sizes politely by allowing the priority of large job sizes to increase as they become smaller; where as FOOLISH policies force small jobs to be the lowest priority the whole time they are in the queue.

Theorem 7.54

In an M/GI/1 queue, all FOOLISH *policies are Always Impolite.*

Proof. Consider the experience of a job of size x. Let x^- be the lower bound of the service distribution. Using the bounds in Theorem 4.14, we have that

$$\begin{aligned} E[Pol(x)] &\leq \frac{x + \frac{\lambda(E[X^2] - m_2(x))}{2(1-\rho+\rho(x))}}{\frac{x}{1-\rho} + \frac{\lambda E[X^2]}{2(1-\rho)^2}} \\ &\to (1-\rho) \text{ as } x \to x^- \end{aligned}$$

□

7.6 Hybrid fairness metrics

Until this point in the chapter, we have presented fairness metrics that study either proportional or temporal fairness in isolation. Following our work, a number of other researchers have used a contrasting approach. Motivated by our metrics, other researchers have begun developing fairness metrics that capture both proportional and temporal fairness within one measure. These hybrid fairness metrics capture both the idea that seniority should be respected and the idea that small jobs should not be forced to wait behind larger jobs. In this section, we will survey the three most well developed metrics to emerge in the last two years: Order Fairness (OF) [19], Resource Allocation Queueing Fairness Measure (RAQFM) [185], and Discrimination Frequency (DF) [195]. We will introduce each of these measures and survey how common individual policies perform under each. Our goal is not to provide a detailed summary of these measures, but rather to alert and excite the reader about ongoing work in the field.

7.6.1 Order Fairness

Order fairness, introduced by Avi-Itzhak and Levy in [19] is defined as follows:

Definition 7.10 *Consider a GI/GI/1 queue and a non-preemptive policy P, the **expected order fairness** is defined as $E[OF]^P = Var[W]^P$.*

Order fairness is actually primarily motivated by the concept of temporal fairness. The authors develop the measure by studying the behavior of non-preemptive policies in a GI/D/1 queue, where proportional fairness is inconsequential because all jobs have the same size. In this setting they take an axiomatic approach to developing a measure for temporal fairness in such a system: they present four axioms of fairness and then categorize the metrics that satisfy these axioms. The four axioms they present each characterize how the fairness measure should react when two jobs are interchanged by the scheduling policy (i.e. when the positions of jobs i and j in the service order are switched). Define $f_{i,j,P}$ to be the change in the fairness measure after switching the positions of i and j under policy P and let P' be the resulting policy.

- *Monotonicity under job interchange*: $f_{i,j,P}$ is strictly increasing in the seniority difference of jobs i and j.
- *Reversibility under job interchange*: $f_{i,j,P} = -f_{j,i,P'}$.
- *Independence of position and time*: $f_{i,j,P}$ is independent of the positions of i and j in the queue and of the time when the switch occurs.
- *Independence of customers not interchanged*: $f_{i,j,P}$ is independent of all customers in the queue other than i and j.

In the GI/D/1 queue it can be proven that fairness measures that obey these axioms are of the form

$$c\sum_i a_i d_i^P + \alpha_i \tag{7.33}$$

where $c > 0$ and α are constants and d_i^P is the displacement of the ith job under P (i.e. the number of positions job i is pushed ahead or behind in the queue under P). Further, Definition 7.10 follows from the (7.33) by taking expectation under the assumption that c is the same across all busy periods and $\alpha = 0$. This serves as a nice validation of Definition 7.10 in the GI/D/1 setting.

However, outside of the GI/D/1 setting, it is clear that the four axioms above are not achievable. Further, the axioms above are only appropriate under non-preemptive scheduling policies (where the policy can be viewed as an order on the jobs in the queue). The restriction to non-preemptive scheduling cannot be avoided, but the authors do argue for the appropriateness of order fairness outside of the GI/D/1 setting.

Though Definition 7.10 does not satisfy the four axioms outside of the GI/D/1 setting, the authors extend the applicability of order fairness by injecting the notion of proportional fairness into the measure. They argue intuitively that it is unfair to force a small job to wait behind a large job unless the large job has been in the system a long time before the small job arrives. Thus, the fairness of customers i, j should be dependent on the difference in the waiting times of these jobs, w_i, w_j. In particular, it should be some function $h(\cdot)$ that is increasing in $|w_j - w_i|$. Noting that

$$|w_j - w_i| = \begin{cases} |a_i - a_j - s_j|, & j \text{ is served ahead of } i \\ |a_i - a_j + s_k|, & \text{else} \end{cases}$$

it follows that running job j with arrival time a_j and service demand s_j ahead of job i is "more fair" than running i ahead of j when

$$a_i - a_j > (s_j - s_i)/2$$

Using pairwise comparisons, the order fairness of a sample path can be written as

$$\lim_{n\to\infty} \frac{1}{2n^2} \sum_{i=1}^{n} \sum_{j=1}^{n} h(|w_j - w_i|)$$

Finally, by taking $h(x) = c/2x^2$ this notion of fairness can be seen to be equivalent to comparing $Var[W]$ as in Definition 7.10. As noted by the authors, this generalization of order fairness to the GI/GI/1 queue is less than ideal because many of the original axioms no longer hold. For instance, when interchanging two jobs, the pairwise comparison of other jobs in the system is affected (i.e. the fourth axiom is violated). However, the simplicity of the resulting measure ($Var[W]^P$) is appealing.

The simplicity of the order fairness measure allows the comparison of many non-preemptive policies. It is immediately clear that FCFS is more fair than LCFS and SJF is more fair than LJF under this measure. Thus the measure is in some sense capturing both the notions of temporal and proportional fairness. Further, it is interesting to note that the comparison of SJF and FCFS with respect to order fairness is dependent on the service distribution. For instance, when the service distribution is lightly variable, FCFS can be more fair than SJF but when the service distribution is highly variable SJF can be more fair than FCFS.

7.6.2 RAQFM

Following the introduction of the Order Fairness measure, Avi-Itzhak and Levy in combination with Raz present a second hybrid fairness measure that aims directly at accounting for both proportional and temporal fairness [185].

Definition 7.11 *Consider a GI/GI/1 queue. Let the discrimination of a job i, D_i be $D_i = \int_{a_i}^{d_i} s_i(t) - 1/N(t)dt$ where a_i and d_i are the arrival and departure times of job i, $s_i(t)$ is the percent of the service capacity given to job i and $N(t)$ is the number of jobs in the system at time t. Then the **fairness of policy** P under RAQFM is $Var[D]$.*

The basic philosophy behind this measure is that at all times, every job in the system is worthy of an even share of the server capacity, i.e. at every time t, every job in the system deserves a service rate of $1/N(t)$. Thus, the fairness of a policy is determined by looking at the variation from this service rate jobs experience.[11] In this sense, the measure is a generalization of the idea that PS is the "most fair" policy. In fact, PS is the only policy where $Var[D] = 0$.

It can easily be seen that RAQFM combines aspects of both proportional and temporal fairness. If we consider a non-preemptive setting, it is easy to see that the impact switching the order jobs i and j are serviced has on $Var[D]$ is monotonically increasing in both (i) the difference in the arrival times of i and j and (ii) the difference in the service demands of i and j. Thus RAQFM represents one particular weighting of temporal and proportional fairness, which is motivated by PS; however the measure is not flexible in the sense that the weighting of temporal and proportional fairness cannot be adjusted.

As one can guess from the form of Definition 7.11, deriving $Var[D]$ under scheduling policies is a difficult task. In fact, closed form analyses of $Var[D]$ under most scheduling policies have proven elusive. However, it is possible to obtain numerical results for a few simple policies in the M/M/1 setting. In particular, FCFS, LCFS, ROS, and PLCFShave been compared [185]. These numerical comparisons indicate that

$$Var[D]^{PLCFS} \geq Var[D]^{LCFS} \geq Var[D]^{ROS} \geq Var[D]^{FCFS} \geq Var[D]^{PS} = 0$$

Further, it seems that $Var[D]^P \to 0$ as $\rho \to 0$ and that $Var[D]^P$ increases monotonically with ρ under all P. Unfortunately, no size based policies have been analyzed with respect to $Var[D]$. Though followup work has looked at 2-class priority policies, the non-Markovian properties of exact size and remaining size based policies make analysis more difficult.

It should be noted that Definition 7.11 is quite easy to generalize beyond single server models. Raz, Avi-Itzhak, and Levy have already begun work studying the fairness of multiserver and multiqueue systems using variations of RAQFM [184].

7.6.3 Discrimination Frequency

The final hybrid metric that we will discuss is Discrimination Frequency, which was introduced by Sandmann in [195]. Unlike RAQFM, which implicitly captured the notions of proportional fairness and temporal fairness, Discrimination Frequency is defined as an explicit combination of two measures, one for proportional fairness and one for temporal fairness.

[11] The variation is of interest because $E[D] = 0$ regardless of the work conserving policy being studied. To see this, note that whenever the system is busy at time t, $\sum_i s_i(t) - 1/N(t) = 0$ because $\sum_i s_i(t) = 1$.

Definition 7.12 *Let n_i be the number of jobs that arrived no earlier than and completed no later than job i. Let m_i be the number of jobs with remaining size no smaller than job i when job i arrives that complete no later than job i. Then the Discrimination Frequency of job i is*

$$DF(i) = n_i + m_i$$

. In an M/GI/1 queue, fairness is measured using $E[DF]$.

Clearly, n_i and m_i provide measures of temporal and proportional fairness respectively. Thus, one can easily adjust definition to balance the importance of proportional and temporal fairness as desired for any particular application. This flexibility is a key benefit of Discrimination Frequency over RAQFM.

Another important property of Discrimination Frequency is that the analysis of $E[DF]^P$ is not too difficult under many scheduling policies. In fact, the analysis techniques parallel the simple techniques used to analyze Politeness is Section 7.5. Using these techniques it has been shown that, for example,

$$\begin{aligned} E[DF]^{FCFS} &= \frac{\lambda^2 E[X^2]}{4(1-\rho)} + \rho - \lambda E[\min(X_1, X_2)] \\ E[DF]^{LCFS} &= \frac{\lambda^2 3E[X^2]}{4(1-\rho)} + \rho - \lambda E[\min(X_1, X_2)] \\ E[DF]^{SJF} &= \frac{\lambda^2 E[X^2]}{2} \int_0^\infty \frac{F(x)}{1-\rho(x)} dF(x) + \rho - \lambda E[\min(X_1, X_2)] \end{aligned}$$

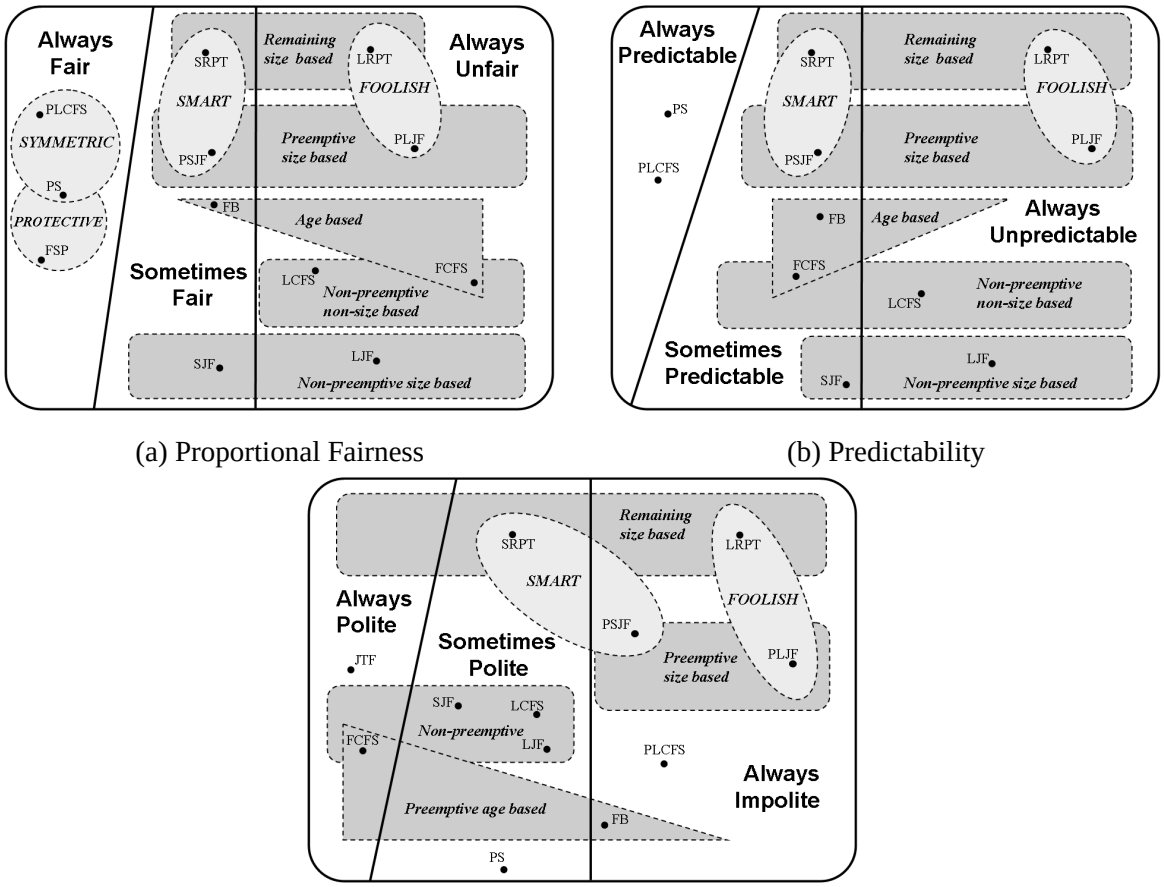
where X_1 and X_2 are i.i.d. service demands [196]. Thus, $E[DF]^{SJF} \leq E[DF]^{FCFS} \leq E[DF]^{LCFS}$ for the M/M/1 setting. Further, results for other more complicated scheduling disciplines are not too difficult to obtain and are forthcoming.

7.7 Concluding remarks

In this chapter, we have moved beyond mean response time to consider the *fairness* of scheduling policies. This is an important task because policies that perform well for mean response time often give priority to small jobs at the expense of large jobs, e.g. SRPT, and thus create worries about whether large jobs are treated fairly. Despite the ubiquitousness of such worries, the fairness of scheduling policies is largely ignored in traditional scheduling research because of the difficulty in defining a measure of fairness.

In this chapter, we have presented, for the first time, a set of metrics for studying the fairness of scheduling policies in M/GI/1 queues. We developed measures for studying the proportional and temporal fairness of scheduling policies. We presented intuitive, philosophic, and mathematical motivation for our measures. This work has served to jump-start a new focus on fairness in the scheduling community, which has led to a number of other researchers introducing their own notions of fairness for use in a wide variety of application settings (see Section 7.6).

In addition to presenting new fairness measures, we analyzed the fairness of both individual policies and scheduling classifications, see Figure 7.7. We found that it is very difficult for scheduling policies to be fair. In particular, there are very few common policies, techniques, or heuristics that lead to policies that are fair



(a) Proportional Fairness (b) Predictability

(c) Politeness

Figure 7.7: *An illustration of the classifications of common prioritization techniques and heuristics with respect to proportional fairness, predictability, and politeness. Note the many parallels between the proportional fairness and predictability classifications, which indicates that policies have similar behaviors with respect to* $E[T(x)]/x$ *and* $Var[T(x)]/x$.

across all loads and service distributions with respect to either proportional or temporal fairness. Further, these two fairness measures are conflicting: we proved that it is impossible for any policy to be both Always Fair and Always Polite (Corollary 7.50). Not only do the two types of fairness conflict with each other, they conflict with the goal of providing a small mean response time. In particular, there are no common scheduling policies, techniques, or heuristics that are near optimal with respect to mean response time and provide either temporal or proportional fairness.

However, we also found that, surprisingly, SRPT can be fair under many common situations. Thus, in some situations it is possible to minimize mean response time and still be fair. In particular, with respect to proportional fairness, SRPT is fair when either the load is low enough or the tail of the service distribution is heavy enough. In fact, beyond SRPT, all SMART policies provide a good balance of proportional fairness,

temporal fairness, and mean response time (efficiency). By giving small jobs high priority, almost all job sizes (and sometimes *all* job sizes) are treated fairly, and mean response time is near optimal.

The results in this chapter help to ease worries that giving priority to small jobs leads to large jobs being treated unfairly. In fact, the results in this chapter have had a huge impact in this regard: since our initial work on fairness, a number of new designs that prioritize small jobs have emerged for a wide variety of applications. For example, in web servers [96, 182], routers, [179, 180], wireless networks [102, 136], peer-to-peer systems [178], operating systems [74], databases [138, 139], and security applications [**?**].

PART IV

Broader Models: Moving Beyond the M/GI/1

Until this point in the thesis, our analysis of scheduling policies has focused almost entirely on the M/GI/1 queue. Limiting our focus to this setting has allowed us to consider general classifications of scheduling policies and a broad range of performance metrics, thus gaining important insights. However, in order to further bridge the gap between theoretical results and practical applications it is essential to understand the impact of practical generalizations of these models. In particular, there are many aspects of the M/GI/1 model that are unrealistic for computer systems. Though the model allows for general job sizes, assuming a Poisson arrival process and a single server is often not realistic. In Part IV, we move beyond these two assumptions and study (i) arrivals from interactive users and (ii) multiserver architectures.

In real systems the arrival process of jobs is far from Poisson. One fundamental difference is that users of computer systems are interactive. That is, a user must wait for her previous request to complete before submitting a new request. In Chapter 8, we study the impact of these dependencies between arrivals and completions. We will show that dependencies between the arrival and completion processes can limit the benefits provided by scheduling with respect to mean response time. However, we also illustrate that, in many practical settings, scheduling is still quite beneficial.

Next, in Chapter 9 we study scheduling in multiserver architectures. This is an increasingly practical generalization to consider given the growing trend towards server farm architectures and multi-core designs, which both employ multiple cheaper, slower servers instead of a single fast, expensive server. In the single server model we have discussed so far, one large job can swamp the system if the scheduling discipline does not allow smaller jobs to bypass the larger job in the queue. However, in a multiserver system one large job cannot swamp the entire system, thus the presence of multiple servers has a dramatic impact on the effectiveness of scheduling. In addition, scheduling can have a dramatic impact on the design of multiserver systems.

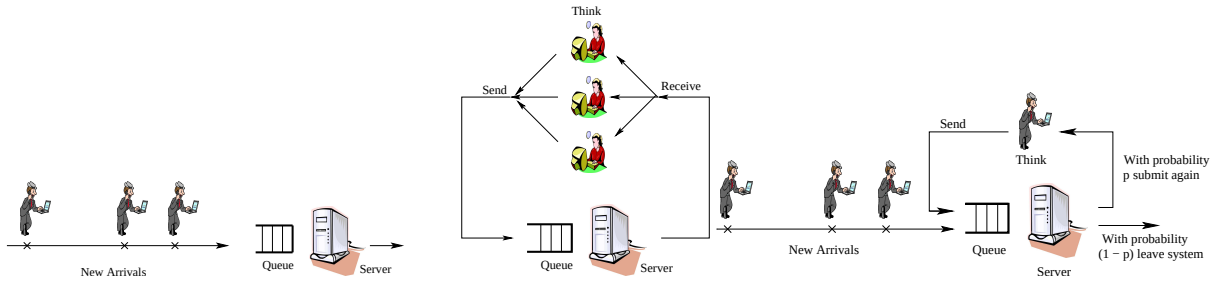
CHAPTER 8

The impact of interactive users

Until this point in the thesis, we have considered only systems where the arrival process is independent of the completion process, as in Figure 8.1(a). Such a model is referred to as an *open system model* and is a simplification of the behavior of real users. In practice, there are many cases where users need to wait for a request to complete before making a second request. For instance, in the case of web servers, a user must wait for a web page to load before clicking on the link to request a new page. Thus, a common alternative to an open system model is a *closed system model* where new job arrivals are only triggered by job completions (followed by think time), as in Figure 8.1(b). Again however, the closed system model is often a simplification of the behavior of real users. For example, users do not permanently remain at a web site. Instead, in many cases, users arrive to a system, behave as if they are in a closed system for a short while, and then leave the system as in Figure 8.1(c). Such a model is referred to as a *partly-open system model*, and behaves as a hybrid of the open and closed system models.

Given the feedback between the arrival and completion processes that occurs in computer systems, it is important to understand how the benefits of scheduling that we have illustrated in the open system model translate to the closed and partly-open system models. Intuitively, it is clear that scheduling will be less effective in settings with feedback than it was in the open system model. This is because, for open systems, scheduling is the only way to avoid having large queues build up while a large job is at the server in an open system, while in systems with feedback this problem is avoided even under FCFS scheduling because only a limited number of new arrivals can occur if there are no completions. However, we will show that scheduling still provides performance benefits in systems with feedback between arrivals and completions in many practical settings, just not the extreme gains we saw in open systems.

In addition to the importance of understanding the impact of the underlying system model (open, closed, or partly-open) on the effectiveness of scheduling, understanding the differences between the models is an important task in its own right due to the fact that a system model is at the heart of every workload generator used by practitioners to evaluate design decisions. Table 8.1 surveys the system models in a variety of web related workload generators used by systems researchers today. The table is by no means complete, but it illustrates the wide range of workload generators and benchmarks available. There is a mixture of both open and closed system models at the heart of these generators/benchmarks, though most assume a closed system model. Interestingly, many generators/benchmarks for the same purpose rely on different system models.



(a) Open system (b) Closed system (c) Partly-open system

Figure 8.1: *Illustrations of the closed, open, and partly-open system models.*

Further, for many of these workload generators, it was quite difficult to figure out which system model was being assumed – the builders often do not seem to view this as an important factor worth mentioning in the documentation. Thus, though every researcher is well aware of the importance of setting up one's experiment so that the system being modeled is "accurately represented," researchers typically pay little attention to whether the workload generator uses a closed, open, or partly-open system model.

In this chapter our goal is to illustrate the enormous impact of the underlying system model (open, closed, or partly-open) in practical settings, particularly the impact of the system model on the effectiveness of scheduling. However, the analysis of closed and partly-open system models is a difficult problem, and outside of scheduling policies where product-form results hold (e.g. FCFS and PS) little is understood analytically. Thus, we cannot hope to obtain results in the generality we have obtained in the open system setting. Instead, we obtain our results primarily via real-world implementations. We consider a range of real world case studies in this chapter, including: web servers receiving static HTTP requests, the back-end database in e-commerce applications, an auctioning web site, and a supercomputing center. Our simulation and implementation experiments in these case studies lead us to identify *eight principles* that summarize the differences between open, closed, and partly-open system models [206]. These principles may be categorized by their area of impact.

The first set of principles (see Sections 8.3 and 8.4.1) describe the *difference in mean response time* under open and closed system models and how *various parameters affect these differences*. We find, for example, that for a fixed load, the mean response time for an open system model can exceed that for a closed system model by an order of magnitude or more. Even under a high Multi-Programming Level (MPL), the closed system model still behaves "closed" with respect to mean response time, and there is still a significant difference between mean response times in closed and open systems even for an MPL of 1000. With respect to service demands (job sizes), while their variability has a huge impact on response times in open systems, it has much less of an effect in closed models. The impact of these principles is that a system designer needs to beware of taking results that were discovered under one system model and applying them to a second system model. For example, if the workload generator being used creates a closed system model, whereas the real world application is closer to an open system model, then the results obtained using the workload generator may be far from those witnessed in practice.

The second set of principles (see Section 8.4.2) deal with the contrasting *impact of scheduling* in closed and open systems. As we have seen throughout this thesis, scheduling is a common mechanism for improving mean response time without purchasing additional resources. When system designers seek to evaluate a

new scheduling policy, they often test its effectiveness using a workload generator and simulation test-bed. We illustrate that one must be very careful to correctly model the application workload as closed or open, since the impact of scheduling turns out to be very different under these two models. For example, our principles show that favoring short jobs is highly effective in improving mean response time in open systems, while this is far less true under a closed system model. We find that closed system models only benefit from scheduling under a narrow range of parameters, when load is moderate and the MPL is very high. The message for system designers is that understanding whether the workload is better modeled with an open or closed system is essential in determining the effectiveness of scheduling.

The third set of principles (see Section 8.5) deal with *partly-open systems*. Our principles specify those parameter settings for which the partly-open model behaves more like a closed model or more like an open model with respect to response time. We also find that, counter to intuition, parameters like think time have almost no impact on the performance of a partly-open model. The principles describing the behavior of partly-open system models are important because real-world applications often fit best into partly-open models, and the performance of these models is not well understood. In particular, the effect of system parameters and scheduling on performance in the partly open system – points which our principles address – are not known. Our results motivate the importance of designing *versatile* workload generators that are able to support open, closed, and partly open system models. We create such versatile workload generators for several common systems, including web servers and database systems, and use these throughout our studies.

The third set of principles also provides system designers with guidelines for *how to choose a system model* when they are forced to pick a workload generator that is either purely closed or purely open, as are almost all workload generators (see Section 8.6). We consider ten different workloads and use our principles to determine for each workload which system model is most appropriate for that workload: closed, open, or partly-open. To the best of our knowledge, no such guide exists for systems researchers. Yet given the tremendous impact of the system model on performance, as described above, it is critical that one take care to make this decision carefully.

The chapter is organized as follows. We begin by presenting more detailed introductions to the closed, open, and partly-open system models in Section 8.1. Then, in Section 8.2 we provide details on our methodology for comparing the closed, open, and partly-open models. Next, we provide details on the case studies in Section 8.3. We present some results contrasting the three system models in Section 8.3, but we provide far more detailed studies of the open and closed models in Section 8.4 and of the partly-open model in Section 8.5. Next, using the results in Sections 8.4 and 8.5, we present a procedure for choosing the appropriate system model for a given workload in Section 8.6. Finally, we conclude in Section 8.7.

8.1 Defining closed, open, and partly-open systems

Before we can begin our comparison of the open, closed, and partly-open systems, we will first provide a more detailed description of the models.

The open model

Figure 8.1(a) depicts an open system configuration. In an open system model there is a stream of arriving users with average arrival rate λ. Each user is assumed to submit one job to the system, wait to receive the

Type of benchmark	Name	System model
Model-based web workload generator	Surge [29], WaspClient [153], Geist [106], WebStone [229], WebBench [231], MS Web Capacity Analysis Tool [143]	Closed
	SPECWeb96 [219], WAGON [129]	Open
Playback mechanisms for HTTP request streams	MS Web Application Stress Tool [144], Webjamma [1], Hammerhead [214], Deluge [213], Siege [80]	Closed
	httperf [151], Sclient [22]	Open
Proxy server benchmarks	Wisconsin Proxy Benchmark [11], Web Polygraph [194], Inktomi Climate Lab [84]	Closed
Database benchmark for e-commerce workloads	TPC-W [228]	Closed
Auction web site benchmark	RUBiS[13]	Closed
Online bulletin board benchmark	RUBBoS[13]	Closed
Database benchmark for online transaction processing (OLTP)	TPC-C [227]	Closed
Model-based packet level web traffic generators	IPB (Internet Protocol Benchmark) [132], GenSyn [101] WebTraf [73], trafgen [53]	Closed
	NS traffic generator [254]	Open
Mail server benchmark	SPECmail2001 [218]	Open
Java Client/Server benchmark	SPECJ2EE [217]	Open
Web authentication and authorization	AuthMark [146]	Closed
Network file servers	NetBench [230]	Closed
	SFS97_R1 (3.0) [216]	Open
Streaming media service	MediSyn [224]	Open

Table 8.1: *A summary table of the system models underlying standard web related workload generators.*

response, and then leave. The number of users queued or running at the system at any time is unbounded. The differentiating feature of an open system is that a *request completion does not trigger a new request: a new request is only triggered by a new user arrival.* As before, *response time*, T, is defined as the time from when a request is submitted until it is completed. The *server load* is defined as the fraction of time that the server is busy. Here load, ρ, is the product of the mean arrival rate of requests, λ, and the mean service demand $E[X]$. Note that the throughput in an open model is always equal to the arrival rate.

The closed model

Figure 8.1(b) depicts a closed system configuration. In a closed system model, it is assumed that there is some fixed number of users, who use the system forever. This number of users is typically called the *multiprogramming level* (MPL) and denoted by N. Each of these N users repeats these 2 steps, indefinitely: (a) submit a job, (b) receive the response and then "think" for some amount of time. In a closed system, *a new request is only triggered by the completion of a previous request.* At all times there are some number of users, N_{think}, who are thinking, and some number of users N_{system}, who are either running or queued to run in the system, where $N_{think} + N_{system} = N$. The *response time*, T, in a closed system is defined to be

the time from when a request is submitted until it is received. In the case where the system is a single server (e.g. a web server), the *server load*, denoted by ρ, is defined as the fraction of time that the server is busy, and is the product of the mean throughput and the mean service demand (processing requirement) $E[X]$.

The partly-open model

Neither the open system model nor the closed system model is entirely realistic for computer systems. Consider for example a web site. On the one hand, a user is apt to make more than one request to a web site, and the user will typically wait for the output of the first request before making the next. In these ways a closed system model makes sense. On the other hand, the number of users at the site varies over time; there is no sense of a fixed number of users N. The point is that users visit to the web site, behave as if they are in a closed system for a short while, and then leave the system.

Motivated by the example of a web site, we also study a more realistic alternative to the open and closed system configurations: the partly-open system shown in Figure 8.1(c). Under the partly-open model, users arrive according to some outside arrival process as in an open system. However, every time a user completes a request at the system, with probability p the user stays and makes a followup request (possibly after some think time), and with probability $1 - p$ the user simply leaves the system. Thus the expected number of requests that a user makes to the system in a visit is Geometrically distributed with mean $1/(1 - p)$. We refer to the collection of requests a user makes during a visit to the system as a *session* and we define the *length* of a session to be the number of requests in the session/visit. The *server load* is the fraction of time that the server is busy equalling the product of the average outside arrival rate λ, the mean number of requests per visit, and the mean service demand. For a given load, when p is small, the partly-open model is more similar to an open model. For large p, the partly-open model resembles a closed model.

8.2 Comparison methodology

Given the many differences between the closed, open, partly-open systems, it is important when comparing the systems that we configure each model in a way that provides a "fair" comparison. In this section, we discuss the relevant parameters and metrics for the three system models and discuss how we set parameters in order to compare the system models fairly.

Throughout this chapter we choose the service demand distribution to be the same for the open , closed, and partly-open systems. In the case studies the service demand distribution is either taken from a trace or determined by the workload generator used in the experiments. In the model-based simulation experiments later in the chapter, we use hyperexponential service demands, in order to capture the highly variable service distributions in web applications. Throughout, we measure the variability in the service demand distribution using the squared coefficient of variation, C^2. The think time in the closed system and partly-open systems, Z, follows an exponential distribution, and the arrival process in the open and partly-open systems is either a Poisson arrival process with average rate λ, or provided by traces.[1] The results for all simulations and experiments are presented in terms of mean response times and the system load ρ. While we do not explicitly report numbers for another important metric, mean throughput, the interested reader can directly infer those

[1] Note that we choose a Poisson arrival process (i.e. exponential inter-arrival times) and exponential think times in order to allow the open, closed, and partly-open system configurations to be as parallel as possible. This setting underestimates the differences between the systems when more bursty arrival processes are used.

numbers by interpreting load as a simple scaling of throughput in all three system models.

In order to fairly compare the open, closed, and partly-open systems, we will hold the system load ρ for the three systems equal, and study the effect of the system model on mean response time. The load in the open system is specified by λ, since $\rho = \lambda E[X]$. The load in the partly-open system is specified by λ and p. Thus, for a fixed p, we adjust the load by adjusting λ. Note that the think time in a partly-open system does not affect the load. Fixing the load of a closed system is more complex, since the load is affected by many parameters including the MPL, the think time, the service demand variability, and the scheduling policy. The fact that system load is influenced by many more system parameters in a closed system than in an open or partly-open system is an interesting difference between the systems. Throughout, we will achieve a desired system load in a closed system by adjusting the think time (see Figure 8.6(b)) since this parallels the notion of varying the interarrival time that is used in the open and partly-open models.

8.3 Real-world case studies

We can now begin to compare the closed, open, and partly-open models. In this section, we compare the three system models in the context of four different real-world applications. The applications include (a) a web server delivering static content in a LAN environment, (b) the database back-end at an e-commerce web site, (c) the application server at an auctioning web site, (d) scheduling at a supercomputing center, and (e) a web server delivering static content in a WAN environment. These applications vary in many respects, including the bottleneck resource, the workload properties (e.g. job size variability), network effects, and the types of scheduling policies considered. We study applications (a), (b), and (e) through full implementation in a real test-bed, while our study of applications (c) and (d) relies on trace-based simulation.

An integral part of these case studies is the development of a set of workload generators, simulators, and trace analysis tools that facilitate experimentation with all three system models: open, closed, and partly-open. For implementation-based case studies we extend the existing workload generator for each system (which is based on only one system model) to enable all three system models. For the case studies based on trace-driven simulation, we implement a versatile simulator that models open, closed, and partly-open systems and takes traces as input. We also develop tools for analyzing web traces (in Common Logfile Format or Squid log format) to extract the data needed to parameterize workload generators and simulators.

Sections 8.3.1 – 8.3.5 provide the details of the case studies. The main results are shown in Figures 8.2 and 8.4. For each case study we first explain the tools developed for experimenting in open, closed, and partly-open models. We then then describe the relevant scheduling policies and their implementation, and finally we discuss the results. *The discussions at the end of the case studies are meant only to highlight the key points; we will discuss the differences between open, closed, and partly-open systems and the impact of these differences in much more detail in Sections 8.4 and 8.5.*

8.3.1 Static web content

Our first case study is an Apache web server running on Linux and serving static content, i.e. requests of the form "Get me a file," in a LAN environment. Our experimental setup involves six machines connected by a 10/100 Ethernet switch. Each machine has an Intel Pentium III 700 MHz processor and 256 MB RAM, and runs Linux. One of the machines is designated as the server and runs Apache. The others generate web

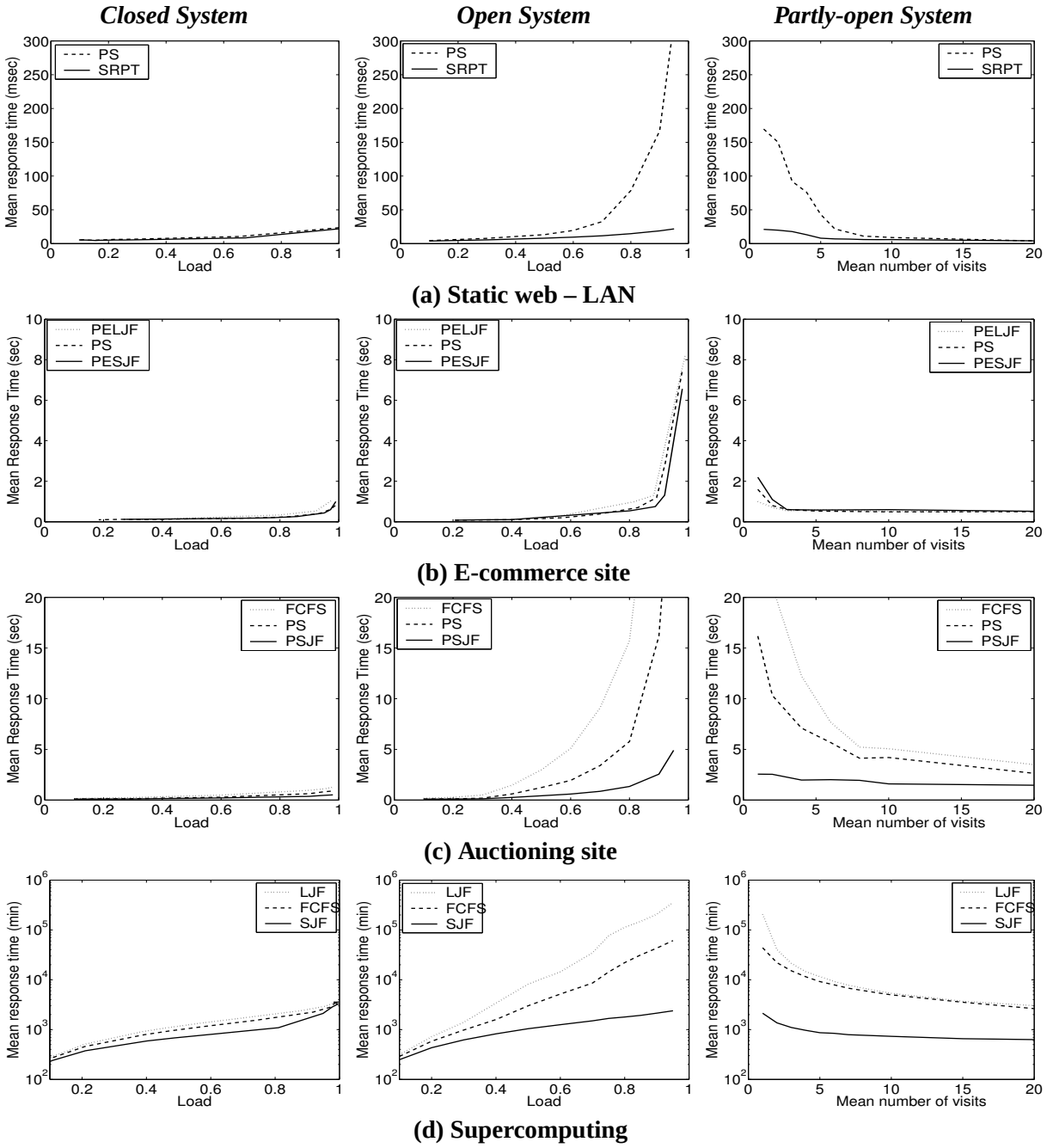


Figure 8.2: *Implementation and simulation results for real-world case studies. Each row shows the results for a real-world workload and each column shows the results for one of the system models. In all experiments with the closed system model the MPL is 50. The partly-open system is run at fixed load* 0.9.

requests based on a web trace.

Workload generation

In this case study we generate static web workloads based on a trace. Below we first describe our workload generator which generates web requests following an open, closed, or partly-open model. We then describe the tool for analyzing web traces that produces input files needed by the workload generator. Finally we briefly describe the actual trace that we are using in our work.

Our workload generator is built on top of the Sclient[22] workload generator. The Sclient workload generator uses a simple open system model, whereby a new request for file y is made exactly every x msec. Sclient is designed as a single process that manages all connections using the `select` system call. After each call to `select`, Sclient checks whether the current x msec interval has passed and if so initiates a new request. We generalize Sclient in several ways.

For the open system, we change Sclient to make requests based on arrival times and filenames specified in an input file. The entries in the input file are of the form $< t_i, f_i >$, where t_i is a time and f_i is a file name.

For the closed system, the input file only specifies the names of the files to be requested. To implement closed system arrivals in Sclient, we have Sclient maintain a list with the times when the next requests are to be made. Entries to the list are generated during runtime as follows: Whenever a request completes, an exponentially distributed think time Z is added to the current time t_{curr} and the result $Z + t_{curr}$ is inserted into the list of arrival times.

In the case of the partly-open system, each entry in the input file now defines a *session*, rather than an individual request. An entry in the input file takes the form $< t_i, f_{i_1}, \ldots, f_{i_n} >$ where t_i specifies the arrival time of the session and $< f_{i_1}, \ldots, f_{i_n} >$ is the list of files to be requested during the session. As before, a list with arrival times is maintained according to which requests are made. The list is initialized with the session arrival times t_i from the input file. To generate the arrivals within a session, we use the same method as described for the closed system above: after request $f_{i_{j-1}}$ completes we arrange the arrival of request f_{i_j} by adding an entry containing the arrival time $Z + t_{curr}$ to the list, where t_{curr} is the current time and Z is an exponentially distributed think time.

All the input files for the workload generator are created based on a web trace. We modify the Webalizer tool [44] to parse a web trace and then extract the information needed to create the input files for the open, closed, and partly-open system experiments. In the case of the open system, we simply output the arrival times together with the names of the requested files. In the case of the closed system, we only extract the sequence of file names. Creating the input file for the partly-open system is slightly more involved since it requires identifying the sessions in a trace. A common approach for identifying sessions (and the one taken by Webalizer) is to group all successive requests by the same client (i.e. same IP address) into one session, unless the time between two requests exceeds some timeout threshold in which case a new session is started. In our experiments, we use the timeout parameter to specify the desired average session length.

The trace we use consists of one day from the 1998 World Soccer Cup, obtained from the Internet Traffic Archive [103]. The details of the trace are summarized in Table 8.2. Note that virtually all requests in this trace are *static*.

Number of Req.	Mean job size	Job size variability (C^2)	Min job size	Max job size
$4.5 \cdot 10^6$	5KB	96	41 bytes	2MB

Table 8.2: *Summary statistics for the trace used in the static web case study.*

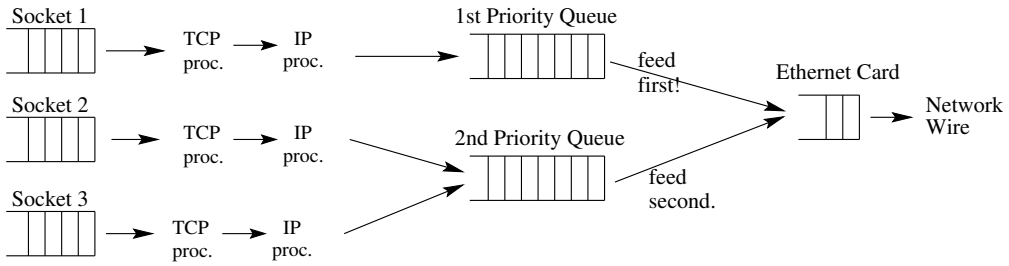


Figure 8.3: *Flow of data in Linux with* SRPT*-like scheduling (only 2 priority levels shown).*

Scheduling

Standard scheduling of static requests in a web server is best modeled by processor sharing (PS). However, recent research suggests favoring requests for small files can improve mean response times at web servers [96]. In this section we therefore consider both PS and SRPT policies.

We have modified the Linux kernel and the Apache Web server to implement SRPT scheduling at the server. For static HTTP requests, the network (access link out of the server) is typically the bottleneck resource. Thus, our solution schedules the bandwidth on this access link by controlling the order in which the server's socket buffers are drained. Traditionally, the socket buffers are drained in Round-Robin fashion (similar to PS); we instead give priority to sockets corresponding to connections where the remaining data to be transferred is small. Figure 8.3 shows the flow of data in Linux after our modifications.

There are multiple priority queues and queue i may only drain if queues 0 to $i-1$ are empty. The implementation is enabled by building the Linux kernel with support for the user/kernel Netlink Socket, QOS and Fair Queuing, and the Prio Pseudoscheduler and by using the `tc`[12] user space tool. We also modify Apache to use `setsockopt` calls to update the priority of the socket as the remaining size of the transfer decreases. For more details on our implementation see [96, 206].

Synopsis of results

Figure 8.2(a) shows results from the the static web implementation under closed, open, and partly open workloads in a LAN environment. Upon first glance, it is immediately clear that the closed system response times are vastly different from the open response times. In fact, the response times in the two systems are orders of magnitude different under PS given a common system load. Furthermore, SRPT provides little improvement in the closed system, while providing dramatic improvement in the open system.

The third column of Figure 8.2(a) shows the results for the partly-open system. Notice that when the mean number of requests is small, the partly-open system behaves very much like the open system. However, as the mean number of requests grows, the partly-open system behaves more like a closed system. Thus, the impact of scheduling (e.g. SRPT over PS) is highly dependent on the number of requests in the partly-open

system.

8.3.2 E-commerce site

Our second case study considers the database back-end server of an e-commerce site, e.g. an online bookstore. We use a PostgreSQL[177] database server running on a 2.4-GHz Pentium 4 with 3GB RAM, running Linux 2.4, with a buffer pool of 2GB. The machine is equipped with two 120GB IDE drives, one used for the database log and the other for the data. The workload is generated by four client machines having similar specifications to the database server connected via a network switch.

Workload generation

The workload for the e-commerce case study is based on the TPC-W [228] benchmark, which aims to model an online bookstore such as Amazon.com. We build on the TPC-W kit provided by the Pharm project [49]. The kit models a closed system (in accordance with TPC-W guidelines) by creating one process for each client in the closed system. Each process submits a request to the database, waits for the response, sleeps for an exponentially distributed think time, and then makes the next request.

We extend the kit to also support an open system with Poisson arrivals, and a partly-open system. We do so by creating a master process that signals a client whenever it is time to make a new request in the open system or to start a new session in the partly-open system. The master process repeats the following steps in a loop: it sleeps for an exponential interarrival time, signals a client, and draws the next inter-arrival time. The clients block waiting for a signal from the master process. In the case of the open system, after receiving the signal, the clients make one request before they go back to blocking for the next signal. In the case of the partly-open system, after receiving a signal, the clients generate a session by executing the following steps in a loop: (1) make one request; (2) flip a coin to decide whether to begin blocking for a signal from the master process or to generate an exponential think time and sleep for that time.

TPC-W consists of 16 different transaction types including the "ShoppingCart" transaction, the "Payment" transaction, and others. Statistics of our configuration are as shown in Table 8.3.

Database size	Mean job size	Job size variability (C^2)	Min job size	Max job size
3GB	101 ms	4	2 ms	5s

Table 8.3: *Summary statistics for the trace used in the e-commerce case study.*

Scheduling

The bottleneck resource in our setup is the CPU, as observed in [138]. The default scheduling policy is therefore best described as PS, in accordance with Linux CPU scheduling. Note that in this application, exact service demands are not known, so SRPT cannot be implemented. Thus, we experiment with PESJF and PELJF policies where the expected service demand of a transaction is based on its type. The "Bestseller" transaction, which makes up 10% of all requests, has on average the largest service demand. Thus, we study 2-priority PESJF and PELJF policies where the "Bestseller" transactions are "expected to be long" and all other transactions are "expected to be short."

To implement the priorities needed for achieving PESJF and PELJF, we modify our PostgreSQL server as follows. Simple CPU prioritization of a process using Linux "Nicing" will just increase the time quantum of the prioritized process, but will not give absolute priority to it. We therefore base our implementation on the real-time scheduler that Linux provides. We use the `sched_setscheduler()` system call to set the scheduling class of a PostgreSQL process working on a high priority transaction to "SCHED_RR," which marks a process as a Linux real-time process. We leave the scheduling class of a low priority process at the standard "SCHED_OTHER." Real-time processing in Linux always has absolute, preemptive priority over standard processes. For more details on the implementation, see [139, 206]

Synopsis of results

Figure 8.2(b) shows results from the e-commerce implementation described above. Again, the difference in response times between the open and closed systems is immediately apparent – the response times of the two systems differ by orders of magnitude. Interestingly, because the variability of the service demands is much smaller in this workload than in the static web workload, the impact of scheduling in the open system is much smaller. This also can be observed in the plot for the partly open system: even when the number of requests is small, there is little difference between the response times of the different scheduling policies.

8.3.3 Auctioning web site

Our third case study investigates an auctioning web site. This case study uses simulation based on a trace from one of the top-ten U.S. online auction sites.

Workload generation

For simulation-based case studies we implement a simulator that supports open, closed, and partly-open arrival processes which are either created based on a trace or are generated from probability distributions. For a trace-based arrival process the simulator expects the same input files as the workload generator described in Section 8.3.1. If no trace for the arrival process is available the simulator alternatively offers (1) open system arrivals following a Poisson process; (2) closed system arrivals with exponential think times; (3) partly-open arrivals with session arrivals following a Poisson process and think times within the sessions being exponentially distributed. The service demands can either be specified through a trace or one of several probability distributions, including hyper-exponential distributions and more general distributions.

For our case study involving an auctioning web site we use the simulator and a trace containing the service demands obtained from one of the top ten online auctioning sites in the US. No data on the request arrival process is available. The characteristics of the service demands recorded in the trace are summarized in Table 8.4.

Number of jobs	Mean job size	Job size variability (C^2)	Min job size	Max job size
300000	0.09s	9.19	0.01s	50s

Table 8.4: *Summary statistics for the trace used in the auctioning case study.*

Scheduling

The policy used in a web site serving dynamic content, such as an auctioning web site, is best modeled by PS. To study the effect of scheduling in this environment we additionally simulate FCFS and PSJF.

Synopsis of results

Figure 8.2(c) shows results from the auctioning trace-based case study described above. The plots here illustrate the same properties that we observed in the case of the static web implementation. In fact, the difference between the open and closed response times is extreme, especially under FCFS. As a result, there is more than a factor of ten improvement of PSJF over FCFS (for $\rho > 0.7$), whereas there is little difference in the closed system.

This effect can also be observed in the partly-open system, where for a small number of requests per session the response times are comparable to those in the open system and for a large number of requests per session the response times are comparable to those in the closed system. The actual convergence rate depends on the variability of the service demands (C^2). In particular, the e-commerce case study (low C^2) converges quickly, while the static web and auctioning case studies (higher C^2) converge more slowly.

8.3.4 Supercomputing center

In this section we model the Cray J90 and Cray C90 machines at the Pittsburgh Supercomputing Center (PSC)[2]. These servers have between 4 and 16 processors and typically execute exactly one job at a time. The jobs are run-to-completion, i.e. no preemption or timesharing.

Workload generation

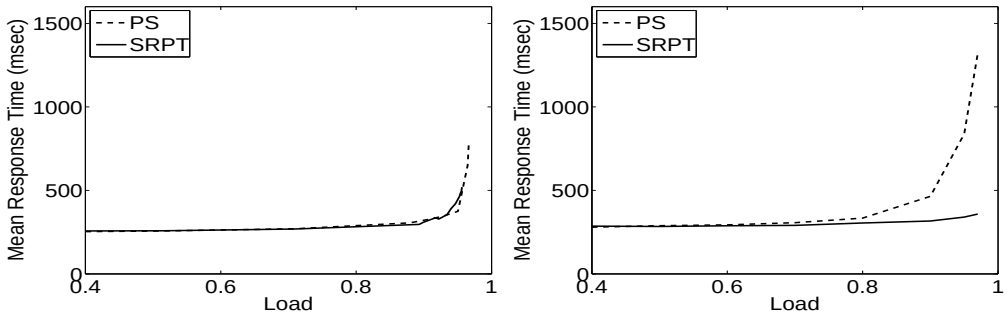
To simulate the workload for the supercomputing case study we use our simulator described in Section 8.3.3 and two traces that we obtained from the PSC. The traces were collected from January through December 1997 and contain the service demands for a total of more than 50,000 jobs. The statistics of the traces are summarized in Table 8.5. Note the high variability in the workloads.

System	Number of jobs	Mean job size	Job size variability (C^2)	Min job size	Max job size
C90	54962	4562.6s	43.16	1s	$2.22e6$s
J90	3582	9448.6s	10.02	4s	$0.61e6$s

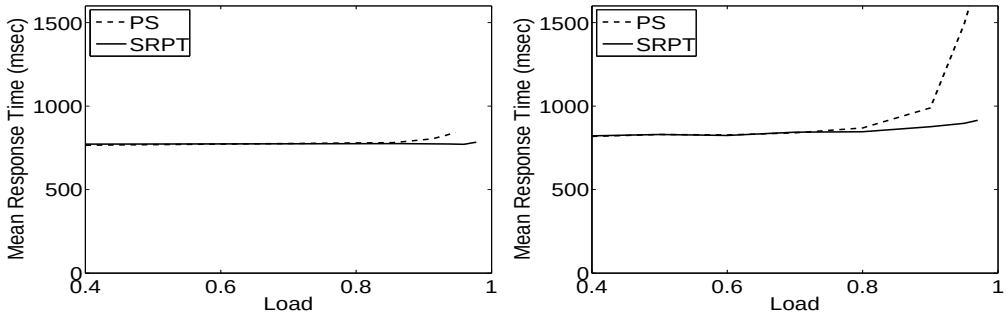
Table 8.5: *Summary statistics for the trace used in the supercomputing case study.*

Scheduling

The Cray J90 and Cray C90 machines can only be scheduled in a non-preemptive fashion. In addition to FCFS, which is the standard non-preemptive policy, we consider two size-based policies that are of interest in this setting. The Non-preemptive-Shortest-Job-First (SJF) policy gives preference to the job with the smallest service demand, while the Non-preemptive-Longest-Job-First policy (LJF) favors the longest one.



(a) Static web – Good WAN conditions



(b) Static web – Poor WAN conditions

Figure 8.4: *Effect of WAN conditions in the static web case study. The top row shows results for good WAN conditions (average RTT=50ms, loss rate=1%) and the bottom row shows results for poor WAN conditions (average RTT=100ms, loss rate=4%). In both cases the closed system has an MPL of 200. Note that, due to network effects, the closed system cannot achieve a load of 1, even when think time is zero. Under the settings we consider here, the max achievable load is* ≈ 0.98.

Synopsis of results

Figure 8.2(d) shows results from the super-computing trace-based case study for the PSC C90 workload. The same trends we have observed in the first three case studies are again prominent here. The closed system response times are orders of magnitude lower for the closed system than the open system. Furthermore, scheduling is far less effective for the closed systems than the open ones. Looking more closely at the closed systems, we see that scheduling is only significant in certain regions of moderate load, even under this highly variable workload. By contrast, for open systems the high variability of the workload results in orders of magnitude disparity between the scheduling policies.

In the partly open system, across all applications, when the mean number of requests per session is small, the system behaves very much like the open system; as the mean number of requests per session grows, the partly-open system behaves more like a closed system. The actual convergence rate depends on the variability of the service demands (C^2). In particular, the e-commerce case study (low C^2) converges quickly, while the supercomputing case study (high C^2) converges slowly.

8.3.5 Study of WAN effects

To study the effect of network conditions, we return to the case of static web requests (Section 8.3.1), but this time we include the emulation of network losses and delays in the experiments.

Workload generation

The setup and workload generation is identical to the case study of static web requests (Section 8.3.1), except that we add functionality for emulating WAN effects as follows. We implement a separate module for the Linux kernel that can drop or delay incoming and outgoing TCP packets (similarly to Dummynet [190] for FreeBSD). More precisely, we change the `ip_rcv()` and the `ip_output()` functions in the Linux TCP-IP stack to intercept in- and out-going packets to create losses and delays. In order to delay packets, we use the `add_timer()` facility to schedule the transmission of delayed packets. We recompile the kernel with `HZ=1000` to get a finer-grained millisecond timer resolution. In order to drop packets, we use an independent, uniform random loss model which can be configured to a specified probability, as in Dummynet.

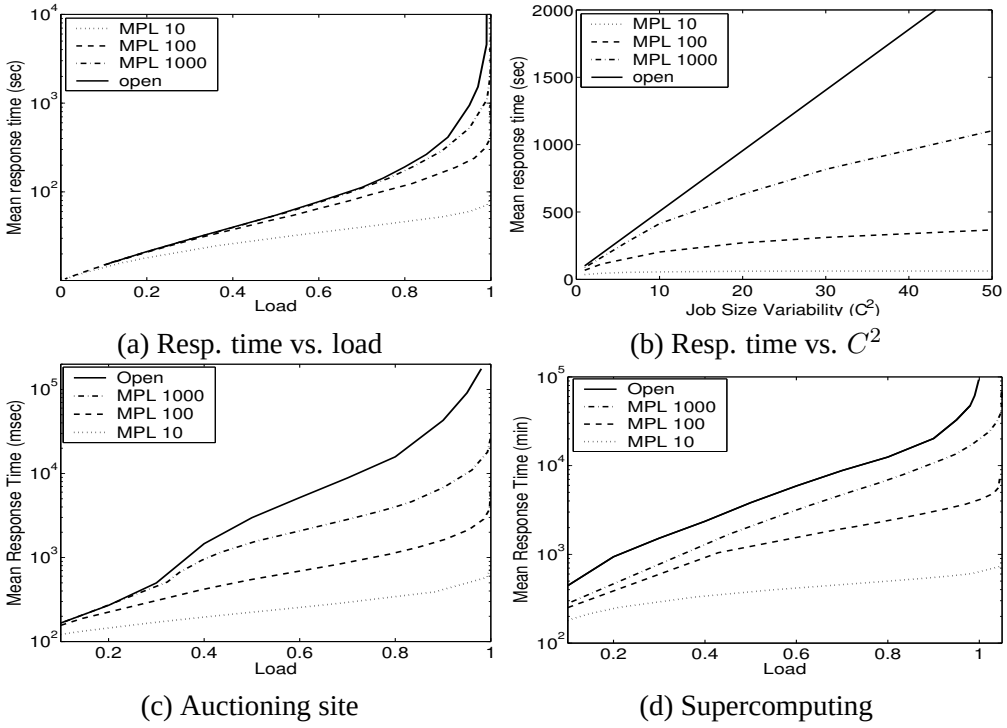
Synopsis of results

Figures 8.4 compares the response times of the closed and the open systems under (a) relatively good WAN conditions (50ms RTT and 1% loss rate) and under (b) poor WAN conditions (100ms RTT and 4% loss rate).

We find that under WAN conditions the differences between the open and closed systems are smaller (proportionally) than in a LAN (Figure 8.2 (a)), however, they are still significant for high server loads (load > 0.8). The reason that the differences are smaller in WAN conditions is that response times include network overheads (network delays and losses) in addition to delays at the server. These overheads affect the response times in the open and closed system in the same way, causing the proportional differences between open and closed systems to shrink. For similar reasons, scheduling has less of an effect when WAN effects are strong, even in the case of an open system. SRPT improves significantly over PS only for high loads, and even then the improvement is smaller than in a LAN. The reason is that scheduling changes only the time spent at the server, not network delays and losses. Therefore, it is effective when server delays dominate response time, which does not happen under low loads when WAN effects are strong.

8.4 Open versus closed systems

The case studies in the previous section have illustrated the dramatic impact of the system model in practice; however, more experimentation is needed in order to understand the reasons for these differences. In this section, we will develop principles that help explain both the differences between the open and closed system and the impact of these differences with respect to scheduling. In addition to the case studies that we have already discussed, we will use model-based simulations in order to provide more control over system parameters, such as job size variability, that are fixed in the case studies. We start with the simple case of FCFS scheduling and then move to more complicated scheduling policies.



(a) Resp. time vs. load (b) Resp. time vs. C^2

(c) Auctioning site (d) Supercomputing

Figure 8.5: *Model and trace-based simulation results showing mean response time as a function of load and service demand variability under* FCFS *scheduling. (a) and (b) use model based simulation, while (c) and (d) uses trace-based simulation. In all cases, the solid line represents an open system and the dashed lines represent closed systems with different MPLs. The load is adjusted via the think time in the closed system, and via the arrival rate in the open system. In the model-based simulations,* $E[X] = 10$*. In (a) we fix* $C^2 = 8$ *and in (b) we fix* $\rho = 0.9$*.*

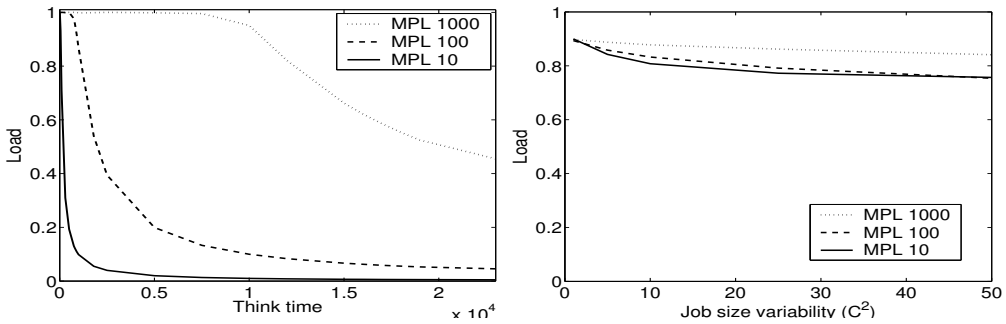
8.4.1 FCFS

Our study of the simple case of FCFS scheduling will illustrate three principles that we will exploit when studying more complex policies.

Principle (i): *For a given load, mean response times are significantly lower in closed systems than in open systems.*

Principle (i) is maybe the most noticeable performance issue differentiating open and closed systems in our case studies (Figure 8.2). We bring further attention to this principle in Figure 8.5 due to its importance for the vast literature on capacity planning, which typically relies on closed models, and hence may underestimate the resources needed when an open model is more appropriate.

For fixed high loads, the response time under the closed system is *orders of magnitude* lower than those for the open system. While Schatte [198, 199] has proven that, under FCFS, the open system will always serve as an upper bound for the response time of the closed system, the magnitude of the difference in



(a) Think time vs. load (b) Variability vs. load

Figure 8.6: *Model-based simulation results illustrating how the service demand variability, the MPL, and the think time can affect the system load in a closed system. These plots use* FCFS *scheduling, however results are parallel under other scheduling policies.*

practical settings has not previously been studied. Intuitively, this difference in mean response time between open and closed systems is a consequence of the fixed MPL, N, in closed systems, which limits the queue length seen in closed systems to N even under very high load. By contrast, no such limit exists for an open system.

Principle (ii): *As the MPL grows, closed systems become open, but convergence is slow for practical purposes.*

Principle (ii) is illustrated by Figure 8.5. We see that as the MPL, N, increases from 10 to 100 to 1000, the curves for the closed system approach the curves for the open system. Schatte [198, 199] proves formally that as N grows to infinity, a closed FCFS queue converges to an open M/GI/1/FCFS queue. What is interesting however, is how slowly this convergence takes place. When the service demand has high variability (C^2), a closed system with an MPL of 1000 still has much lower response times then the corresponding open system. Even when the job service demands are lightly variable, an MPL of 500 is required for the closed system to achieve response times comparable to the corresponding open system. Further, the differences are more dramatic in the case-study results than in the model-based simulations.

We can explain the convergence of a closed system to an open one as N gets very large. In the open M/GI/1 system, the arrival process is Poisson and thus has exponential interarrival times each having a constant rate, which is independent of the completions at the server. In the closed system, the interarrival times are governed by the exponential think times; however, the rate changes with each job completion. When N is small, the rate can change drastically. When N is large though, there will likely be many jobs "thinking" at any given point in the closed system. So a completion, which increments $N_{thinking}$ by one, has very little effect on the arrival rate. As N goes to infinity, the effect of a completion on the arrival rate disappears completely and the closed arrival process matches the open arrival process.

This principle impacts the choice of whether an open or closed system model is appropriate. One might think that an open system is a reasonable approximation for a closed system with a high MPL; however, though this can be true in some cases, the closed and open system models may still behave significantly differently if the service demands are highly variable.

Principle (iii): *While variability has a large effect in open systems, the effect is much smaller in closed systems.*

This principle is difficult to see in the case-study figures (Figure 8.2) since each trace has a fixed variability. However, it can be observed by comparing the magnitude of disparity between the e-commerce site results (low variability) and the others (high variability). This principle is most significant for the supercomputing workload, where C^2 can be very high.

Using simulations, we can study this effect directly. Figure 8.5(b) compares open and closed systems under a *fixed load* $\rho = 0.9$, as a function of the service demand variability C^2. For an open system, we see that C^2 directly affects mean response time. This is to be expected since high C^2, under FCFS service, results in short jobs being stuck behind long jobs, increasing mean response time. In contrast, for the closed system with MPL 10, C^2 has comparatively little effect on mean response time. This is counterintuitive, but can be explained by observing that for lower MPL there are *fewer* short jobs stuck behind long jobs in a closed system, since the number of jobs in the system (N_{system}) is bounded. As MPL is increased, C^2 can have more of an effect, since N_{system} can be higher.

It is important to point out that by holding the load constant in Figure 8.5(b), we are actually performing a conservative comparison of open and closed systems. If we didn't hold the load fixed as we changed C^2, increasing C^2 would result in a slight drop in the load of the closed system as shown in Figure 8.6(b). This slight drop in load, would cause a drop in response times for closed systems, whereas there is no such effect in open systems.

8.4.2 The impact of scheduling

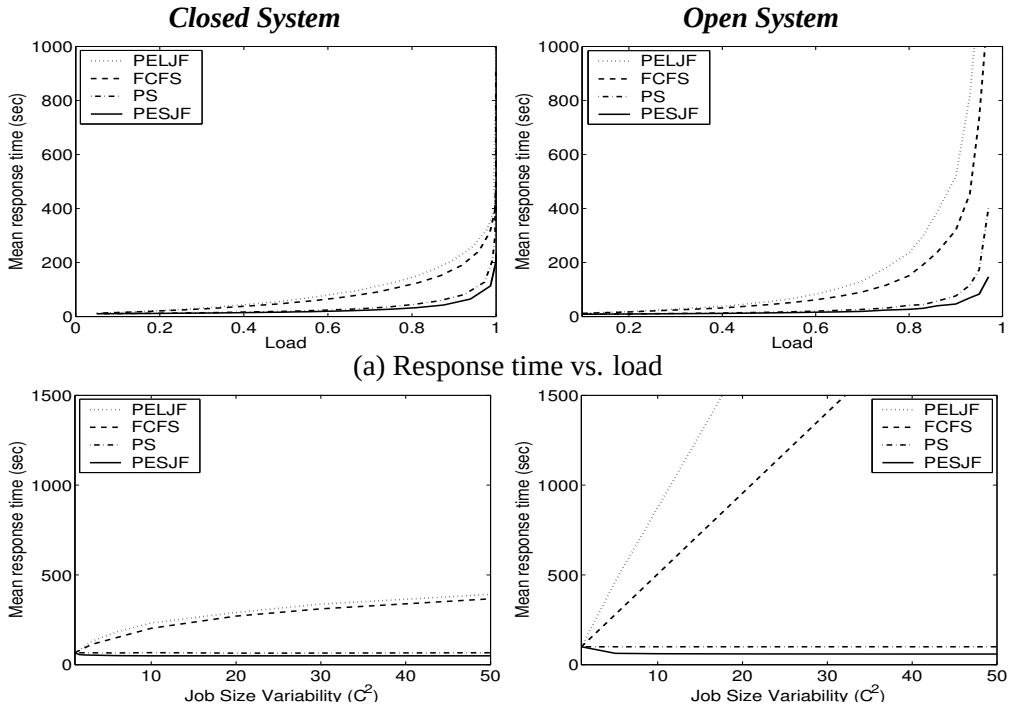
The value of scheduling in open systems is understood and cannot be overstated. In open systems, there are order of magnitude differences between the performance of scheduling policies because scheduling can prevent small jobs from queueing behind large jobs. In contrast, scheduling in closed systems is not well understood.

Principle (iv): *While open systems benefit significantly from scheduling with respect to response time, closed systems improve much less.*

Principle (v): *Scheduling only significantly improves response time in closed systems under very specific parameter settings: moderate load (think times) and high MPL.*

Figure 8.2 illustrates the fundamentally different behavior of mean response time in the open and closed systems in realistic settings. In Figure 8.7, we further study this difference as a function of (a) load and (b) variability using simulations. Under the open system, as load increases, the disparity between the response times of the scheduling policies grows, eventually differing by orders of magnitude. In contrast, at both high and low loads in the closed system, the scheduling policies all perform similarly; only at moderate loads is there a significant difference between the policies – and even here the differences are only a factor of 2.5. Another interesting point is that, whereas for FCFS the mean response time of an open system bounded that in the corresponding closed system from above, this does not hold for other policies such as PESJF, where the open system can result in lower response times than the closed system.

We can build intuition for the limited effects of scheduling in closed systems by first considering a closed feedback loop with no think time. In such a system, surprisingly, the scheduling done at the queue



(a) Response time vs. load

(b) Response time vs. variability

Figure 8.7: *Model-based simulation results illustrating the different effects of scheduling in closed and open systems. In the closed system the MPL is 100, and in both systems the service demand distribution has mean 10. For the two figures in (a) C^2 was fixed at 8 and in the two figures in (b) the load was fixed at 0.9.*

is inconsequential – all work conserving scheduling policies perform equivalently. To see why, note that in a closed system Little's Law states that N is the product of the mean throughput and $E[T]$, where N is the constant MPL across policies. We will now explain why the mean throughput is constant across all work conserving scheduling policies (when think time is 0), and hence it will follow that $E[T]$ is also constant across scheduling policies. The mean throughput is the long-run average rate of completions. Since a new job is only created when a job completes, over a long period of time, all work conserving scheduling policies will complete the same set of jobs plus or minus the initial set N. As time goes to infinity, the initial set N becomes unimportant; hence the mean throughput is constant. This argument does not hold for open systems because for open systems Little's Law states that $E[N] = \lambda E[T]$, and $E[N]$ is not constant across scheduling policies.

Under closed systems with think time, we now allow a varying number of jobs in the queue, and thus there is some difference between scheduling policies. However, as think time grows, load becomes small and so scheduling has less effect.

Throughout Figure 8.7, the MPL is held constant at 100. Recall from Principle (ii) in Section 8.4.1 that the effect of MPL is to transition between open and closed systems. Thus, under smaller MPL, the effects of scheduling are even less noticeable in the closed system; however, for larger MPL scheduling can have a larger effect.

A very subtle effect, not yet mentioned, is that in a closed system the scheduling policy actually affects the throughput, and hence the load. "Good" policies, like PESJF, increase throughput, and hence load, slightly (less than 10%). Had we captured this effect (rather than holding the load fixed), the scheduling policies in the closed system would have appeared even closer, resulting in even starker differences between the closed and open systems.

The impact of Principles (iv) and (v) is clear. For closed systems, scheduling provides small improvement across all loads, but can only result in substantial improvement when load (think time) is moderate. In contrast, scheduling always provides substantial improvements for open systems.

Principle (vi): *Scheduling can limit the effect of variability in both open and closed systems.*

For both the open and closed systems, better scheduling (PS and PESJF) helps combat the effect of increasing variability, as seen in Figure 8.7. The improvement; however, is less dramatic for closed systems due to Principle (iii) in Section 8.4.1, which tells us that variability has less of an effect on closed systems in general.

8.5 Partly-open systems

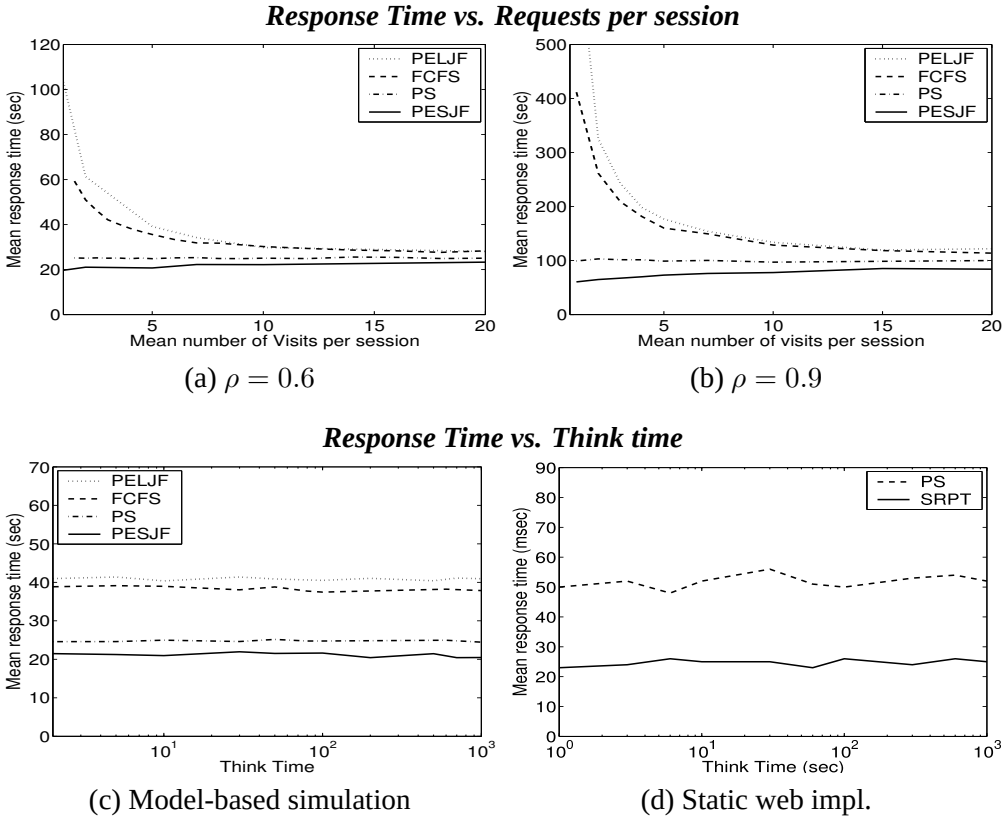
So far we have limited our comparisons primarily to the open and closed system models. We now begin our discussion of the partly-open model. The partly-open model is of particular interest because it (a) serves as a more realistic system model for many applications; and (b) helps illustrate when a "purely" open or closed system is a good approximation of user behavior.

We focus on the effects of the mean number of requests per session and the think time because the other parameters, e.g. load and job size variability, have similar effects to those observed in Sections 8.4.1 and 8.4.2. Throughout the section, we fix the load of the partly-open system by adjusting the arrival rate, λ. Note that, in contrast to the closed model, adjusting the think time of the partly-open model has no impact on the load.

The partly-open model we discuss aims to mimic user behavior at a web site where, after making a request, the user will stay and make another request with probability p. This model has been mentioned both by prior theoretical [191, 77, 253] and implementation [96] research. Many other variations of partly-open systems have also been proposed in the literature. For instance, Dowdy and Chopra create a hybrid system by specifying the MPL of a closed system using a probability distribution [67]. Another proposed hybrid model places upper and lower bounds on the number of jobs allowed into an open system [124]. A differentiating feature of the partly-open model we discuss is its behavioral nature.

Principle (vii): *A partly-open system behaves similarly to an open system when the expected number of requests per session is small (≤ 5 as a rule-of-thumb) and similarly to a closed system when the expected number of requests per session is large (≥ 10 as a rule-of-thumb).*

Principle (vii) is illustrated clearly in the case study results shown in Figure 8.2 and in the simulation results shown in Figure 8.8(a). When the mean number of requests per session is 1 we have a significant separation between the response time under the scheduling policies, as in open systems. However, when the mean number of requests per session is large, we have comparatively little separation between the response times of the scheduling policies; as in closed systems. Figures 8.2 and 8.8(a) are just a few examples of the



(a) $\rho = 0.6$ (b) $\rho = 0.9$

(c) Model-based simulation (d) Static web impl.

Figure 8.8: *Model and implementation-based results for the partly-open system. (a) and (b) are model-based simulations showing mean response time as a function of the expected number of requests per session. (c) and (d) show the mean response time as a function of the think time, for a fixed load. In (a)-(c),* $E[X] = 10$ *and* $C^2 = 8$*. In (c) and (d), we fix* $\rho = 0.6$ *and* $p = 0.75$*, which yields and average of 4 requests per session.*

range of configurations we studied, and across a wide range of parameters, the point where the separation between the performance of scheduling policies becomes small is, as a rule-of-thumb, around 10 requests per session. Note however that this point can range anywhere between 5 and 20 requests per session as C^2 ranges from 4 to 49 respectively. We will demonstrate in Section 8.6 how to use this rule-of-thumb as a guideline for determining whether a purely open or purely closed workload generator is most suitable, or whether a partly-open generator is necessary.

Principle (viii): *In a partly-open system, think time has little effect on mean response time.*

Figure 8.8 illustrates Principle (viii). We find that the think time in the partly-open system does not affect the mean response time or load of the system under any of these policies. This observation holds across all partly-open systems we have investigated (regardless of the number of requests per session), including the case-studies described in Section 8.3.

Principle (viii) may seem surprising at first, but for PS and FCFS scheduling it can be shown formally under product-form workload assumptions. Intuitively, we can observe that changing the think time in the partly-open system has no effect on the load because the same amount of work must be processed. To change the load, we must adjust either the number of requests per session or the arrival rate. The only effect of think time is to add small correlations into the arrival stream.

8.6 Choosing a system model

The previous sections brought to light vast differences in system performance depending on whether the workload generator follows an open or closed system model. A direct consequence is that the accuracy of performance evaluation depends on accurately modeling the underlying system as either open, closed, or partly-open.

A safe way out would be to always choose a partly-open system model, since it both matches the typical user behavior in many applications and generalizes the open and closed system models – depending on the parameters it can behave more like an open or more like a closed system. However, as Table 8.1 illustrates, available workload generators often support only either closed or open system models. This motivates the following fundamental questions for workload modeling: *"Given a particular workload, is a purely open or purely closed system model more accurate for the workload? When is a partly-open system model necessary?"*

In the remainder of this section we illustrate how our eight principles might be used to answer this question for various web workloads. Our basic method is as follows. For a given system we follow these steps:

1. Collect traces from the system.
2. Construct a partly-open model for the system, since the partly-open model is the most general and accurate. In particular, obtain the relevant parameters for the partly-open model.
3. Given the parameterized partly-open model, determine if an open/closed model is an appropriate substitute for the partly-open model or if the partly-open model is necessary.

Table 8.6 summarizes the traces we collected as part of Step 1. Our trace collection spans many different types of sites, including busy commercial sites, sites of major sporting events, sites of research institutes, and an online gaming site.

We next model each site as a partly-open system. According to Principles (vii) and (viii) the most relevant parameter of a partly-open system model is the number of requests issued in a user session. Other parameters such as the think time between successive requests in a session are of lesser importance. Determining the average number of requests per user session for a web site requires identifying user sessions in the corresponding web trace. While there is no 100% accurate way to do this, we employ some common estimation techniques [15, 140].

First, each source IP address in a trace is taken to represent a different user. Second, session boundaries are determined by a period of inactivity by the user, i.e. a period of time during which no requests from the corresponding IP address are received. Typically, this is accomplished by ending a session whenever there is a period of inactivity larger than timeout threshold τ. In some cases, web sites themselves enforce such a threshold; however, more typically τ must be estimated.

	Type of site	Date	Total #Req.
1	Large corporate web site	Feb'01	1609799
2	CMU web server [4]	Nov'01	90570
3	Online department store	June'00	891366
4	Science institute (USGS[3])	Nov'02	107078
5	Online gaming site [232]	May'04	45778
6	Financial service provider	Aug'00	275786
7	Supercomputing web site [2]	May'04	82566
8	Kasparov-DeepBlue match	May'97	580068
9	Site seeing "slashdot effect"	Feb'99	194968
10	Soccer world cup [103]	Jul'98	4606052

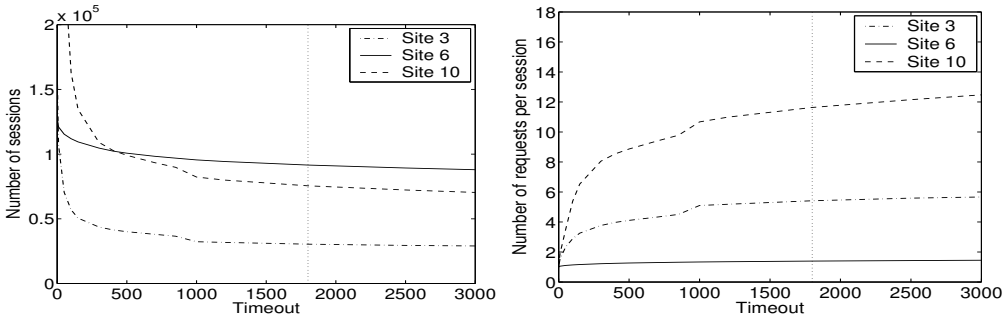
Table 8.6: *A summary table of the web traces used to illustrate how to choose between the open and closed system models.*

We consider two different ways of estimating τ. The first one is to use a defacto standard value for τ, which is 1800s (30 min) [140]. The second method is to estimate τ from the traces themselves by studying the derivative of how τ affects the total number of sessions in the trace. We illustrate this latter method for a few representative traces in Figure 8.9(a). Notice that as the threshold increases from 1-100s the number of sessions decreases quickly; whereas from 1000s on, the decrease is much smaller. Furthermore, Figure 8.9(b) shows that with respect to the number of requests, stabilization is also reached at $\tau > 1000$s. Hence we adopt $\tau = 1800$s in what follows.

The mean number of requests per session when $\tau = 1800$s for all traces is summarized in Table 8.7. The table illustrates that the average number of requests for web sessions varies largely depending on the site, ranging from less than 2 requests per session to almost 13. Interestingly, even for similar types of web sites the number of requests can vary considerably. For example sites 8 and 10 are both web sites of sporting events (a chess tournament and a soccer tournament), but the number of requests per session is quite low (2.4) in one case, while quite high (11.6) in the other. Similarly, sites 2, 4, and 7 are all web sites of scientific institutes but the number of requests per sessions varies from 1.8 to 6.

Using the rule of thumb in principle (vii), we can conclude that neither the open nor the closed system model accurately represents all the sites. For sites 1, 2, 4, 6, 8, and 9 an open system model is accurate; whereas a closed system model is accurate for the sites 5 and 10. Further, it is not clear whether an open or closed model is appropriate for sites 3 and 7.

The impact of choosing between open and closed system models correctly is demonstrated by site 10, the world cup dataset. This is the same dataset used in the static web case study, where we saw large differences depending on whether we modeled the workload using an open or a closed system. We have just concluded that a closed model is most appropriate for this workload, thus the magnitude of differences between the open and closed results in Figure 8.2 illustrates the impact of the choice of a system model.



(a) Number of sessions vs Timeout length (b) Number of requests vs Timeout length

Figure 8.9: *Statistics for 3 representative web traces (sites 3, 6, and 10) illustrating (a) the number of user sessions as a function of the timeout threshold and (b) the expected number of requests per session as a function of the timeout threshold. The vertical line on each plot corresponds to a timeout of 1800s. From these plots we can conclude that an open model is appropriate for site 6, a closed model is appropriate for site 10, and neither an open or a closed is appropriate for site 3.*

Site	1	2	3	4	5	6	7	8	9	10
Avg. Req. per session	2.4	1.8	5.4	3.6	12.9	1.4	6.0	2.4	1.2	11.6
Approp. Sys. Model	open	open	?	open	closed	open	?	open	open	closed

Table 8.7: *A summary table of the expected number of visits in the web traces used to illustrate how to make a choice between the open and closed system models.*

8.7 Concluding remarks

In this chapter we considered one particularly important practical extension of the M/GI/1 model: dependencies between service completions and new arrivals. Throughout computer applications it is quite common that users must wait for one request to complete before making a new request, and this factor is ignored in traditional queueing models, such as the M/GI/1, that rely on open system models. We have seen that dependencies between the arrival and completion processes in the closed and partly-open system models limit the benefits provided by scheduling with respect to mean response time. However, we have also seen that in many practical settings scheduling can still be quite beneficial.

This chapter presented eight simple principles that function to explain the differences in behavior of closed, open, and partly-open system models. The more intuitive of these principles point out that response times under closed systems are typically lower than in the corresponding open system with equal load, and that as MPL increases, closed systems approach open ones. Less obviously, our principles point out that: (a) the magnitude of the difference in response times between closed and open systems can be very large, even under moderate load; (b) the convergence of closed to open as MPL grows is slow, especially when service demand variability (C^2) is high; and (c) scheduling is far more beneficial in open systems than in closed ones. We also compare the partly-open model with the open and closed models. We illustrate the

strong effect of the number of requests per session and C^2 on the behavior of the partly-open model, and the surprisingly weak effect of think time.

These principles underscore the importance of choosing a workload generator with the appropriate system model when experimenting with system design changes. For example, in the context of web applications, the arrival process at a web site is best modeled by a partly-open system yet most web workload generators are either strictly open or strictly closed. Our findings provide guidelines for choosing whether an open or closed model is the better approximation based on characteristics of the workload. A high number of simultaneous users (more than 1000) suggests an open model, but a high number of requests per session (more than 10) suggests a closed model. Both these cutoffs are affected by service demand variability: highly variable demands requires larger cutoffs. Contrary to popular belief, it turns out that think times are irrelevant to the choice of an open or closed model since they only affect the load.

The work in this chapter highlights the fact that understanding the appropriate system model is essential to understanding the impact of scheduling. In particular, once it has been determined whether a closed, open, or partly-open model is a better approximation, that in turn provides a guideline for the effectiveness of scheduling. Scheduling is most effective in open systems, but can still provide benefits in closed systems when both the load is moderate and service demand variability is high. So, though there are some cases when scheduling is not effective, it is possible to identify these cases easily, and in most practical settings scheduling is valuable resource for improving system performance, even when there is feedback between arrivals and completions.

CHAPTER 9

The impact of multiserver architectures

So far in this thesis, we have considered only single server systems. In practice there are many cases where multiple, slower, cheaper servers are preferable to a single, faster, more expensive server. For instance, high traffic web sites are increasingly moving towards server farm architectures and processors are increasingly using multi-core designs. This trend towards multiserver designs is also prevalent across other applications such as wireless networks, where multi-channel designs are becoming common.

Given the increased adoption of multiserver designs in computer applications, it is important to understand how the benefits of scheduling that we have studied in single server systems translate to multiserver settings. Intuitively, it is clear that scheduling will be less effective in multiserver settings than it was in single server settings. This is because scheduling is the only way to avoid forcing small jobs to queue behind larger jobs in a single server system, while in a multiserver system this happens even under FCFS scheduling. However, we will show that scheduling still provides performance benefits in multiserver systems, just not the extreme gains we saw in single server systems.

In this chapter our goal is to characterize the impact of scheduling in multiserver systems. However, the analysis of multiserver systems is a difficult problem, and outside of FCFS scheduling little is understood analytically. Thus, we cannot hope to obtain results in the generality that we obtained for the single server queue. Instead, we will focus on understanding one important aspect of scheduling: prioritization.

Much of queueing theory is devoted to analyzing priority queues, where jobs are labeled and served in accordance with a preemptive priority scheme: high-priority jobs preempt medium-priority jobs, which in turn preempt low-priority jobs in the queue (see Section 3.2.3). These simple priority schemes occur frequently in practice. For example, sometimes the priority of a job is determined by the job's owner via a Service Level Agreement (SLA), whereby certain customers have chosen to pay more so as to get high-priority access to some high-demand resource. Other times, the priority of a job is artificially created, so as to maximize a company's profit or increase system utilization. For example, an online store may choose to give high priority to the requests of big spenders, so that those customers are less likely to go elsewhere, see [138]. In addition to the practical importance of priority queues, they serve as a theoretical building block for the analysis of more complicated scheduling policies, e.g. SRPT and PSJF, in the single server setting (see Section 3.2). Thus, one can view the results in this chapter as a first step towards the analysis of SRPT in the multiserver setting.

Though analyzing the mean response time (and higher moments of response time) for different classes of jobs is clearly an important problem, which has been well understood in the case of a single-server M/GI/1 queue since the 1950's [58], the problem becomes much more difficult when considered in the context of a multiserver M/GI/k system. Even for an M/M/k system when jobs have different completion rates little is known. The reason that priority queueing is difficult to analyze in a multiserver setting is that jobs of different priorities may be in service (at different servers) at the same time, thus the Markov chain representation of the multi-class, multiserver queue appears to require tracking the number of jobs of each class. Hence one needs a Markov chain which is infinite in m dimensions, where m is the number of priority classes. While the analysis of a 1-dimensionally infinite Markov chain is easy, the analysis of an m-dimensionally infinite Markov chain ($m > 1$) is largely intractable.

In this chapter, we introduce a new analytical approach that provides the first near-exact analysis of the M/PH/k queue with $m \geq 2$ preemptive-resume priority classes. Our approach, which we refer to as Recursive Dimensionality Reduction (RDR) [242, 95], is very different from prior approaches. RDR allows us to recursively reduce the m-dimensionally infinite state space created by tracking the m priority classes to a 1-dimensionally infinite state space, which is analytically tractable. The dimensionality reduction is done without any truncation; rather, we reduce dimensionality by introducing "busy period transitions" within our Markov chain, for various types of busy periods created by different job classes. The only approximation in the RDR method stems from the fact that we need to approximate these busy periods using Markovian (phase-type) PH distributions. We find that matching three moments of these busy periods is usually possible using a 2-phase Coxian distribution, and provides sufficient accuracy, within a couple percent of simulation, for all our experiments.

Our new analytic technique allows us to obtain many interesting insights about prioritization in multiserver settings [95]. For instance, RDR allows us to compare the performance of priority queueing in a multiserver system with k servers each of speed $1/k$ with the performance of a priority queue with single server of speed 1. We find that the effect of priorities in a single server system can be very different than in a multiserver system of equal capacity. In addition, RDR allows us to study the effect of priority policies that favor short jobs ("smart prioritization") versus priority policies that favor long jobs ("foolish prioritization") under systems with different numbers of servers. Understanding the effect of "smart" prioritization is important because many common scheduling policies are designed to give priority to short jobs. Further, RDR allows us to study how effective class aggregation (aggregating $m > 2$ priority classes into just 2 priority classes) is as an approximation for dealing with systems having more than two priority classes. We evaluate two types of class aggregation in order to illustrate when class aggregation serves as a reasonable approximation.

Our new analytic technique also allows us to address the question of system design [242]. In particular, in we ask: given the choice of k slow servers of speed s/k or one fast server of speed s, which is preferable? The question of "how many servers are best" has a long history in the literature, but this history is limited to the FCFS setting. We will study this question in the setting of priority queues and then contrast the results with the case of FCFS scheduling in order to develop an understanding of the effect of prioritization on the design of multiserver systems.

9.1 Prior work analyzing multiserver priority queues

The literature on multiserver priority queues is vast, however almost all results are restricted to only *two priority classes*. Further, of the results for two priority classes, all assume *exponential service times*. The only papers *not* restricted to two priority classes are approximations based on assuming that the multiserver behavior parallels that of a single server system [38] or approximations based on aggregating priority classes in multi-class systems so that it becomes a dual priority queue [148, 157].

Dual priority queues

We start by describing the papers restricted to *two priority classes* and *exponentially distributed service demands*. Techniques for analyzing the M/M/k dual priority system can be organized into four types on which we elaborate below: (i) approximations via aggregation or truncation; (ii) matrix analytic methods; (iii) generating function methods; (iv) special cases where the priority classes have the same mean. Unless otherwise mentioned, preemptive-resume priorities should be assumed.

Nearly all analysis of dual priority M/M/k systems involves the use of Markov chains, which with two priority classes grows infinitely in two dimensions (one dimension for each priority class). In order to overcome this, researchers have simplified the chain in various ways. Kao and Narayanan truncate the chain by either limiting the number of high priority jobs [107], or the number of low priority jobs [108]. Nishida aggregates states, yielding an often rough approximation [157]. Kapadia, Kazmi and Mitchell explicitly model a finite queue system [110]. Unfortunately, aggregation or truncation is unable, in general, to capture the system performance as the traffic intensity grows large.

Although, in theory, matrix analytic methods can be used to directly analyze a 2D-infinite Markov chain (see for example [45]), matrix analytic methods are much simpler and more computationally efficient when applied to a 1D-infinite Markov chain. Therefore, most papers that use the matrix analytic method to analyze systems involving 2D-infinite Markov chains first reduce the 2D-infinite chain to a 1D-infinite chain by, for example, truncating the state space by placing an upper bound on the number of jobs [107, 108, 126, 156]. Miller [145] and Ngo and Lee [156] partition the state space into blocks and then "super-blocks," according to the number of high priority customers in queue. However, [145] experiences numerical instability issues when $\rho > 0.8$.

A third stream of research capitalizes on the exponential job sizes by explicitly writing out the balance equations and then finding roots via generating functions. In general these yield complicated mathematical expressions susceptible to numerical instabilities at higher loads. See King and Mitrani [148]; Gail, Hantler, and Taylor [81, 82]; Feng, Kowada, and Adachi [75]; and Kao and Wilson [109].

Finally there are papers that consider the special case where the multiple priority classes all have the same mean. These include Davis [63], Kella and Yechiali [111] (for non-preemptive priorities), and Buzen and Bondi [48].

The only work dealing with non-exponential service times is contained in a pair of very recent papers, by Sleptchenko et. al. [211, 212]. Both papers consider a two-priority, multiserver system where within each priority there may be a number of different classes, each with its own different exponential job size distribution. This is equivalent to assuming a *hyper-exponential job size distribution* for each of the *two priority classes*. The problem is solved via a combination of generating functions and matrix analytic methods. In theory, their technique may be generalizable to PH distributions, though they evaluate only hyper-exponential distributions due to the increased complexity necessary when using more general PH distributions.

More than two priority classes

For the case of *more than two priority classes*, there are only coarse approximations. The Bondi-Buzen (BB) approximation [38] is beautiful in its simplicity and usability. It is based on an intuitive observation that the "improvement" of priority scheduling over FCFS scheduling under k servers is similar to that for the case of one server with equal total capacity:

$$\frac{E[D^{\text{M/GI/}k\text{/prio}}]}{E[D^{\text{M/GI/}k\text{/FCFS}}]} \approx \frac{E[D^{\text{M/GI/1/prio}}]}{E[D^{\text{M/GI/1/FCFS}}]} = \text{ scaling factor.} \tag{9.1}$$

Here $E[D^{\text{M/GI/}k\text{/prio}}]$ is the overall mean delay under priority scheduling with k servers of speed $1/k$, and $E[D^{\text{M/GI/}k\text{/FCFS}}]$ is defined similarly for FCFS, while M/GI/1 refers to a single server queue with speed 1. This relation is exact when job sizes are exponential with the same rate for all classes; however what happens when this is not the case is studied for the first time in this chapter.

The other approximation (which we denote by MK-N) that allows for more than two priority classes and exponential job sizes is due to Mitrani and King [148], and also used by Nishida [157] to extend the latter author's analysis of two priority classes to $m > 2$ priority classes. The MK-N approximation analyzes the mean delay of the lowest priority class in an M/M/k queue with $m \geq 2$ priority classes by *aggregating all the higher priority classes*. Thus, instead of aggregating all jobs into one class, as BB does, MK-N aggregates into two classes. The job size distribution of the aggregated class is then approximated with an exponential distribution by matching the first moment of the distribution.

9.2 Analyzing the M/PH/k with m priority classes

In this section we describe the RDR technique, dividing our explanation into three parts. As an introduction, in Section 9.2.1, we deal only with the simplest case of $m = 2$ priority classes and exponential job sizes, which we solve using the techniques in [165, 242]. We then move to the difficult case of $m > 2$ priority classes, but still with exponential service times, in Section 9.2.2. Here the techniques from [165, 242] do not apply, so we introduce Recursive Dimensionality Reduction (RDR). The RDR approach uses the analysis of the $m - 1$ priority classes to analyze the m-th priority class. This is a non-trivial procedure for $m > 2$ since it involves evaluating many complex passage times (busy periods) in the chain representing the $m - 1$ priority classes, as these passage times now form transitions within the chain representing m priority classes. Finally in Section 9.2.3, we show how RDR can be applied to the most general case of PH service times with $m > 2$ priority classes.

All the analysis up to through Section 9.2.3 deals with how to derive mean per-class response times. In Section 9.2.4 we illustrate how the RDR method can be extended to obtain *variance of response time* for each class. Finally, in Section 9.2.5, we introduce RDR-A, which is an approximation of RDR, allowing very fast (< 1 second) evaluation of high numbers of priority classes and servers, with small ($< 5\%$) error.

9.2.1 Exponential job sizes and two priority classes

Consider the simplest case of two servers and two priority classes, high (H) and low (L), with exponentially distributed sizes with rates μ_H and μ_L respectively. Figure 9.1(a) illustrates a Markov chain of this system,

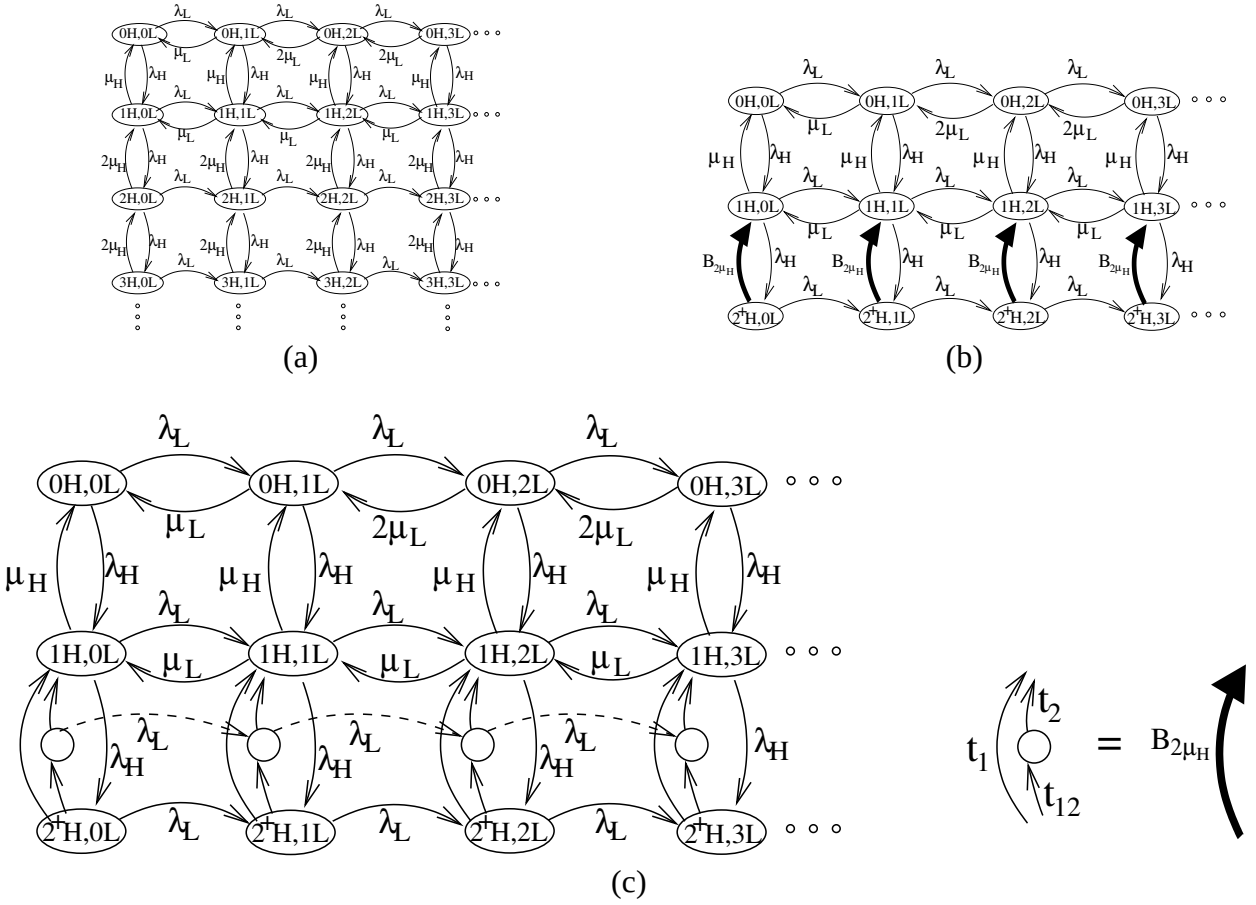


Figure 9.1: *(a) Markov chain for an M/M/2 queue with two priority classes. This Markov chain is infinite in two dimensions. Via the Dimensionality Reduction technique, we arrive at the chain in (b), which uses busy period transitions, and is only infinite in one dimension. In (b), the busy period is represented by a single transition. In (c), the busy period is represented by a two phase PH distribution (with Coxian representation), yielding a 1D-infinite Markov chain.*

whose states track the number of high priority and low priority jobs; hence this chain grows infinitely in two dimensions. Observe that high priority jobs simply see an M/M/2 queue, and thus their mean response time is well-known. Low priority jobs, however, have access to either an M/M/2, M/M/1, or no server at all, depending on the number of high priority jobs. Thus their mean response time is more complicated, and this is where we focus our efforts.

Figure 9.1(b) illustrates the reduction of the 2D-infinite Markov chain to a 1D-infinite chain. The 1D-infinite chain tracks the number of low priority jobs exactly. For the high priority jobs, the 1D-infinite chain only differentiates between zero, one, and two-or-more high priority jobs. As soon as there are two-or-more high priority jobs, a *high priority busy period* is started. During the high priority busy period, the system

only services high priority jobs, until the number of high priority jobs drops to one.[1] The length of time spent in this high priority busy period is exactly an M/M/1 busy period where the service rate is $2\mu_H$. We denote the duration of this busy period by the transition labelled $B_{2\mu_H}$.

The busy period $B_{2\mu_H}$ is *not* exponentially-distributed. Hence it is not clear how it should fit into a Markov model. We use a PH distribution (specifically a Coxian distribution) to match the first three moments of the distribution of $B_{2\mu_H}$. Parameters of the PH distribution, whose first three moments match those of $B_{2\mu_H}$, are calculated via the closed form solutions provided in [164].

Figure 9.1(c) illustrates the same 1D-infinite chain as in Figure 9.1(b), except that the busy period transition is now replaced by a two phase PH distribution with parameters t_1, t_{12} and t_2. The limiting probabilities in this 1D-infinite chain can be analyzed using matrix analytic methods [125]. These in turn yield the mean number of low priority jobs, which via Little's law yields the mean response time for low priority jobs. The only inaccuracy in the above approach is that only three moments of the high priority busy period are matched. We will see later that this suffices to obtain very high accuracy across a wide range of load and job size distributions.

More formally, the 1D-infinite Markov chain is modeled as a (nonhomogeneous) QBD process, where level ℓ of the process denotes the ℓ-th column, namely all states of the form (iH,ℓL) for each ℓ. The generator matrix, $\mathbf{Q}$, of this process can be expressed as a block diagonal matrix:

$$\mathbf{Q} = \begin{pmatrix} \mathbf{L}^{(0)} & \mathbf{F}^{(0)} & & & \\ \mathbf{B}^{(1)} & \mathbf{L}^{(1)} & \mathbf{F}^{(1)} & & \\ & \mathbf{B}^{(2)} & \mathbf{L}^{(2)} & \mathbf{F}^{(2)} & \\ & & \ddots & \ddots & \ddots \end{pmatrix}$$

where submatrix $\mathbf{F}^{(\ell)}$ encodes (forward) transitions from level (column) ℓ to level $\ell+1$ for $\ell \geq 0$, submatrix $\mathbf{B}^{(\ell)}$ encodes (backward) transitions from level ℓ to level $\ell-1$ for $\ell \geq 1$, and submatrix $\mathbf{L}^{(\ell)}$ encodes (local) transitions within level ℓ for $\ell \geq 0$. Specifically, for the Markov Chain depicted in Figure 9.1(c), we order the 4 states in level ℓ as: $(0H, \ell L)$, $(1H, \ell L)$, $(2^+H, \ell L)$, $(xH, \ell L)$. The pair of states $\{(2^+H, \ell L), (xH, \ell L)\}$ are the states used to represent the busy period, where the state $(2^+H, \ell L)$ denotes the start of the busy period and $(xH, \ell L)$ denotes the intermediate "bubble" state in the busy period. Given this ordering of states, we have:

$$\mathbf{L}^{(\ell)} = \left(\begin{array}{cc|cc} -\sigma_1 & \lambda_H & & \\ \mu_H & -\sigma_2 & \lambda_H & \\ \hline & t_1 & -\sigma_3 & t_{12} \\ & t_2 & & -\sigma_4 \end{array}\right)$$

where $\sigma_i = \sum_{j \neq i} \mathbf{Q}_{ij}$. The delineations of the matrix $\mathbf{L}^{(\ell)}$ are intended to highlight the busy period.

[1] Throughout the paper a "higher priority busy period" is defined as the time from when the system has k higher priority jobs until there are only $k-1$ higher priority jobs.

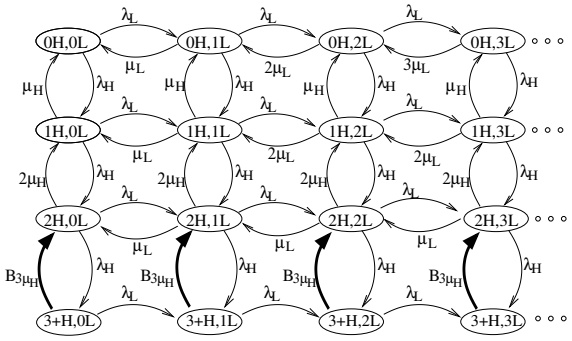


Figure 9.2: *This chain illustrates the case of two priority classes and three servers. The busy period transitions are replaced by a Coxian phase-type distribution matching three moments of the busy period duration, as shown in Figure 9.1.*

$\mathbf{F}^{(\ell)} = \lambda_L \mathbf{I}$, for $\ell \geq 0$, where $\mathbf{I}$ is a 4×4 identity matrix, and

$$\mathbf{B}^{(\ell)} = \mu_L \begin{pmatrix} \min(2,\ell) & & & \\ & 1 & & \\ & & 0 & \\ & & & 0 \end{pmatrix}$$

for $\ell \geq 1$.

The stationary probabilities of being in level ℓ, $\vec{\pi}_\ell$, are then given recursively by $\vec{\pi}_\ell = \vec{\pi}_{\ell-1} \cdot \mathbf{R}^{(\ell)}$, where $\vec{\pi}_0$ and $\mathbf{R}^{(\ell)}$ are calculated as follows: Define $\hat{\ell}$ to be the level that the QBD process starts repeating (in Figure 9.1(c), $\hat{\ell} = 1$), then for $\ell = 1, \ldots, \hat{\ell}$, we have that $\mathbf{R}^{(\ell)}$ is given recursively by:

$$\mathbf{F}^{(\ell-1)} + \mathbf{R}^{(\ell)} \cdot \mathbf{L}^{(\ell)} + \mathbf{R}^{(\ell)} \cdot \mathbf{R}^{(\ell+1)} \cdot \mathbf{B}^{(\ell+1)} = \mathbf{0},$$

where $\mathbf{0}$ is a zero matrix of appropriate dimension (4×4). For $\ell \geq \hat{\ell} + 1$, $\mathbf{R}^{(\ell)} = \mathbf{R}$, where $\mathbf{R}$ is given by the minimal solution to the following matrix quadratic equation:

$$\mathbf{F}^{(\hat{\ell})} + \mathbf{R} \cdot \mathbf{L}^{(\hat{\ell})} + \mathbf{R}^2 \cdot \mathbf{B}^{(\hat{\ell})} = \mathbf{0}.$$

Vector $\vec{\pi}_0$ is given by a positive solution of $\vec{\pi}_0 \left(\mathbf{L}^{(0)} + \mathbf{R}^{(1)} \cdot \mathbf{B}^{(1)} \right) = \vec{0}$, normalized by the equation $\vec{\pi}_0 \sum_{\ell=0}^{\infty} \prod_{i=1}^{\ell} \mathbf{R}^{(i)} \cdot \vec{1} = 1$, where $\vec{0}$ and $\vec{1}$ are column vectors with an appropriate number of elements of 0 and 1, respectively. The mean response time can now be computed using the stationary distributions, $\vec{\pi}_\ell$'s, given above, via Little's law.

Figure 9.2 shows the generalization to a three server system. We simply add one row to the chain shown in Figure 9.1, and now differentiate between 0, 1, 2, or 3-or-more high priority jobs. This can be easily extended to the case of $k > 3$ servers.

9.2.2 Exponential job sizes and m priority classes,

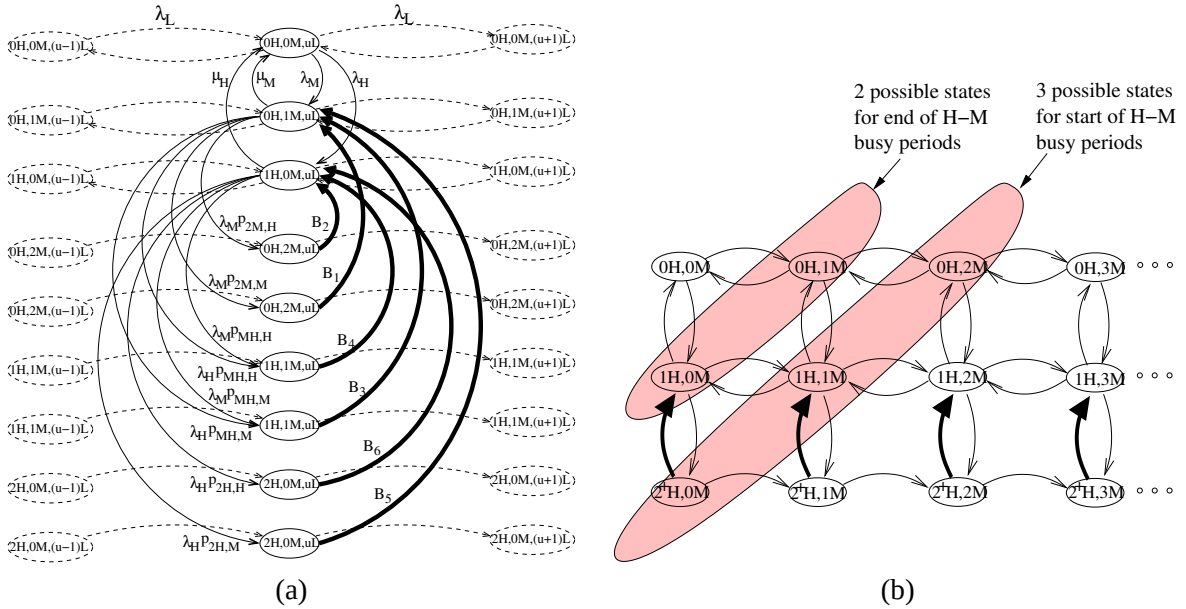
We now turn to the more difficult case of $m > 2$ priority classes. We illustrate this for the case of two servers and three priority classes: high priority (H), medium-priority (M), and low priority (L). The mean response time for class H jobs and that for class M jobs are easy to compute. Class H jobs simply see an M/M/2 queue. Class M jobs see the same system that the low priority jobs see in an M/M/2 queue having two priority classes. Replacing the L's by M's in the chain in Figure 9.1 yields the mean response time for the M class jobs.

The analysis of the class L jobs is the difficult part. The obvious approach would be to aggregate the H and M jobs into a single class, so that we have a 2-class system (H-M versus L jobs). Then we could apply the technique of the previous section, tracking exactly the number of low priority jobs and maintaining limited state information on the H-M class. This is the approach that we follow in Section 9.2.5 in deriving our RDR-A approximation. However, this approach is imprecise because the duration of the busy periods in the H-M class depends on whether the busy period was started by 2H jobs, 1H and 1M job, or 2M jobs in service. By ignoring the priorities among H's and M's, we are ignoring the fact that some types of busy periods are more likely than others. Even given the information on who starts the busy period, this still does not suffice to determine its duration, because the duration is also affected by the prioritization within the aggregated H-M class.

Thus, a precise response time analysis of class L requires maintaining more information. As before, we want to exactly track the number of class L jobs. Given that there are two servers, we need to further differentiate between whether there are zero H and M jobs, one H or M job, or two or more H and M jobs. Whenever there are two or more H and M jobs, we are in an H-M busy period. For an M/M/2 with three priority classes, there are *six types of busy periods* possible, depending on the state at the start of the busy period – $(2H, 0M)$, $(1H, 1M)$, or $(0H, 2M)$ – and the state in which the busy period ends – $(1H, 0M)$ or $(0H, 1M)$. We derive the busy period duration by conditioning on who starts and ends the busy period.

Figure 9.3 (a) shows the level of the 1D-infinite chain in which the number of class L jobs is u. In state (wH,vM,uL), v class M jobs and w class H jobs are in the system if $v + w < 2$; otherwise, the state (wH,vM,uL) denotes that we are in a H-M busy period that was started by v class M jobs and w class H jobs. Observe that there are six types of busy periods depicted, labelled B_1, B_2, ..., B_6; the busy period is determined by the state in which it was started and the state in which it ends. Notice, for example, that both states in the fourth and fifth row are labelled (0H,2M,uL), meaning that the busy period is started by two class M jobs; but these two states differ in the class of the job that is left at the end of the H-M busy period. In state (0H,2M,uL) of the fourth row, the busy period ends leaving a class H job, whereas in state of the fifth row, the busy period ends leaving a class M job. (Recall that the class of job left at the end of a busy period is probabilistically determined at the *beginning* of the busy period and the duration of the busy period is conditioned on the class of the job left at the end.) Here $p_{2M,H}$, for example, denotes the probability that the busy period started by two class M jobs ends leaving one class H job, whereas $p_{HM,M}$ denotes the probability that the busy period started by one class H and one class M job, ends leaving one class M job. The remaining probabilities are defined similarly.

It remains to derive the moments of the duration of busy periods, B_1, B_2, ..., B_6, and probabilities $p_{2M,M}$, $p_{2M,H}$, $p_{HM,M}$, $p_{HM,H}$, $p_{2H,M}$, and $p_{2H,H}$ in Figure 9.3(a). The trick to deducing these quantities is to observe that the six busy periods correspond to passage times between two "diagonal" (shaded) levels in the chain shown in figure 9.3(b), which is the 1D-infinite chain that we used to analyze the class M performance. We refer to the right shaded diagonal level as level ℓ and the left shaded diagonal level as



(a) (b)

Figure 9.3: *An overview of applying RDR in the case of 3 priority classes. (a) shows a level of the 1D-infinite chain used to compute mean response time for low priority jobs in the case of three priority classes and two servers, and all exponential service times. (b) shows the chain used to compute moments of the durations of the six busy period transitions.*

level $\ell - 1$ (where $\ell = 3$). Note that the 3 states in level ℓ correspond to the three possible "start" states for busy periods, and the two states in level $\ell - 1$ correspond to the two possible "end" states for the busy periods. Thus, for example, busy period B_1 in Figure 9.3(a) corresponds to the first passage time from state $(0H, 2M)$ to state $(0H, 1M)$ in the chain in Figure 9.3(b), given that $(0H, 1M)$ is the first state reached in level $\ell - 1$, when starting in state $(0H, 2M)$ of level ℓ. Likewise, probability $p_{2M,M}$ corresponds to the probability that, in Figure 9.3(b), state (0H,1M) is the first state of the two possible "end" states that is reached in level $\ell - 1$, given that the "start" state is $(0H, 2M)$. These conditional inter-level passage times and ending probabilities within the chain in Figure 9.3(b) can be calculated using techniques developed by Neuts in [155]. We provide a precise description of this in [95]. Observe that these computations are greatly facilitated by the fact that our chains are infinite in only one dimension.

More formally, the 1D-infinite Markov chain shown in Figure 9.3(a) is modeled as a (nonhomogeneous) QBD process, as in Section 9.2.1. Here, level ℓ of the QBD process denotes the ℓ-th column, namely all states of the form (iH,jM,ℓL) for each ℓ. The submatrices of the QBD process, $\mathbf{L}^{(\ell)}$, $\mathbf{F}^{(\ell)}$, and $\mathbf{B}^{(\ell)}$, have size 15×15. Specifically, the forward transitions are $\mathbf{F}^{(\ell)} = \lambda_L \mathbf{I}$, for $\ell \geq 0$, where $\mathbf{I}$ is a 15×15 identity matrix. The backward transitions are

$$\mathbf{B}^{(\ell)} = \mu_L \begin{pmatrix} \min(2,\ell) & & \\ & 1 & \\ & & \mathbf{0} \end{pmatrix}$$

for $\ell \geq 1$, where $\mathbf{0}$ is a zero matrix of size 13×13. We again order the 15 states in level ℓ as: $(0H, 0M, \ell L)$,

$(0H, 1M, \ell L)$, $(1H, 0M, \ell L)$, followed by the two states associated by busy period B_i, for $i = 1, 2, \ldots, 6$, where, as before, the two states associated with each busy period are ordered by start state, intermediate state. The local transitions are then given by:

$$\mathbf{L}^{(\ell)} = \left(\begin{array}{ccc|c|c|c|c|c|c}
-\sigma_1 & \lambda_M & \lambda_H & \lambda_M\vec{p}^{(2M,M)} & \lambda_M\vec{p}^{(2M,H)} & \lambda_M\vec{p}^{(MH,M)} & \lambda_M\vec{p}^{(MH,H)} & & \\
\mu_M & -\sigma_2 & & & & \lambda_H\vec{p}^{(MH,M)} & \lambda_H\vec{p}^{(MH,H)} & \lambda_H\vec{p}^{(2H,M)} & \lambda_H\vec{p}^{(2H,H)} \\
\mu_H & & -\sigma_3 & & & & & & \\
\hline
 & \vec{t}^{(1)} & & \mathbf{T}^{(1)} & & & & & \\
\hline
 & & \vec{t}^{(2)} & & \mathbf{T}^{(2)} & & & & \\
\hline
 & \vec{t}^{(3)} & & & & \mathbf{T}^{(3)} & & & \\
\hline
 & & \vec{t}^{(4)} & & & & \mathbf{T}^{(4)} & & \\
\hline
 & \vec{t}^{(5)} & & & & & & \mathbf{T}^{(5)} & \\
\hline
 & & \vec{t}^{(6)} & & & & & & \mathbf{T}^{(6)}
\end{array}\right)$$

for $\ell \geq 0$, where the lines delineate the six busy periods ordered as $B_1, B_2, \ldots, B_6$, and where

$$\vec{t}^{(i)} = \begin{pmatrix} t_1^{(i)} \\ t_2^{(i)} \end{pmatrix}, \quad \mathbf{T}^{(i)} = \begin{pmatrix} -\sigma_{2i+2} & t_{12}^{(i)} \\ 0 & -\sigma_{2i+3} \end{pmatrix}, \quad \text{and } \sigma_i = \sum_{j \neq i} \mathbf{Q}_{ij}$$

$$\vec{p}^{(X,Y)} = (p_{X,Y}, 0) \quad \text{where } X \in \{2H, HM, 2M\}, Y \in \{H, M\},$$

for $1 \leq i \leq 6$, corresponding to busy periods $B_1, B_2, \ldots, B_6$. Here $t_1^{(i)}, t_2^{(i)}, t_{12}^{(i)}$ are the rates of the PH distribution used to represent busy period B_i.

Now, the stationary probability of this QBD process can be calculated via matrix analytic methods, as in Section 9.2.1. The mean response time in the priority system can then be computed using the stationary distributions via Little's law.

The extension of RDR to $m > 3$ classes is straightforward but increases in complexity. For example, for the case of $m = 4$ classes, we proceed as in Figure 9.3, where we first create a chain that tracks exactly the number of jobs in class 4, and creates busy periods for the aggregation of the three higher priority classes. Then, to derive the busy periods for the three higher priority classes, we make use of the existing chain for three classes shown in Figure 9.3(a), and compute the appropriate passage times for that chain. For an M/M/k with m priority classes, there are $\binom{m+k-2}{k}\binom{m+k-3}{k-1}$ possible busy periods, where the first term in the product represents the number of possible start states (all combinations of up to $m-1$ priority classes over k servers) and the second term represents the number of possible end states (all combinations of up to $m-1$ priority classes over $k-1$ servers). That is, the number of different types of busy periods is polynomial in k if m is constant ($\Theta(k^m)$), and it is polynomial in m if k is constant ($\Theta(m^k)$); however, it is exponential in k and m if neither k nor m is constant.[2]

[2]We note that in practice the number of busy periods can be reduced further, so that an M/M/k with m priority classes has $\binom{m+k-3}{k-1}^2$ busy periods of class 1 to class $m-1$ jobs. An advantage of this reduction is that the number of busy periods of class 1 to class $m-1$ jobs becomes independent of the type of PH distributions that is used to approximate the busy period of class 1 to class $m-2$ jobs. The trick to reducing the number of busy periods is illustrated by considering the example of the M/M/2 with three classes, shown in Figure 9.3. Here, by taking the mixture of the six busy periods, B_1, B_2, ..., B_6, we can approximate the H-M busy period by *four* PH distributions. These four distributions of the H-M busy period are differentiated by the state from which we *enter* the H-M busy period (either (1H,0M) or (0H,1M)) and by the state we return to after the H-M busy period (either (1H,0M) or (0H,1M)). More details are provided in [163].

Practically speaking, the RDR approach is fast for a small number of servers and a small number of priority classes. In examples we ran with an M/M/2 and 10 priority classes, the RDR algorithm yielded mean response times within tens of seconds.[3]

9.2.3 The M/PH/k with m priority classes

In this section, we explicitly describe how RDR can be applied to analyze the case of PH job size distributions. We describe RDR for the case of two servers ($k = 2$) and two priority classes ($m = 2$), high (H) and low (L) , where the class H jobs have a particular 2-phase PH job size distribution with Coxian representation, shown in figure 9.4(a).[4] Generalization to higher k's and higher m's is straightforward by applying the recursive algorithm introduced in Section 9.2.2.

Analyzing the performance of class H is trivial, since high priority jobs simply see the mean response time in an M/PH/2 queue, which can be analyzed via standard matrix analytic methods. To analyze the class L jobs, as before, we create a 1D-infinite Markov chain tracking the class L jobs, and use busy period transitions to represent needed information regarding the class H jobs.

Observe that under the 2-phase Coxian job sizes distribution, we will need *four* different types of busy periods for high priority jobs, depending on the phases of the two jobs starting the busy period (1 & 1, or 1 & 2) and the phase of the job left at the end of the busy period (1 or 2). To derive the durations of these busy periods, we observe that the busy periods correspond to passage times from shaded level 3 to shaded level 2 in the Markov chain shown in Figure 9.4(b). Figure 9.4(b) describes the behavior of class H jobs, where states track the number of high priority jobs in the system and the phases of the jobs being processed. Namely, at state (0H) there are no high priority jobs in the system; at state (1H,i), there is one high priority job in phase i; at state (nH,i, j) there are n high priority jobs in the system and the two jobs are being processed are in phase i and j, respectively (jobs in the queue are all in phase 1). The first passage times in Figure 9.4 are computed again using Neuts' method, as described in [95].

Figure 9.4(c) shows a level of the chain that tracks the number of low priority jobs, where the number of low priority jobs is u. The low priority job sizes are assumed to be exponentially distributed, but this can be generalized to PH distributions. In state (0H,uL), no high priority jobs are in the system. An arrival of a high priority job in state (0H,uL) triggers a transition to state (1H,1,uL). In state (1H,j,uL), one high priority job in phase j is in service for $j = 1, 2$. An arrival of a high priority job in state (1H,j,uL) triggers a transition to state (2^+H,1, j,uL) for $j = 1, 2$. In state (2^+H,1, j,uL), at least two high priority jobs are in the system, and the two jobs that started the busy period were in phase 1 and j, respectively, for $j = 1, 2$. The four types of busy periods are labelled as B_1, B_2, B_3, and B_4, and the duration of these busy periods is approximated by PH distributions by matching the first three moments of the busy period distribution (note that the busy period cannot start with two jobs in phase two). Finally, $p_{(1,j),i}$ denotes the probability that a busy period started by two jobs in phases 1 and j, ends with a single job in phase i, for $j = 1, 2$, and $i = 1, 2$.

[3] Help with implementing the procedure described in this chapter is provided at [163].

[4] Under the Coxian job size distribution, a job starts in phase one where it is processed for a time exponentially distributed with rate $\mu_H^{(1)}$, and then either completes (with probability $q_H = 1 - p_H$) or moves to phase two (with probability p_H).

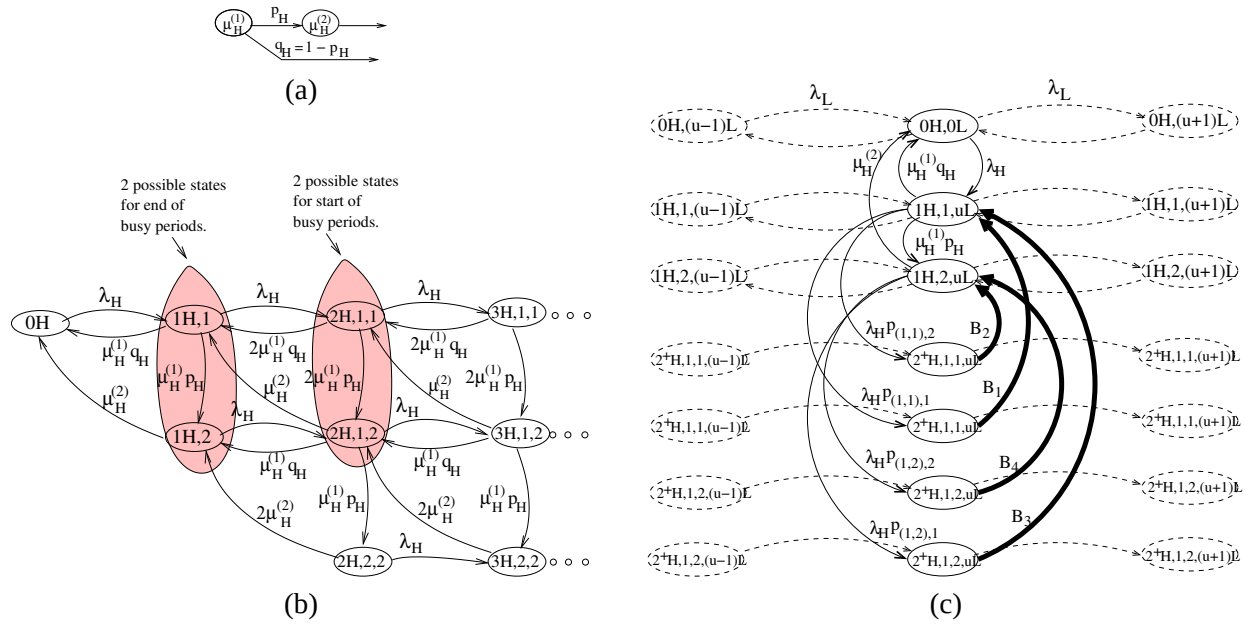


Figure 9.4: *(a) A 2-phase PH distribution with Coxian representation. (b) Markov chain which will be used to compute the high priority job busy periods, in the case where high priority job size have a PH distribution with Coxian representation shown in (a). (c) Chain for a system with two servers and two priority classes where high priority jobs have Coxian service times.*

9.2.4 Computing higher moments of response time

Throughout our discussion of RDR thus far, we have been concerned with computing the mean per-class response time. It turns out that computing higher moments of per-class response time is not much more difficult. Before we present our approach, we make two remarks. First, observe that it is trivial to derive all moments of the steady-state per-class *number of jobs* in the system, directly from the steady-state probabilities for the Markov chain, which we have already computed. Unfortunately, however, we cannot apply the beautiful generalization of Little's Law to higher moments (see [234, 31]) to immediately get the per-class higher moments of response time, since jobs do not necessarily leave our system in the order in which they arrive.

Below we sketch our approach for computing per-class variance in response time for the case of two servers, two priority classes (H and L), and exponential service times. We will refer to Figure 9.1(c) during our discussion. For class H jobs, it is easy to compute the variance of their response time, since they are oblivious to class L jobs, and the variance of response time in an M/M/2/FCFS queue is well known (see page 529 in [104]). Thus we will concentrate on class L jobs.

Consider the 1D-infinite Markov Chain shown in Figure 9.1(c) that tracks the number of class L jobs. We use the limiting probabilities to condition on what a class L arrival sees. Specifically, by PASTA (Poisson Arrivals See Time Averages) a class L arrival with probability $\pi_{(iH,\ell L)}$ will see state $(iH, \ell L)$ when it arrives, and will cause the system state to change to $(iH, (\ell+1)L)$ at that moment.

To calculate the variance in response time seen by this "tagged" (class L) arrival, we remove all the λ_L

arcs from the Markov chain in Figure 9.1(c), so that there are no more class L arrivals. This enables us to view the response time for the tagged arrival as the first passage time of this modified chain from state $(iH, (\ell+1)L)$ to the state where the tagged arrival departs. The only complexity is in figuring out exactly in which state the tagged arrival departs.

The tagged arrival may depart the modified Markov chain the first time it hits $(0H, 1L)$ or $(1H, 1L)$, depending on the sample path the chain follows. We will compute the passage time to go from state $(iH, (\ell+1)L)$ to one of these states { $(0H, 1L)$ or $(1H, 1L)$ }. It is important to observe that the first time we hit a state with 1L, cannot be state $(2^+H, 1L)$, by virtue of the fact that the Markov chain does not have decreasing arcs in its bottom rows.

If $(1H, 1L)$ is the first state that we hit with 1L, then we know that we must have gotten there from $(1H, 2L)$, which means that the single L job remaining is in fact the tagged arrival. (We're assuming preemption is done "youngest first to be preempted"). Thus we need to now add on the passage time to go from $(1H, 1L)$ to $(*, 0L)$ to get the full response time for the tagged arrival.

If $(0H, 1L)$ is the first state that we hit with 1L, then we know that we got there from state $(0H, 2L)$. In this case, the remaining 1L is equally likely to be the tagged arrival or not. With probability half, the tagged arrival is already gone, in which case we add nothing to the response time. With probability half, the tagged arrival remains, in which case we now add on the passage time to go from $(0H, 1L)$ to $(*, 0L)$ to get the full response time for the tagged arrival.

Observe that computing the moments of the above passage times is straightforward, since all the λ_L arcs have been removed.

9.2.5 A computationally efficient approximation

Clearly, the RDR method can become computationally intensive as the number of priority classes grows. This motivates us to introduce an approximation based on RDR called RDR-A. RDR-A applies to $m > 2$ priority classes and PH job size distributions.

The key idea behind RDR-A is that the RDR computation is far simpler when there are only two priority classes: H and L. In RDR-A, under m priority classes, we simply aggregate these classes into two priority classes, where the $m-1$ higher priority classes become the new aggregate H class and the m^{th} priority class becomes the L class. We define the H class to have a PH job size distribution that matches the first three moments of the aggregation of the $m-1$ higher priority classes.

The RDR-A method is similar to the MK-N approximation. The difference is that in MK-N, both the H and L classes have exponentially distributed service times. Thus under MK-N, the H class only matches the *first* moment of the aggregate $m-1$ classes, whereas under RDR-A *three* moments are matched. The reason that we are able to match the first three moments, rather than just the first, is that we have the RDR technique, which allows the analysis of multiserver priority queues with *PH* job size distributions, as described in Section 9.2.3.

9.3 Numerical validation and results

We now present numerical results for per-class mean response times in M/M/k and M/PH/k queues with $m = 4$ priority classes, derived using RDR and RDR-A, respectively. We will validate our results against

simulation and show that their relative error is small. Furthermore, the time required to generate our results is short, typically less than a second for each data point.

Figure 9.5 (top row) shows our results for per-class mean response times in an M/M/2 queue (a) and an M/PH/2 queue (b), both as a function of load ρ. The PH distribution used has two phases and $C^2 = 8$. All job classes have the same distribution, and the load is distributed evenly between the four classes. The left plot is derived using RDR and the right plot using RDR-A. Observe that the M/PH/2 queue (right plot) results in higher mean response time than the M/M/2 queue (left plot), as expected. In both cases the mean response time of the lower-priority classes dwarfs that of the higher-priority classes.

Figure 9.5 (bottom row) shows the relative per-class error for our results, when compared with simulation. Throughout the paper we always show error in *delay* (queueing time) rather than response time (sojourn time), since the error in delay is proportionally greater. We define relative error as

$$\text{error} = 100 \times \frac{\text{(mean delay by RDR or RDR-A)} - \text{(mean delay by simulation)}}{\text{(mean delay by simulation)}} \quad (\%).$$

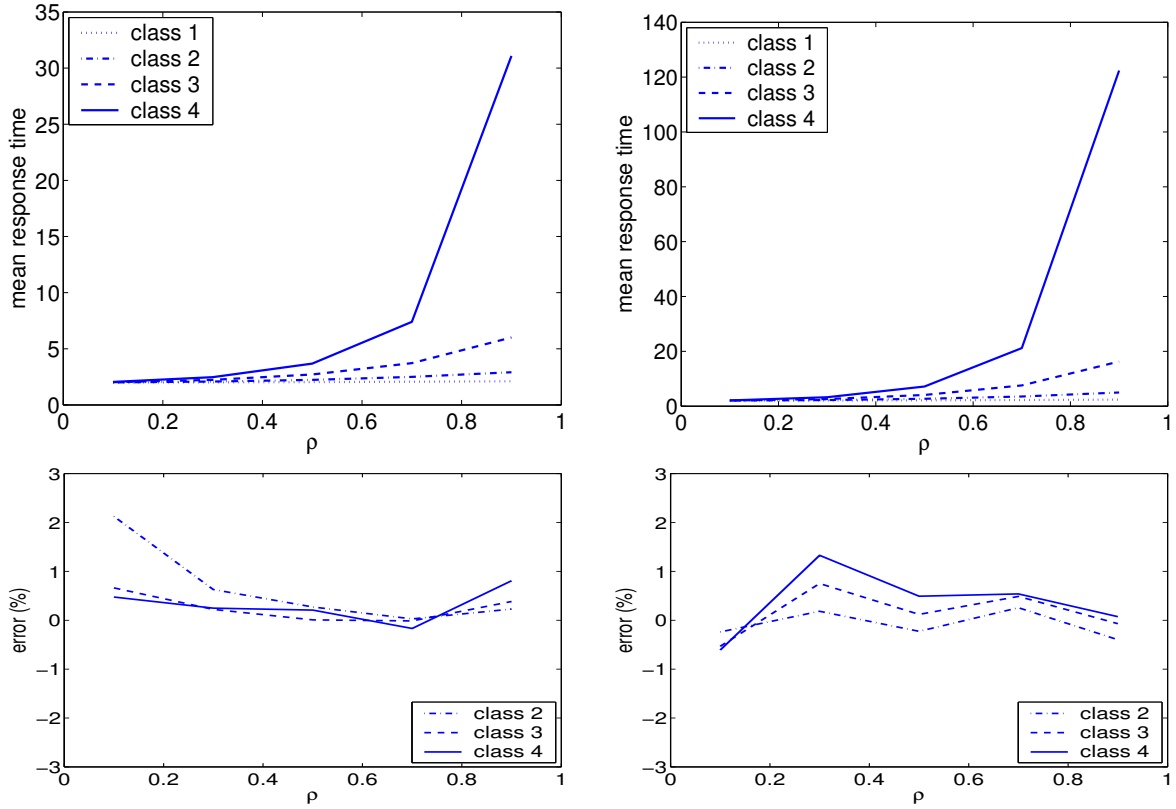
We only show the error for classes 2, 3, and 4, since our analysis is virtually exact for class 1 (solved via matrix-analytic methods). We see that the relative error in the mean delay of RDR and RDR-A compared to simulation is within 2% for all classes and typically within 1%, for all ρ's (the jaggedness of the figure is due to the fact that error is only evaluated at discrete loads). This error increases only slightly when we move to the case of priority classes with different means.

Figure 9.6 (top row) again uses RDR-A to calculate per-class mean response time in the M/PH/2 queue with four classes, but this time as a function of C^2, the squared coefficient of variation of the job size distribution. (Again, all classes have the same job size distribution). As we see from the figure, the per-class mean response time increases nearly linearly with C^2. Figure 9.6 (bottom row) shows the relative error in mean delay when the results of the RDR-A analysis in the left plot are compared with simulation. Again the error is under 2%. Again, this error increases only slightly when we move to the case of priority classes with different means.

Finally, we note that in the above computations RDR is much more computationally efficient than simulation. Simulation requires tens of minutes to generate each figure, since the simulation is run 30 times, and in each run 1,000,000 events are generated. By comparison our analysis takes only a few seconds for each figure. Further, if we try to reduce the number of events in the simulation to 100,000 events, in order to speed it up, we see five times as much variation in the simulation around our analytical values.

9.4 The impact of prioritization in an M/PH/k

Using RDR and RDR-A, we can now study the impact of prioritization in multiserver systems. We perform three studies of the impact of prioritization. First, in Section 9.4.1 we study the interaction of the effect of prioritization and the number of servers in the system. Thus, we characterize how the impact of prioritization changes as the number of servers in the system grows. Then, in Section 9.4.2, we evaluate the effect of prioritization schemes that favor short jobs ("smart" prioritization schemes) in multiserver systems. This is an important interaction to study because multiserver systems inherently allow some small jobs to bypass large jobs using alternate servers. Finally, in Section 9.4.3 we contrast the behavior of dual priority systems with that of systems with more than two priority classes. This is an important study because it provides



(a) M/M/2 with four classes

(b) M/PH/2 with four classes

Figure 9.5: *The top row shows per-class mean response time for the M/M/2 (a) and the M/PH/2 (b) with four priority classes. (a) is derived using RDR and (b) is derived using RDR-A. The bottom row shows the error in our analytically-derived mean delay relative to simulation results, for the corresponding graphs in the top row.*

an understanding of the effect of aggregating multiple priority classes into just two classes, which is the technique used in RDR-A to speed up the numerical evaluations.

9.4.1 The effect of the number of servers

We start by studying the interaction of prioritization and the number of servers in multiserver systems. Note that throughout these comparisons, *we hold the total system capacity fixed.* That is, we compare a single server of unit speed with a 2-server system, where each server has speed half, with a 4-server system, where each server has speed one-fourth, etc.

Figure 9.7 begins our study by considering an M/PH/k system with two priority classes where k is one, two, and four. The total system capacity is held fixed and load is fixed at $\rho = 0.8$. The low priority jobs are exponentially distributed. The high priority jobs follow a PH distribution where the squared coefficient of variation for high priority jobs, C_H^2, is varied. The means of the two classes are the same and the load is

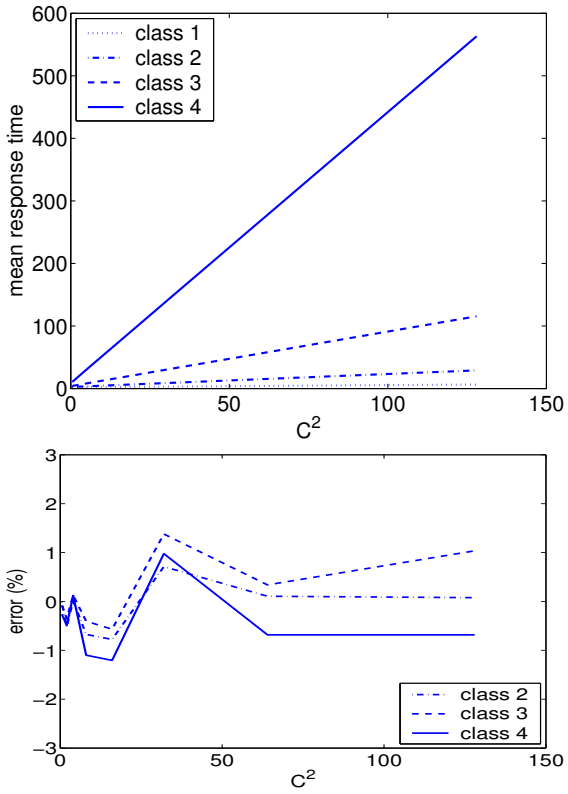
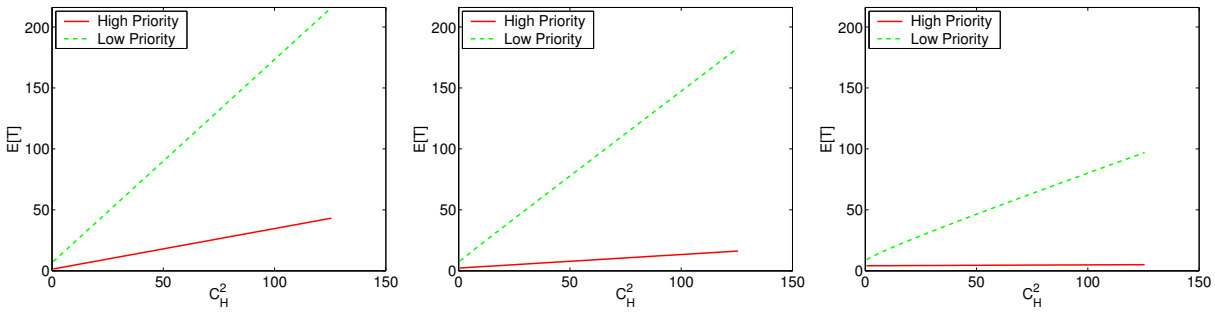


Figure 9.6: *The top row shows the per-class mean response times for the M/PH/2 with four priority classes derived via RDR-A analysis. The bottom row shows the relative error in the analytically-derived mean delay compared with simulation.*

split evenly between the two classes. The plots show per-class mean response time as a function of C_H^2. All results are computed using RDR.

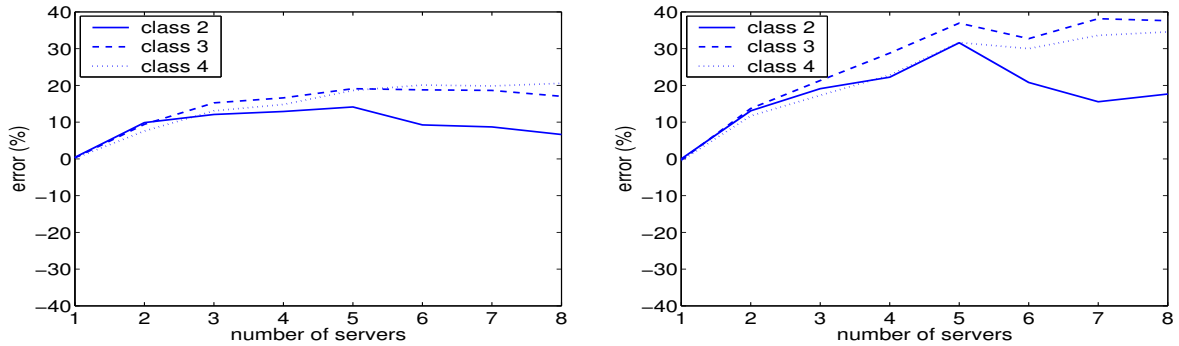
The first thing to observe is that the response times in the case of one server appear very different from the response times in the case of two servers or four servers. The effect of prioritization in a single server system offers little (quantitative) insight into the effect of prioritization in a multiserver system, aside from the fact that in all cases the response times appear to be a nearly linear function of C_H^2.

Figure 9.7 also illustrates some other interesting points. We see that as the number of servers increases, under *high* C_H^2, the performance of both high priority and low priority jobs improves. By contrast, under *low* C_H^2, the performance can get worse as we increase the number of servers (this fact is more visible in Figure 9.7 for the high priority jobs). To understand this phenomenon, observe that when C_H^2 is high, short jobs can get stuck behind long jobs, and increasing the number of servers can allow the short jobs a chance to be served. By contrast when C_H^2 is low, all jobs are similar in size, so we do not get the benefit of allowing short jobs to jump ahead of long jobs when there are more servers. However we do get the negative effect of increasing the number of servers, namely the under utilization of system resources when there are few jobs in the system, since each of the k servers only has speed $1/k$. The behavior under low C_H^2, where more



(a) 1 server (b) 2 servers (c) 4 servers

Figure 9.7: *Contrasting per-class mean response time under (a) one server, (b) two server, and (c) four server queues with two priority classes and PH service times. Total system capacity is fixed throughout, and* $\rho = 0.8$*. Results are obtained using RDR.*

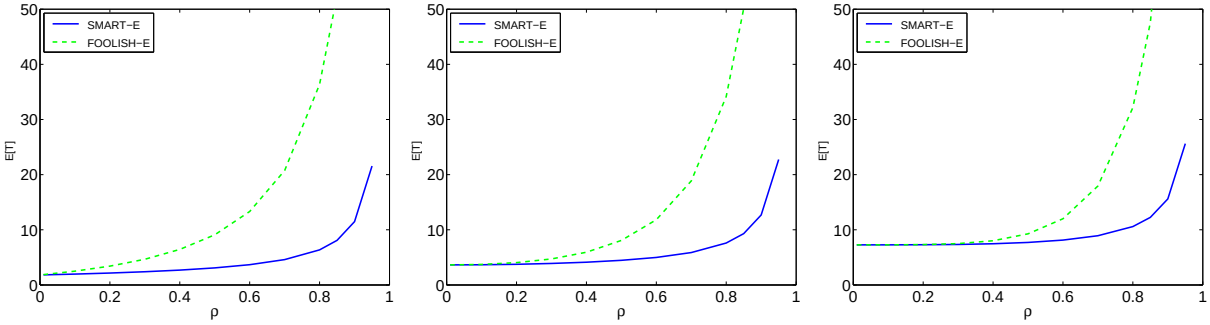


(a) $C^2 = 8$ (b) $C^2 = 25$

Figure 9.8: *Error in predicting mean delay using the BB approximation (compared with simulation) for an M/PH/2 with four classes where* $\rho = 0.8$ *and (a)* $C^2 = 8$ *or (b)* $C^2 = 25$*.*

servers lead to worse performance, is more prominent under lower load ρ.

Figure 9.7 already implies that the effect of prioritization on mean response time in a multiserver system may be quite different from that in a single server system. In Figure 9.8 we investigate this phenomena more closely, by evaluating the accuracy of the BB approximation [38], which is based on this assumption of similar behavior in single and multiserver priority queues. Looking at Figure 9.8, we see that the error in the BB approximation appears to increase for higher C^2 (right graph) and for more classes. With four classes and two servers, the error is already 10% when $C^2 = 8$ and higher for higher C^2. By contrast, for the same 4-class case as shown in figure 8, the error in RDR is always $< 2\%$ independent of C^2 and the number of servers. In the above graphs all classes were statistically identical. In the case where the classes have different means, the error in BB can be much higher, whereas RDR-A is much less insensitive to this.



(a) 1 server (b) 2 servers (c) 4 servers

Figure 9.9: *Mean response time under SMART-E versus FOOLISH-E prioritization in a 2-class system, where the classes are exponentially distributed with means one and ten respectively, for the case of one server, two servers, and four servers.*

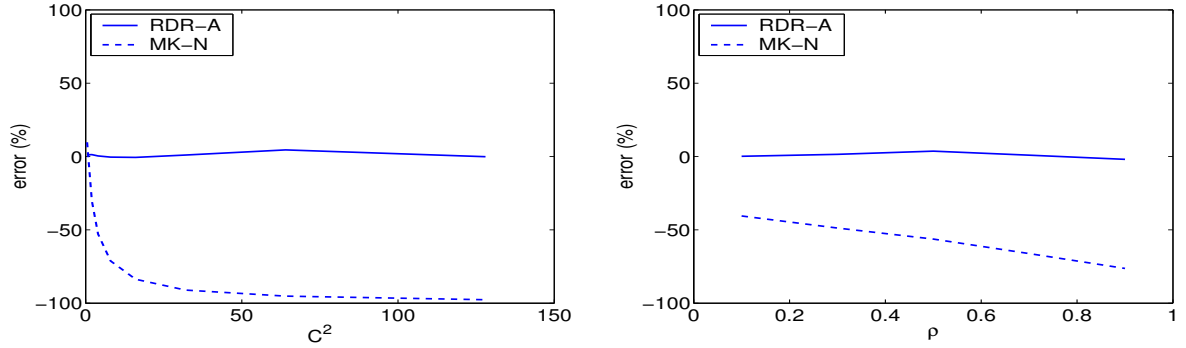
9.4.2 The effect of "smart" prioritization

Until now, we have assumed that all job classes have the same means. In this section and the next section, we remove this assumption. In particular, our goal in this section is to consider the effect of priority schemes which favor short jobs in multiserver systems. As we have seen throughout this thesis, biasing towards small jobs is a common method for improving mean response time. Here, we use RDR to understand how the benefit of favoring short jobs in a single server system compares to that for a multiserver system.

Figure 9.9 considers a job size distribution comprised of an exponential of mean 1, representing jobs which are "short" in expectation, and an exponential of mean 10, representing jobs which are "long" in expectation (where job sizes are measured in a single-server system). The probability of each type of job is chosen to split load evenly between the short and long jobs (e.g., with $m = 2$ classes, $\frac{10}{11}$ of the jobs are short and $\frac{1}{11}$ of the jobs are long). The SMART-E scheduling policy assigns high priority to jobs that are short in expectation, and the FOOLISH-E scheduling policy assigns high priority to the jobs that are long in expectation. Figure 9.9 shows the results for a (a) one server, (b) two server, and (c) four server system.

Looking at Figure 9.9, the SMART-E and FOOLISH-E policies are the same when load ρ is low. At low load, the response time for both policies converges to simply the mean job size, which in these figures is $\frac{20}{11}$ for the single server system, $\frac{40}{11}$ for the 2-server system, and $\frac{80}{11}$ for the 4-server system (recall that in a system with k servers, each server runs at $1/k$th the speed).

The most interesting observation is that more servers lead to less differentiation between SMART-E and FOOLISH-E schemes. For example, at load $\rho = 0.6$, there is a factor of five differentiation between SMART-E and FOOLISH-E with one server and only a 25% difference between SMART-E and FOOLISH-E with four servers. The effect appears more prominent under lighter load. This can be explained by recalling our earlier observation that multiserver systems allow short jobs a chance to jump ahead of long jobs, hence the negative effects of the FOOLISH-E scheme are mitigated.



(a) $\rho = 0.8$ (b) $C^2 = 8$

Figure 9.10: *An illustration of the impact of aggregating priority classes in multiserver systems. The graphs show error in mean delay of the 4th (lowest priority) class in the MK-N and RDR-A approximations for an M/PH/2 with SMART-E prioritization, as compared with simulation. (a) shows mean delay as a function of C^2 where $\rho = 0.8$, (b) shows mean delay as a function of ρ where $C^2 = 8$. The classes all have a 2-phase PH distribution with the same squared coefficient of variation C^2 and different means: 1, 2, 4, and 8.*

9.4.3 The effect of priority aggregation

Aggregation of multiple priority classes into two priority classes is a common approximation technique. We have used this idea in RDR-A, and as early as the 80's Mitrani and King (later followed by Nishida in the early 90's) proposed analyzing prioritization in a multiserver system via aggregation. The approach of Mitrani and King was as follows: obtain the mean response time of the m^{th} class by simply aggregating classes 1 through $m-1$ into a single high priority class and letting class m represent the low priority class. The above MK-N approximation required further approximating the single aggregate class by an *exponential* job size distribution, since it was not known how to analyze even a two class multiserver system with non-exponential job size distributions. Since RDR enables the analysis of multiserver priority queues with general PH job size distributions, we can reapply the MK-N aggregation idea, but where now we are able to capture the higher moments of the aggregated class. This approximation is what we have introduced as RDR-A, since it combines the use of RDR together with aggregation.

In this section, we will study the effect of priority aggregation. To accomplish this, we consider a two server system with four priority classes. All the classes have a two phase PH distribution, with varying squared coefficient of variation (C^2). The classes differ however in their mean, having means 1, 2, 4, and 8, respectively, and are prioritized according to the SMART-E scheme; classes with lower means have higher priority. (FOOLISH-E prioritization yields similar insights.) Figure 9.10 examines the error in the mean delay of the 4th class under RDR-A and under MK-N as a function of C^2 and as a function of ρ.

We see that the error in RDR-A is never more than 5% regardless of C^2 or ρ. By contrast, the error in MK-N is almost never less than 50%, and gets worse under higher load and C^2. We find experimentally that when the classes are identical, RDR-A incurs only slightly more error than RDR. This makes sense since aggregating identical classes does not incur additional variability. However when the classes are different, as in the case of SMART scheduling in Figure 9.10, the error can increase to 5% under RDR-A as ρ and C^2 are varied, while it remains below 3% for RDR over the full range of ρ and C^2 depicted in Figure 9.10.

The bottom line is that "aggregation into two classes" is a good method for approximating prioritization in multiserver systems where the number of classes is $m > 2$. However, the aggregation needs to be done carefully – the distribution of the aggregate class must be modeled more closely than can be captured by an exponential distribution. Thus another benefit of RDR is revealed; by allowing for PH job size distributions it enables more accurate approximations of multi-class systems via aggregation.

9.5 Designing multiserver systems

We will now apply the understanding of prioritization in multiserver systems that we have developed so far in order to approach the task of designing multiserver systems. In particular, one fundamental aspect of designing multiserver systems is the task of deciding between using a few fast (but expensive) servers or many slow (but cheaper) servers given a limited budget. By using fewer, faster servers one avoids underutilizing the servers when there are only a small number of customers in the system; however by using many, slower servers the variability of service demands has less of an impact.

In this section, we address this tradeoff by studying the following question:

> *Is it preferable to use a single fast server of speed s, or k slow servers each of speed s/k? What is the optimal k?*

Though this question uses a simplified cost model, it provides a clear view of the tradeoffs between using many cheap, slow servers and a few fast, expensive servers. Further, this question has been the focus of a stream of research [150, 220, 133, 200, 149, 207] (which we discuss in detail in Section 9.5.1). All of this prior work considers the question under FCFS scheduling. We will address the question in the FCFS setting and extend the discussion to the case of priority queues. Armed with our analysis of the M/PH/k dual-priority queue, we focus directly on questions involving choosing the optimal resource configuration. In particular we are interested in the following questions:

1. Under what conditions are multiple slow servers preferable to a single fast server? Is the optimal number of servers sensitive to changes in the relative arrival rates of the priority classes and changes in the variability of the service time distributions?
2. Does the answer to "how many servers are optimal" differ for the different priority classes? E.g., does the lower priority class prefer more or fewer servers than that preferred by the higher priority class?
3. How does the optimal number of servers in a *dual* priority system differ from the case when all jobs have been aggregated into a *single* priority class?
4. If one chooses a non-optimal number of servers, how does that affect the overall mean response time and the per-class mean response time?

9.5.1 Prior work

The question of how many servers are best has a long history, all assuming a single priority class. Throughout this section, we will assume that the performance metric of interest is mean response time rather than mean delay (queueing time) since it is clear that to minimize mean queueing time one wants an infinite number of servers [50, 51].

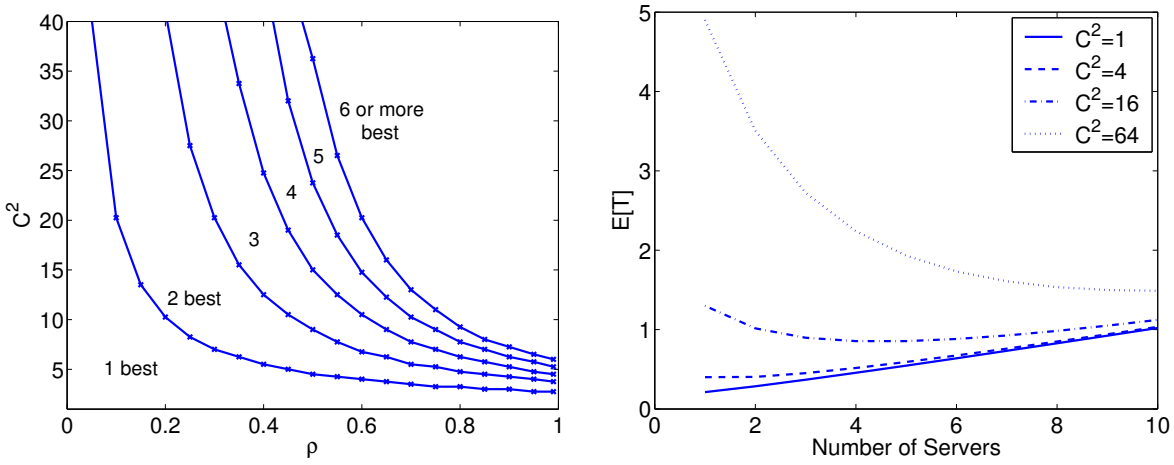


Figure 9.11: *The case of a single priority class. (a) The optimal number of servers as a function of the load, ρ, and the variability of the job size distribution, C^2. (b) Mean response time, $E[T]$, as a function of the number of servers at various job size variabilities ($C^2 = 1, 4, 16, 64$) for a fixed $\rho = 0.6$.*

As early as 1958 Morse observed that for an M/M/k system the optimal number of servers is one [150]. This was formalized by Stidham [220], who showed that under a general arrival process, and service times that are exponential, Erlang, or deterministic, a single server minimizes the expected number in system. Likewise, for a single server, Mandelbaum and Reiman [133] show that one server is best in the extreme cases when traffic is very light, regardless of job size variability. So, not only is variability important, but so too is traffic intensity. Scheller-Wolf [200] characterizes the effect of traffic intensity. He shows that under so-called power-law service times, moments of response time may move from infinite to finite as a function of both the number of servers and the traffic intensity. Very recently, Molinero-Fernandez et al. [149] consider the question of how many servers are best in an M/HT/k single priority system, where HT denotes a *heavy-tailed* service distribution. To answer this question, they approximate a heavy-tailed distribution with a bimodal distribution (BM), and then provide an approximate closed-form analysis of the M/BM/k queue, which they argue provides a reasonable approximation of the M/HT/k queue. The question of how many servers is best has also been considered via simulation in the context of an M/G/k queue by [207]. None of the above work considers priorities.

9.5.2 How many servers are best in a FCFS system

To begin, we consider the simplified problem of determining the number of servers that minimizes the mean response time under just *one* priority class. The M/PH/k/FCFS queue is easily analyzable via matrix analytic methods [125], as its Markov chain has a state space infinite in only one dimension.

Figure 9.11(a) shows the optimal number of servers as a function of the load and the variability of the job size. All of our results are expressed as a function of the variability of the job size distribution, and the server load. While other factors, e.g., the exact form of the distribution might affect our results, we posit that load and variability will be the most relevant factors.

Observe that under high job size variability and/or high load, the optimal number of servers is more than 1; we prefer k slow servers to 1 fast server. For example, at load $\rho = 0.4$ and the squared coefficient of variation $C^2 = 20$, we see that 3 servers are best. Computations are only done for up to 6 servers — the level curves shown will continue into the upper right portion of the plot if larger numbers of servers are considered.

Figure 9.11(b) shows that for any particular job size variability, $C^2 > 1$, having a larger number of slower servers may reduce the mean response time up to a point, after which further increasing the number of servers increases the mean response time. To understand why, note that by increasing the number of servers (while maintaining fixed total capacity), we are allowing short jobs to avoid queueing behind long jobs — specifically, an arriving short job is more likely to find a server free. Thus increasing the number of servers mitigates variability, hence improving performance. If the number of servers is too great however, servers are more likely to be idle, under-utilizing the system resources.

This simple analysis of the single priority M/PH/k queue motivates questions about how having two priority classes changes the answer to the question of "how many servers?" and whether approximating the more complicated two priority system with a single priority system is feasible. These questions are central to the remainder of this chapter.

9.5.3 How many servers are best in a dual priority system

We set out to answer four questions about multiserver system design. We have already addressed the first of these. In answer to Question 1, we have seen that in both the case of single priority class and in the case of dual-priority classes multiple slow servers can be preferable to a single fast server. Further, we have seen that the preference depends on service time variability and system load. The reason why multiple slow servers are preferable under high variability job sizes is that they offer short jobs a chance to avoid queueing behind long jobs, which in turn lowers mean response time.

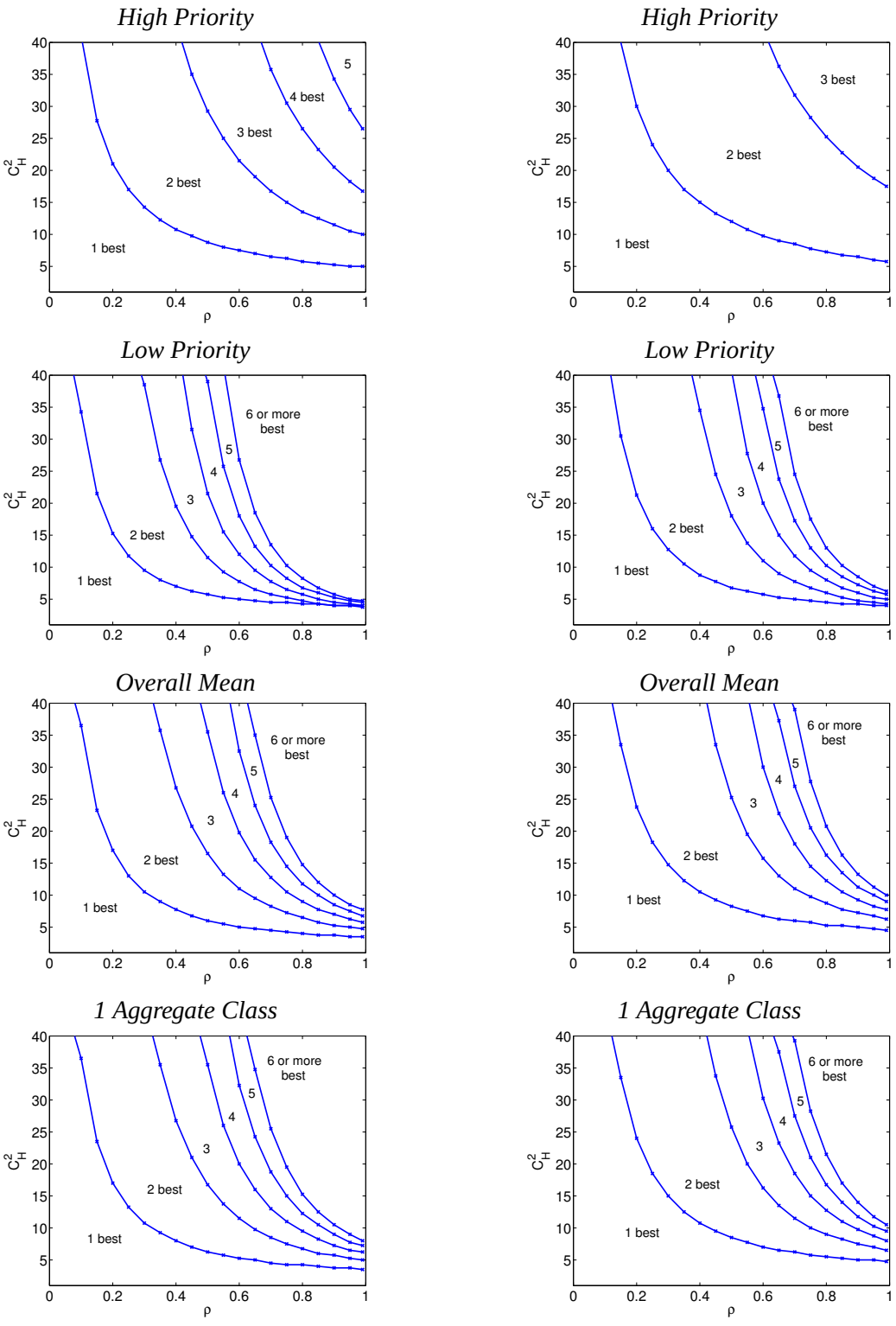
We will now focus our investigation on the three remaining questions. (Question 2) How does the answer the the question of "how many servers are optimal" differ among priority classes? (Question 3) How does a dual-priority system differ from its corresponding aggregate single priority system in terms of the optimal number of servers? (Question 4) How much improvement in mean response time can choosing the optimal number of servers provide?

We find that the answers to these questions depend on the relative sizes and relative proportions (loads) of the classes, as well as the variability of high priority jobs. The number of servers preferred by low priority versus high priority jobs can vary widely. Moreover, the number of servers preferred in the dual priority case when averaged over both classes typically differs substantially from the number preferred for the single class aggregate case. Furthermore, the absolute effect on mean response time can be dramatic (ranging from a factor of 2 to 6) as the number of servers is varied. In all studied cases, there exists an "optimal" number of servers where using fewer or more servers results in worse performance under highly-variable service distributions.

9.5.3.1 Evaluation setup

We split up our evaluation into 3 cases, depending on the relative sizes of high and low priority jobs:

(i) The mean size of high priority jobs equals that of low priority jobs: $E[X_H] = 1$, $E[X_L] = 1$.
(ii) The mean size of high priority jobs is smaller than that of low priority jobs: $E[X_H] = 1$, $E[X_L] = 10$.



(a) $\rho_H = \rho_L$ (b) $2\rho_H = \rho_L$

Figure 9.12: *How many servers are best when the two priority classes have the same mean job size ($E[X_H] = 1$, $E[X_L] = 1$)?*

(iii) The mean size of high priority jobs is larger than that of low priority jobs: $E[X_H] = 1$, $E[X_L] = 1/10$.

Note that the mean service time changes depending on how many servers are in the system (1 fast server or k slow servers) so that the systems are comparable. The values specified are the values for the maximum number of servers used in each plot and the mean sizes for each of the other number of servers is scaled appropriately.

Throughout our evaluations, we will consider a range of variability in the high priority job sizes, typically shown on the y-axis, and a range of load typically shown on the x-axis. The variability of the low priority job sizes is held constant ($C^2 = 1$). Observe that variability in the low priority jobs is less interesting since the low priority jobs only affect each other under preemptive resume. Lastly we also vary the proportion of the load made up by high priority and low priority jobs.

Some technicalities of the setup follow. In all the results shown, the high priority job sizes follow a 2-phase PH distribution with Coxian representation, allowing any variability $C^2 \geq 0.5$. When varying the proportion of load in each priority class, we vary the arrival rates of the classes only. In order to compare our results for the dual-priority system to the same system having a single aggregate class, we use a mixture of the two-phase PH high priority job size distribution and the exponential low priority job size distribution to obtain the overall job size distribution aggregated across both classes.

9.5.3.2 Discussion

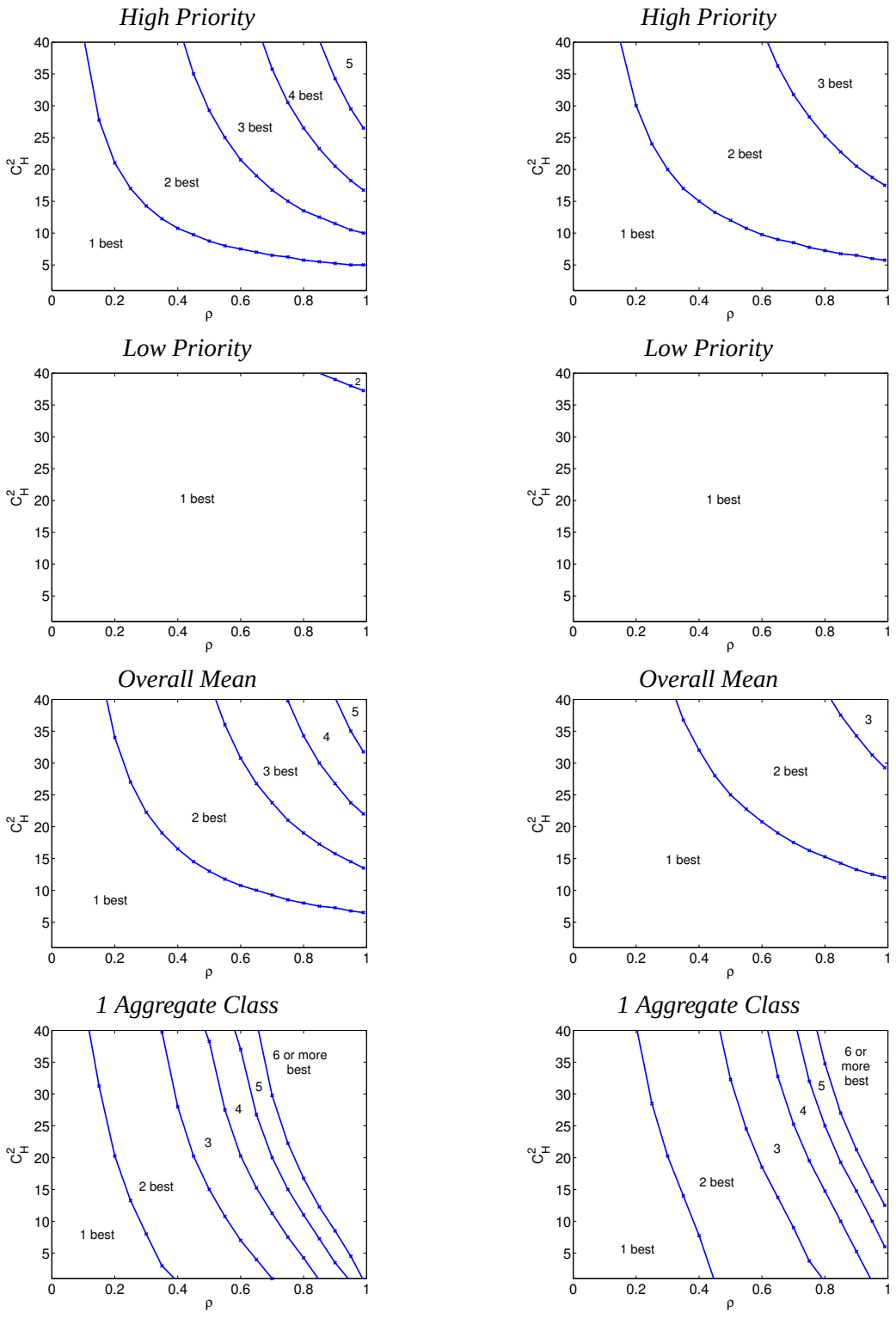
Figures 9.12, 9.13, and 9.14 illustrate the results of our analysis for the three cases described above: (i) $E[X_H] = 1$, $E[X_L] = 1$; (ii) $E[X_H] = 1$, $E[X_L] = 10$; and (iii) $E[X_H] = 1$, $E[X_L] = 1/10$ respectively. For each figure column (a) shows the case where the load made up by high and low priority jobs is equal, and column (b) shows the case where $\rho_H < \rho_L$. We also discuss, but omit showing, the case where $\rho_H > \rho_L$. For each figure we consider both the case of dual-priority classes and the case of a single aggregate class.

Figure 9.12: Equal mean sizes

Looking at the topmost plot in Figure 9.12 column (a), we see that the high priority jobs do not always prefer one server. In fact in the case of higher variability and/or load, they may prefer five or more servers. This is to be expected based on our results in Section 9.5.2.

Surprisingly however, the number of servers preferred by low priority jobs (shown in the second plot in column (a)) is much greater than that preferred by high priority jobs. Although only up to six servers are considered in these plots, we will see in later plots (Figure 9.15(b)) that the difference in the number of servers preferred by low and high priority jobs can be more than 10 servers. Low priority jobs prefer more servers because low priority jobs are preempted by high priority jobs and thus their mean response time improves with more servers, which allows them to escape from the dominance of high priority jobs.

The preferred number of servers with respect to the overall mean response time (the average of all jobs, including both low and high priority jobs) is shown in the third plot in column (a), where we see that the number of servers preferred by the overall mean, as expected, is a hybrid of that preferred by low and high priority jobs. Note though that this hybrid is more weighted toward the preference of low priority jobs because adding extra servers only hurts high priority jobs a small amount; whereas adding extra servers helps low priority jobs enormously. Interestingly, the number of servers preferred with respect to the overall mean is nearly identical to that shown for a single aggregate class of high and low priority jobs, shown in the bottom most plot in column (a). To understand why, observe that all jobs in this case have the same mean, and thus prioritizing in favor of some of them over others does not affect the mean response time greatly.



(a) $\rho_H = \rho_L$ (b) $2\rho_H = \rho_L$

Figure 9.13: *How many servers are best when the high priority jobs have a smaller mean job size ($E[X_H] = 1$, $E[X_L] = 10$)?*

Even though the classes have different variabilities, that is a smaller-order effect. This will not remain true in general.

Moving to column (b) of the same figure, we see that the same trends are evident when the high priority jobs make up a smaller fraction of the load. However, the specific numbers are quite different. For example, in the topmost plot of column (b), we see that the number of servers preferred by high priority jobs is much lower. An explanation of this is that the high priority jobs only interfere with each other and they are fewer in number in column (b) than in column (a); thus they want fewer, faster servers.

Less obvious is the fact that the number of servers preferred by low priority jobs in column (b) is also fewer than that in column (a). This follows from the same reasoning; the low priority jobs are most strongly affected by preemptions from high priority jobs, and with fewer high priority jobs, there are fewer interruptions and thus fewer servers are needed to avoid queueing behind high priority jobs.

Since both the high and low priority jobs in column (b) prefer fewer servers than in column (a), it makes sense that their overall mean (shown in the third plot of column (b)) also indicates that fewer servers are desired. This third plot also matches the bottom most plot in column (b) consisting of a single-class aggregation of high and low priority jobs, for the same reason explained above – that jobs have the same mean.

Not shown in Figure 9.12 is the case where high priority jobs comprise more of the load. In this case, both classes prefer more servers and, therefore, the mean of the two classes also prefers more servers. The reason for this is the converse of the above situation – there are more high priority jobs, and therefore they see more interference and want more servers. Further, the low priority jobs are preempted more frequently by high priority jobs and therefore also want more servers to alleviate the effect. Again the single aggregate class looks very similar to the two priority class overall mean.

Figure 9.13: High priority class has smaller mean

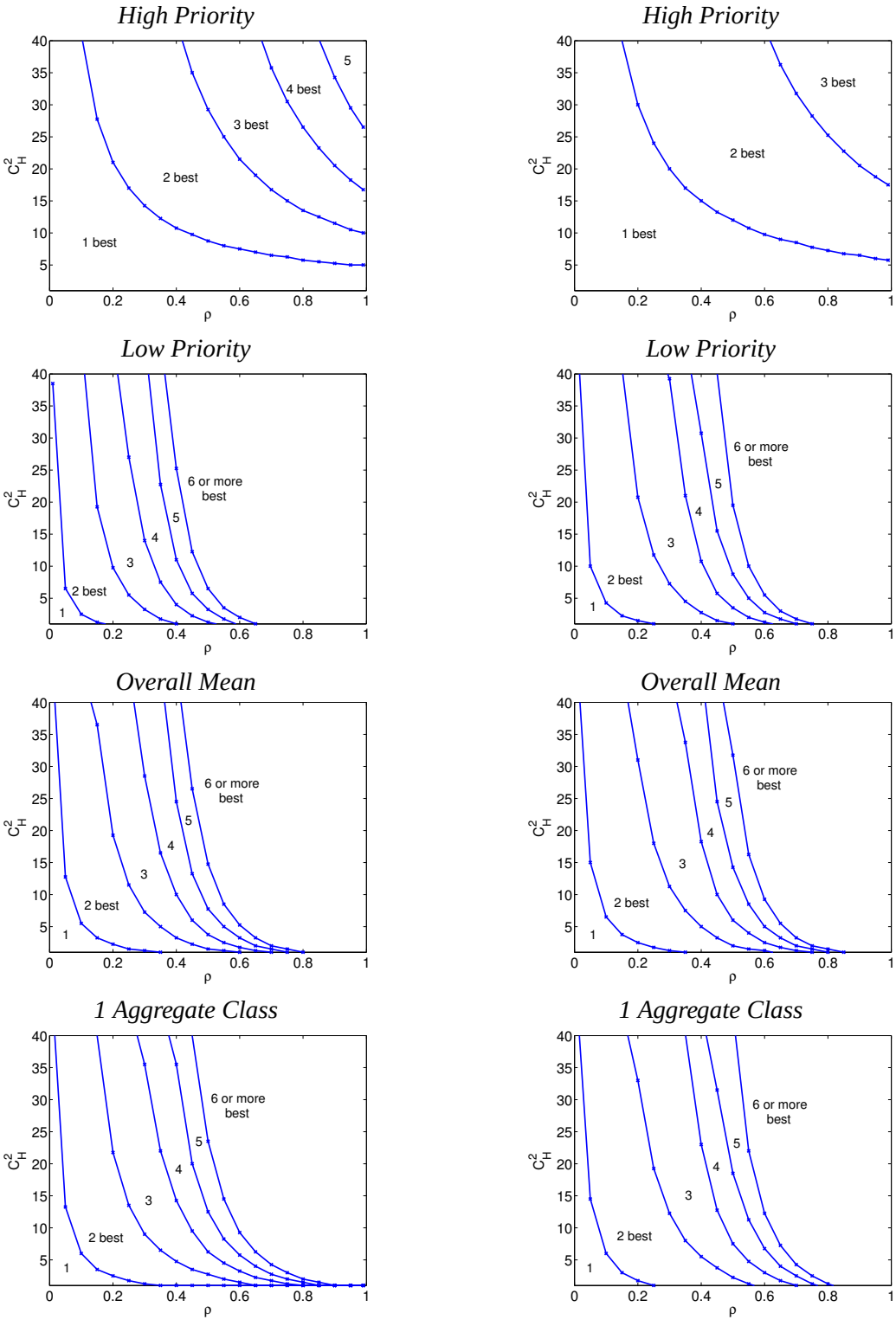
Moving to Figure 9.13, we continue to hold the mean high priority job size at 1 and increase the low priority job size to 10. Here, giving high priority jobs preference schedules the system more efficiently with respect to minimizing the overall mean response time.

Notice that the preferred number of servers for the high priority jobs is identical to that in Figure 9.12 because the high priority job size distribution is unchanged. However, the number of servers preferred by low priority jobs is now very different: they almost always prefer only one server. This follows from the fact that there are very few low priority jobs; so there is unlikely to be more than one low priority job in the system at a time. Thus, low priority jobs prefer a single fast server.

The overall preferred number of servers, averaged over the two priority classes, is again a hybrid of the preferences of the two classes, but this time is biased toward the preferences of the high priority jobs because they are in the majority, implying a preference for fewer servers than the corresponding graph in Figure 9.12. Recall that adding servers is a way to help small jobs avoid queuing behind larger jobs. Since we are in the case where small jobs have priority already, we do not need the effect of multiple servers. *Thus, in this case, priority classes can be viewed as a substitute for adding more servers.*

Comparing the overall preferred number of servers for the case of dual priorities with that preferred under a single aggregate class, we see that this time there is a significant difference in preferences. The single aggregate class prefers many more servers. This again is a consequence of the fact that in this case prioritization is a substitute for increasing the number of servers.

Column (b) of Figure 9.13 illustrates the same graphs for the case where the high priority jobs comprise



(a) $\rho_H = \rho_L$ (b) $2\rho_H = \rho_L$

Figure 9.14: *How many servers are best when the high priority class has a larger mean job size ($E[X_H] = 1$, $E[X_L] = 1/10$)?*

less of the total load. The trends are the same as in column (a); however the preferred number of servers is significantly smaller in all figures. This follows from the same argument as that given for column (b) of Figure 9.12. In the case (not shown) where high priority jobs make up a greater proportion of the total load, the number of servers preferred is, as before, always higher than in column (a).

Figure 9.14: High priority class has larger mean

In Figure 9.14 column (a), we once again hold the mean high priority job size fixed at 1 and now assume the low priority job sizes have a mean size of $1/10$. This case differs from the prior figure because now we are giving priority to the larger job sizes: this reduces efficiency and, consequently, many more servers will be needed in this case.

Once again, looking the topmost plot in column (a), we see that the preferred number of servers for high priority jobs is unaffected, since the high priority mean job size distribution has not changed. The low priority jobs, shown in the second plot of column (a), have vastly different preferences from the prior case. Here the low priority jobs prefer a very large number of servers; whereas in Figure 9.13 they almost always preferred one server. Because the low priority jobs are very small compared to the high priority jobs, they want more servers in order to avoid being blocked, and forced to queue behind the large, high priority jobs.

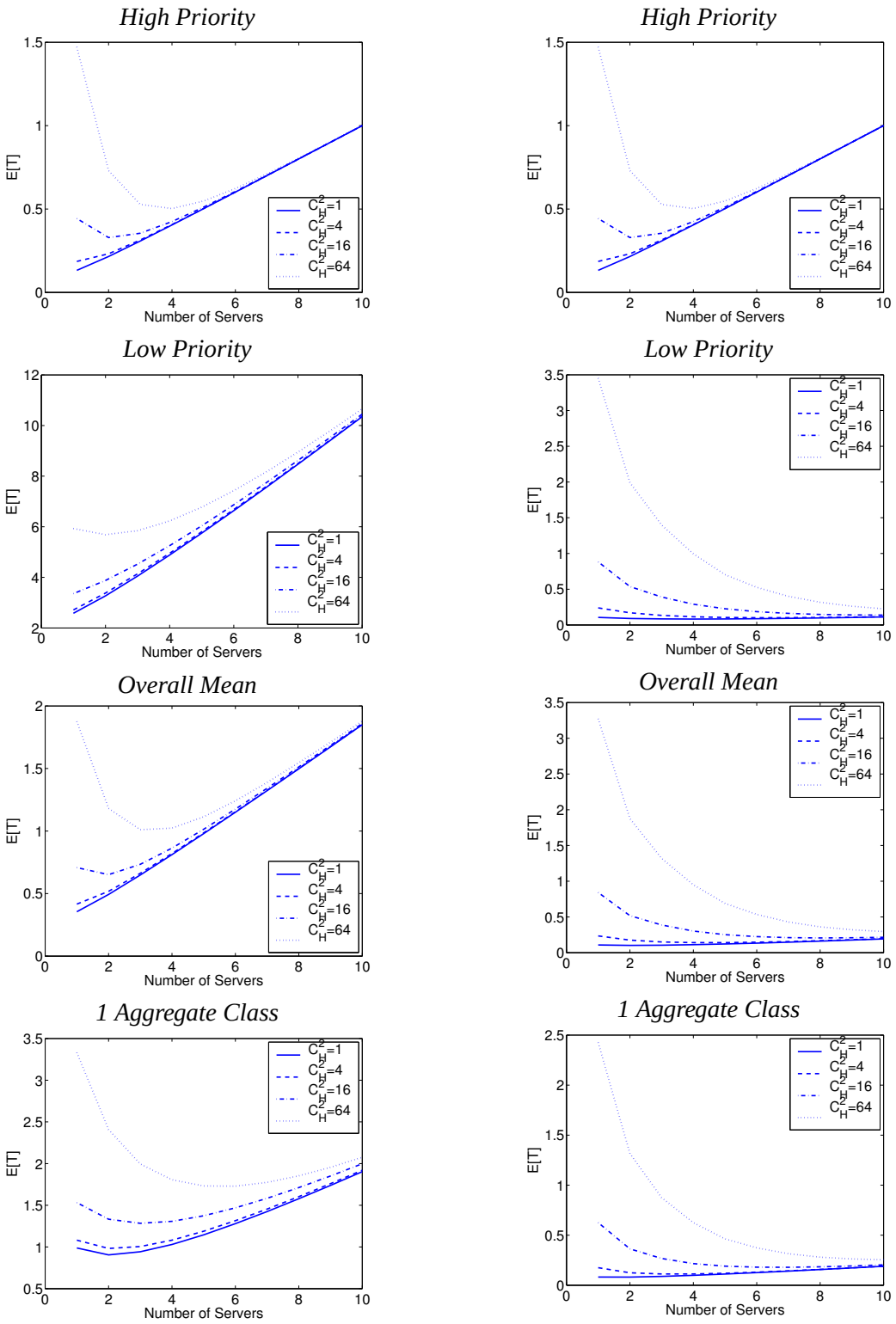
The preferred number of servers for the overall mean response time in the dual-priority system, shown in the third plot of column (a), is again a hybrid of the preferences of the low and high priority jobs, but this time is strongly biased toward the low priority jobs because there are more of them. Notice therefore, that the number of servers preferred is much greater in this case. Comparing this with the single class aggregate, we see that the single class prefers slightly fewer servers than the dual class overall mean. This is due to the fact that the prioritization toward large jobs in the dual class system is inefficient. Note that because this case of prioritization is inefficient, the multiple servers provide an even larger benefit than in the other cases.

Column (b) of Figure 9.14 illustrates the same graphs for the case where the high priority jobs comprise less of the total load. The trends are the same as in Column (a); however the preferred number of servers is significantly smaller in all figures. This follows from the same argument as that given for column (b) of Figure 9.12. In the case (not shown) where high priority jobs make up a greater proportion of the total load, more servers are preferable.

Figure 9.15: Response time as a function of the number of servers

In all the prior results figures, we were concerned with determining the optimal number of servers as a function of system load and the variability of high priority jobs. Although we sometimes found k servers to be better than 1 server, we never looked at the actual mean response time as a function of the number of servers. In Figure 9.15 we do so, ranging the number of servers from 1 to 10. The key points made by this figure are that: (i) the mean response time of both priority classes is sensitive to the number of servers and (ii) increasing the number of servers may reduce mean response time up to a point; however making the number of servers too large increases mean response time – thus forming a "U-shape." This figure also reinforces the prior message that *the greater the variability of the high priority jobs, the greater the number of servers needed to mitigate this variability*.

Figure 9.15 is divided into two columns: column (a) considers the job size distribution shown in Figure 9.13 and column (b) considers the distribution shown in Figure 9.14. In the previous figures, we have already discussed the differences in the number of servers preferred by each class. This same information can be read off of Figure 9.15 by observing that each of the plots in the figure have a "U-shape" and the bottom of



(a)$E[X_H] = 1$, $E[X_L] = 10$ (b)$E[X_H] = 1$, $E[X_L] = \frac{1}{10}$

Figure 9.15: *Mean response time as a function of the number of servers, which range from 1 to 10. The system load in these plots is* $\rho = 0.6$*, with* $\rho_H = \rho_L = 0.3$*.*

the "U" indicates the optimal number of servers.

Figure 9.15 however makes the following additional points. First, we see that, under high variability ($C^2 = 64$), the difference in the overall mean response time between the case of 1 server and the optimal number of servers is about a factor of 2 in column (a) and, even more, close to a factor of 6 in column (b). Thus, variability does play a crucial role, imperfectly explored in prior research. Second, we see that, whereas in column (a) the optimal number of servers is quickly reached, in column (b) the optimal number of servers is in some cases greater than 10, not even appearing on the plot. Thus, how prioritization is performed has a large impact on how the system should be designed.

9.6 Concluding remarks

Motivated by the growing adoption of multiserver designs in computer systems, our goal in this chapter is to begin to understand the impact of scheduling in multiserver systems and to contrast this impact with the results we have obtained in the single server setting. However, the analysis of multiserver queues is known to be difficult even under FCFS scheduling, so we could not study the multiserver scheduling in the generality that we have studied single server scheduling. Instead, we focused on one important scheduling mechanism, prioritization, and sought to understood its impact in multiserver systems. These results provide a first step towards the analysis of SRPT in the multiserver setting.

In order to study multiserver prioritization, we first developed a new analytic approach capable of studying multiserver priority queues with non-exponential service demands: the RDR technique. RDR provides the first near-exact analysis of an M/PH/k queue with $m \geq 2$ priority classes. The RDR algorithm is efficient (requiring only a second or two for each data point in the paper) and accurate (resulting in $< 2\%$ error for all cases that we studied). Although the RDR algorithm is efficient when the number of priority classes is small, it becomes less practical when the number of priority classes grows (e.g., for an M/M/2 with 10 priority classes, the running time can get as high as tens of seconds). Hence we also introduce the RDR-A approximation, which works by aggregating the $m > 2$ priority classes into only two priority classes.

This new analysis allows us to obtain insights about priority queueing in multiserver systems. We find that the effect of prioritization in multiserver systems differs significantly from the effect of prioritization in comparable single server systems. The reason is that adding servers creates complex effects not present in a single server. For example, multiple servers provide a strong benefit in dealing with highly variable job sizes, but they also hinder performance under lighter load. This is especially evident when studying the effect of "smart" prioritization, where classes of jobs with smaller means are given priority over those with larger means. We find that, though "smart" prioritization is beneficial in both multiserver and single server systems, "smart" prioritization has a much stronger impact in a single-server system than in a multiserver system of equal capacity. This can be explained in part by the observation that multiple servers inherently aid short jobs by allowing them to jump ahead of long jobs.

Our analysis also allows us to approach the question of *system design* in the multiserver setting. In particular, we illustrated that choosing the correct number of servers can improve mean response time dramatically, and we provided a number of guiding principles that can aid in determining the optimal number of servers in both single and dual priority systems. Further, we illustrated how the optimal number of servers is affected by the heuristic used to prioritize.

Aside from improving mean response time, choosing the number of servers carefully can also ease the

pain experienced by low priority jobs. It is possible to mitigate the penalty to low priority jobs (particularly the penalty caused by high variability of high priority job sizes), by choosing a server configuration which is more favorable to low priority jobs, typically one with more servers. This can often substantially improve the mean performance of low priority jobs without being significantly detrimental to high priority jobs; thus it aids in limiting the "unfairness" experienced by low priority jobs. Intuitively, having more servers reduces unfairness because low priority jobs are not forced to wait for all higher priority jobs to finish before being served, they just need to wait for one (of many) servers to become available.

The work in this chapter highlights the interaction between prioritization and multiserver system design. We illustrated that prioritization does indeed provide less dramatic improvements in mean response time in multiserver systems than in single server systems. However, we illustrated that prioritization is still beneficial in multiserver systems in practical settings. Further, we showed that prioritization in multiserver systems is more "fair" than prioritization in single server systems. So, though prioritization in multiserver systems does not provide the same efficiency gains as in single server systems, the gains provided generally come with a lower degree of "unfairness."

PART V

Impact and future directions

CHAPTER 10

Conclusion

Scheduling policies are at the heart of a wide array of computer systems. Whenever a resource is demanded by a number of users, a scheduling policy determines the order in which the resource is allocated. The study of scheduling policies has a long history including a vast literature of analytic results, but in recent years, the field has been going through a resurgence. This resurgence is a result of a variety of scheduling success stories in computer systems. In particular, at all levels of computer systems, designers dramatically reduced user response times by making small changes to the scheduling policy used at the bottleneck resource. There are examples of scheduling success stories in web servers [96, 182], routers, [179, 180], wireless networks [102, 136], peer-to-peer systems [178], operating systems [74], databases [138, 139], and beyond. But, these scheduling success stories have highlighted a number of disconnects between the theoretical research studying scheduling policies and the needs of system designers, which we detailed in Chapter 1. As a result, the traditional analytic results about scheduling policies often do not immediately apply to the scheduling policies that are implemented in modern systems. *The goal of this thesis has been to develop a modernized theory of scheduling that can provide analytic results that apply to today's computer systems.*

In order to accomplish this goal, we have identified three categories of disconnects between the needs of system designers and what is provided by traditional theoretical results:

- **The idealized policies studied traditionally in theory cannot be used in practice.** For example, pure SRPT is never implemented in practice. Instead, the policies that are implemented (i) use estimates of remaining size, (ii) use only 5-10 priority levels, or (iii) are hybrids of SRPT and PS-type policies. Each of these variants of SRPT will provide response times that are larger than under pure SRPT, however traditional theoretic results do not provide any information about how much performance will suffer.

- **Many performance measures that are important in practice are not studied in theory.** Mean response time is typically the focus of theoretical scheduling research, and SRPT is optimal with respect to mean response time. However, in practice, QoS and fairness metrics are also important. Additionally, power management, reliability, and many other performance measures are important. Once these other measures are considered, SRPT is no longer the clear choice. Worries that SRPT is unfair to large job sizes due to its bias towards small jobs pervade. Similarly, worries about providing

good QoS guarantees for large job sizes are common. Traditional theoretical results cannot be used to address such worries.

- **The traditional, simplified theoretical models include many unrealistic assumptions.** The M/GI/1 model is at the heart of a majority of research studying the performance of scheduling policies, but both the assumptions of a Poisson arrival process (the M) and a single server (the 1) are often unrealistic. For example, real arrival processes tend to be bursty and real users tend to be interactive and impatient. Further, many modern system designs make use of multiserver architectures, e.g. server farms and multi-core processers. Though SRPT is optimal in the M/GI/1 setting, once one considers interactive, impatient users and multiserver settings, SRPT may no longer be the optimal policy for mean response time. Further, the traditional theoretical research does not study the performance of SRPT in these more complex settings.

In this thesis, we have provided a number of tools and results that allow us to begin to bridge each of these categories of disconnects:

- **Moving beyond idealized policies**
  We have seen that the idealized policies studied in theory are not used in practice and, instead, a wide variety of variants of these policies are used. Thus, traditional analytic results are inadequate for system designers. The approach we have developed is to move beyond the study of individual, idealized policies and to study the impact of scheduling heuristics and techniques instead. In particular, we have formalized many common scheduling heuristics and techniques as scheduling classifications and, additionally, proven bounds which hold for the performance of all scheduling policies in each of these classifications. For example, SRPT is characterized by the fact that it uses the scheduling technique of "prioritizing based on remaining sizes" to apply the heuristic of "prioritizing small jobs." So, instead of studying all the variants of SRPT used in practice, we have defined and analyzed a class of policies that prioritizes based on remaining sizes and a class of policies that prioritizes small jobs. This new style of scheduling research is motivated by the fact that, though the idealized policies studied in theory are not used in practice, real system designs tend to apply the same heuristics and techniques found in the idealized policies. So, we can study these scheduling heuristics directly and avoid studying a huge array of non-idealized policies individually.

- **Moving beyond mean response time**
  Though mean response time is an important metric for computer systems, system designs must do more than provide small response times. We have seen that there are a wide variety of other performance measures that are also important. In this thesis we have focused on two performance measures of importance: fairness and the distribution of response time. In both cases, we not only provide new results for individual scheduling policies, we also provide the first analytic results for scheduling classifications. Further, it is important to point out that we have provided the *first* formal metrics for studying the fairness of scheduling policies.

- **Moving beyond the M/GI/1**
  Due to the difficulty in the analysis of scheduling policies, traditionally they have been analyzed (primarily) in the M/GI/1 queue. Though this model allows for general job sizes, the assumptions of Poisson arrivals and a single server are often overly restrictive. In this thesis, we have moved

beyond the M/GI/1 model and studied scheduling policies in settings where the arrivals are generated by interactive users and in settings where the system uses a multiserver architecture. In both of these settings, our work provides the first thorough study of the effectiveness of scheduling.

Obviously, we cannot provide a complete overview of the results from each chapter of the thesis, so we will instead provide a summary of a number of important lessons and surprises from the thesis (Section 10.1). Then, we will provide a few examples of the impact these lessons and surprises have had for both system design (Section 10.2) and theoretical research on scheduling (Section 10.3). Finally, we will summarize a number of future research directions motivated by the work in this thesis (Section 10.4).

10.1 Lessons and surprises

We have covered a wide array of different topics in this thesis, and in each topic our results have provided us with new, often surprising, lessons about the use of scheduling in computer systems. In this section, our goal is to conclude the thesis by summarizing a number of these lessons to remind the reader of the most important results in the thesis. Though we will only discuss the results at a high-level in order to keep the exposition crisp, we will provide references to the main results that lead to each lesson/surprise. Then, in Sections 10.2 and 10.3, we will provide examples of the impact of these lessons and surprises both for system designers and scheduling researchers.

All policies that give priority to small jobs perform well

Traditional theoretical results prove that SRPT is optimal for mean response time and that the improvement of SRPT over other policies is dramatic. However, as we have discussed, SRPT is not implemented in practice. Instead there are many variants of SRPT that have been suggested by system designers. However, these variants are not analyzed by the theory community. In this thesis, we have defined the SMART class to formalize the heuristic of "prioritizing small jobs" in a way that broad enough to include many practical variations of SRPT and still simple enough to be easy to apply. Further, we proved that all SMART policies have mean response time within a factor of 2 of optimal (SRPT), regardless of the load or service distribution (Theorem 4.2). Not only that, we proved that all SMART policies have an asymptotically equivalent response time distribution (Theorems 6.5 and 6.10). These results eliminate the need for researchers to analyze each individual variant of SRPT used in practice, since all such variants are captured by the results on SMART policies.

Job size estimates are enough

Though the SMART class includes many practical variations of SRPT, it is limited by the fact that all SMART policies must use exact job size information. This is a severe limitation because, in practice, it is common that applications are forced to use estimates of job sizes, for instance this is true in wireless networks and at web servers. To handle this case, we defined a generalization of the SMART class called SMART_ϵ that includes policies that prioritize small jobs using job size estimates. Further, we proved that all SMART_ϵ policies have mean response time that is within a constant of optimal across all loads and service distributions (Theorem 4.8). In addition, we characterized how this constant factor depends on the accuracy of the job size estimates and properties of the true service distribution (Corollary 4.11).

Using a finite number of priority levels may not be enough

In practice, system designs often cannot a continuum of priority levels, as SRPT requires (every possible remaining size is a different priority level). Instead, real designs often bin remaining sizes into 5-7 different priority levels in order to approximate SRPT. In this thesis we show that under some service distributions, policies that use a finite number of priority levels can still provide mean response times within a constant of optimal, for instance under Exponential (Theorem 3.3) or bounded (Corollary 4.12) service time distributions. However, this is not always the case. We also prove that under heavy-tailed distributions no finite number of priority levels is enough to provide response times within a constant of optimal under heavy traffic (Theorem 3.4).

If you're blind, pay attention to the workload

In some cases, computer systems do not even have estimates of job sizes. For instance, routers only know how much service has been given to a flow; they know nothing about how much more service will be required by the flow. In this case, we have seen that the optimal policy to use is strongly dependent on the workload. In particular, it has long been known that when job sizes have an increasing failure rate it is best to use FCFS and when job sizes have a decreasing failure rate it is best to use FB. However, the question of "which policy is best?" for intermediate distributions is unclear. We prove a number of new results that help to answer this question. In particular, we show that a key determinant as to which is better for mean response time is whether or not the service distribution is bounded (Theorem 3.23). Further, we prove that a key determinant as to which is better for the response time distribution is whether the service distribution is light-tailed or heavy-tailed (Section 6.2.4).

The tail behavior of the service distribution affects system design

In Chapter 6, we focused on understanding the tail of the response time distribution, and we found that there is a huge difference in the behavior of scheduling policies under light-tailed and heavy-tailed service distributions. We saw that no common scheduling heuristics or techniques is successful under both heavy-tailed and light-tailed service distributions. For example, all SMART policies minimize the response time tail under heavy-tailed service distributions (Theorem 6.5) but maximize under light-tailed service distributions (Theorem 6.10). In contrast, some non-preemptive policies minimize the response time tail under light-tailed service distributions, but all non-preemptive policies maximize the response time tail under heavy-tailed service distributions (Theorem 6.4). Thus, the tail behavior of the service distribution must play a key role in determining the scheduling policy used in computer system designs.

There is no universal notion of fairness

In Chapter 7, we provide the first formal definitions of fairness in the M/GI/1 queue. However, we illustrate that fairness is an amorphous concept, whose meaning depends on the context in which it is considered. As a result, we discuss and define two distinct notions of fairness: proportional fairness and temporal fairness. Proportional fairness refers to the idea that all job sizes should receive equitable service, i.e. no job size experiences response times disproportionate to its size (Section 7.5). Temporal fairness refers to the idea that it is fair to respect the seniority of jobs in the queue, i.e. it is in some sense unfair for a small job that just arrived to the queue to jump in front of the large job (Section 7.5). Though there are many other notions of fairness as well, these two cover the needs of a wide range of computer applications.

Prioritizing small jobs can help large jobs

The acceptance of designs based on SRPT and other SMART policies has often been hindered by worries that large jobs will be starved of service because of the priority given to small jobs. In other words, people worry that SRPT and SMART policies are unfair to large job sizes. Surprisingly, we prove in Section 7.1 that SMART policies can actually provide improved mean response time for all job sizes when compared with PS, which is often the status-quo in computer systems. For example, Theorem 7.4 shows that when the load ≤ 0.5, all job sizes prefer SRPT over PS when it comes to mean response time. Further, for all loads and service distributions, Theorem 7.19 shows that the largest job sizes are treated equivalently under PS, SRPT, and all SMART policies.

Pay attention to the interactivity of users

In Chapter 8 we discussed the impact of interactive users on the performance of scheduling policies. Interestingly, we found that user behavior has an enormous impact on the effectiveness of scheduling. In particular, if users have long interactive sessions with the server, then the scheduling policy used at the server has very little impact on the system performance. However, if users have only short interactive sessions, then the scheduling policy can have an enormous impact on system performance. This is a cautionary tale for system designers, since the degree of user interaction is typically not viewed as an important workload characteristic.

How many servers are best?

Our analysis in Chapter 9 allows us to approach the question of *system design* in the multiserver setting. In particular, we illustrated that choosing the correct number of servers can improve mean response time dramatically, and we provided a number of guiding principles that can aid in determining the optimal number of servers in both single and dual priority systems. Further, we characterized how prioritization affects the optimal number of servers. Aside from improving mean response time, choosing the number of servers carefully can also reduce the response times experienced by low priority jobs. It is possible to mitigate the penalty to low priority jobs by choosing a server configuration with more servers. This can often substantially improve the mean performance of low priority jobs without being significantly detrimental to high priority jobs; thus it aids in limiting the "unfairness" experienced by low priority jobs.

10.2 The impact for system design

Though the thesis has included almost entirely analytic results, the goal of this analysis was to bring theory closer to practice, and to provide results that apply more directly to real computer systems and, thus, can act as tools for system design. To illustrate how the results in the thesis can be used for system design, let us consider a few examples.

Example: Scheduling using job size estimates

In many applications, exact job size information is not known, but it is possible to estimate job sizes using some system measurement. For example, in web servers and wireless networks, designs that prioritize the job with the smallest estimated remaining size have been proposed [182, 131, 130, 102, 136]. But, in such designs a key question is "how much can response times be reduced by more

accurately estimating job sizes?" The reason this question is so important is that there are overheads involved in estimating the job sizes. For example, in a web server, estimating the network delay a request will experience requires using packet probing techniques.

In Section 4.2, we introduce the SMART$_\epsilon$ *classification to capture the effect of using job size estimates in order to prioritize. We provide simple bounds that provide an illustration of the tradeoff between the accuracy of job size estimates and the performance of the resulting policy. Further, our results expose the effect of the underlying job size distribution on this tradeoff.*
□

Example: Choosing a workload generator

Workload generators are an invaluable resource when evaluating the performance of proposed system designs. Most workload generators for web server and database workloads assume a closed system model, where new job arrivals are only triggered by job completions (followed by a think time). In contrast, whenever a trace is used to generate the workload, an open system model is implicitly assumed, i.e. new jobs arrive independently of job completions. Though every systems researcher is well aware of the importance of setting up one's experiment so that the system being modeled is "accurately represented;" system designers generally pay little attention to whether a workload generator is closed or open.

The work in Chapter 8 illustrates that there is a vast difference in behavior between the open and closed models in real-world settings. Not only is the measured response time different under the two system models, but the two system models respond fundamentally differently to variations in parameters and scheduling policies, e.g. the impact of scheduling is far more dramatic in the open model than in the closed model. Further, the differences between these two system models are present across a range of applications, including static and dynamic web servers, a database back-end, and an auctioning web site. The differences between the open and closed models motivate the need for system designers to be able to determine how to choose if an open or closed model is appropriate for evaluating new designs. As a result, we also provide a simple recipe for how to make this choice in Chapter 8.
□

Example: Online adaptation to time-varying workloads

Not only do scheduling classifications provide a way to analyze existing policies used in practice, they also provide a technique for improving existing designs. In particular, a defining aspect of computer system workloads is that they are time-varying, e.g. time of day effects result in certain periods being far busier than others at e-commerce web sites. As a result of time-varying workloads, the best scheduling policy to use also changes over time. But, providing adaptive scheduling policies is extremely difficult, and analyzing them is notoriously hard. Scheduling classifications provide a way to do both.

To illustrate this, let us consider an example: the SMART *class. One can imagine a parameterized version of the class where policy (i, j) gives priority to the job with the lowest $s^i r$, where s is the original size of the job and r is the remaining size of the job. Then, one could implement the* SMART *class as a policy by using machine learning techniques to adaptively choose the best (i, j). Not only will this approach lead to a policy that outperforms any static* SMART *policy, but the resulting adaptive policy will still be in the* SMART *class. Thus, all of the results we have proven*

about the SMART *class will hold about the adaptive policy, even as it adapts itself online.*
□

Example: Server farm design
An important question facing server farm designers is that of "how many servers are best?" In particular, given a fixed budget, the question is whether to use a small number of fast, expensive servers or to use a larger number of cheaper, slower servers.

Our results in Chapter 9 provide a number of guiding heuristics for designers. We illustrate that systems with fewer servers have an advantage over larger systems with respect to utilization, but pay a price when job sizes are highly variable. Specifically, when fewer servers are used is it more likely that some servers will sit idle unnecessarily, but it is also more likely that small jobs will become trapped behind larger jobs. Not only do our results provide these high-level heuristics for designers, they also provide a technique for obtaining exact results for which server configuration is optimal. In addition, they illustrate how the optimal configuration is affected by scheduling.
□

We could go on to list many other examples but, hopefully, these examples already make the point that the results in the thesis begin to bridge the disconnects between the needs of system designers and traditional analytic results.

10.3 The impact for theoretical scheduling research

Beyond the impact of the thesis for system design, the results in this thesis also present a number of new directions for theoretical research on scheduling – both by defining new performance measures and classifications to study and by developing new analytic techniques. To illustrate the impact to the theoretical community, let us consider a few examples.

Example: Studying Classifications
Following the introduction of the SMART *classification at the Sigmetrics conference in 2005 [241], many other researchers also became interested in scheduling classifications. This led to a collaboration with Bert Zwart and Misja Nuyens analyzing the distribution of response times under* SMART *policies in the large buffer large deviations regime (see Chapter 6) [161]. In addition, it led to a collaboration with Sanjay Shakkottai and Chang Woo Yang on the analysis of* SMART *policies in the many sources large deviations regime [248]. Further, other researchers have started to introduce their own scheduling classifications. For example, the* PROTECTIVE *class that we discuss in this thesis was introduced by Friedman and Hurley [79]. Further, Feng, Misra, & Rubenstein [74], Nunez-Queija & Kherani [118], and Kherani [117] have all introduced interesting classifications of other scheduling techniques and heuristics.*
□

Example: Defining Fairness
Our initial work defining a metric for studying the fairness of scheduling policies appeared in the Sigmetrics conference in 2003 [238], where it won the Best Student Paper Award. It has since been

cited over 50 times, has been used for many applications outside of web servers, and has served to jump-start a new focus on fairness in the theoretical scheduling community. Many researchers, e.g. Rai, Biersack, et al. [179, 180], Gong & Williamson [87, 88, 86], and Brown [47], have analyzed a wide array of policies with respect to the fairness measure we introduced. Still others, including Friedman & Henderson [78], have invented new policies that perform well with respect to this fairness measure. In addition, many researchers, such as Levy & Raz [185, 184, 20] and Sandmann [195, 196], have developed new fairness measures for use in other applications.
□

Example: Recursive Dimensionality Reduction

In Chapter 9 we provided the first near-exact analysis of multiserver priority queue using a new technique called Recursive Dimensionality Reduction (RDR), which serves to recursively use the solution to an $n-1$ dimensional Markov chain in order to solve an n dimensional Markov chain. In addition to its usefulness in studying multiserver systems, RDR has turned out to be useful in many other application domains. For example, it has been applied to analyze dispatching algorithms in multi-queue systems and affinity scheduling algorithms. In all, the technique has led to more than a dozen papers from our research group at Carnegie Mellon, e.g. [163, 94, 165, 242, 95].
□

Example: Large deviations analysis in the GI/GI/1

In Chapter 6, we provided the first GI/GI/1 large buffer, large deviations analysis of the SMART *class using a novel probabilistic approach based on an explicit random walk decomposition. Before this work, only the M/GI/1 analysis of an individual* SMART *policy,* SRPT*, was known. Further, the M/GI/1 analysis of* SRPT *depended on an explicit characterization of the moments of $T(x)$, and therefore was infeasible for use in the GI/GI/1 setting. Our new approach relies on purely probabilistic analysis and thus extends easily beyond* SMART *policies to* FB*,* FOOLISH *policies, and beyond.*
□

Again, we could go on to list many other examples, but hopefully, these examples already make the point that the results in the thesis provide a number of new directions for theoretical scheduling research. Specifically, these examples illustrate that the thesis has provided new models, new metrics, and new tools that can help the theoretical scheduling research community provide results that are applicable to computer system designs.

10.4 Further directions

The work in this thesis has begun to bridge the gaps between theoretical work on scheduling and the needs of practitioners, however there is much work that remains on this topic. We have studied a number of common scheduling heuristics and techniques, but there are other important heuristics to consider. We have studied a diverse set of metrics, but there are many other performance measures that are important in computer systems. In addition, we have studied two generalizations of the traditional M/GI/1 model, but there are many other practical complexities that need to be studied. To end the thesis, we will summarize a few of the open questions along each of these themes.

New classifications

One of the key contributions of this thesis is the introduction of "scheduling classifications" as a way to move beyond the analysis of individual, idealized policies and include the policies that are implemented in real system designs. To that end, we have introduced scheduling classifications that cover a wide range of common scheduling heuristics and techniques. However, there are many other interesting scheduling heuristics and techniques that one could formalize into scheduling classifications. For example, it would be interesting to define a scheduling classification that include PS variants such as Discriminatory Processor Sharing (DPS) or Generalized Processor Sharing (GPS).

More metrics

In this thesis we have gone beyond mean response time to study fairness and the distribution of response time, but there are obviously many other performance measures that are important to study. Even among the two metrics that we studied, there is much more interesting research to perform.

In particular, we studied one particular scaling of the distribution of response time, the large buffer large deviations scaling, but there are many other ways to study the distribution of response time. For example, as we described in Section 6.4, it is also interesting to study the distribution of response time in the many sources large deviations regime. Further, it would be interesting to understand more about the contrast of higher moments of response time, such as variance, across scheduling policies.

The story is similar with fairness. We introduced a number of new measures characterizing the fairness of scheduling policies, but fairness is such an amorphous concept that there are many other interesting aspects of fairness that are important to study. In fact, following our work on fairness a number of researchers have gone on to introduce other definitions of fairness, many of which we discuss in Section 7.6. But, it remains to find useful metrics for studying fairness of scheduling policies beyond the single server setting.

Beyond other measures of fairness and the distribution of response time, there are a wide variety of weighted response time measures that are important in practice, e.g. slowdown. Under these weighted response time measures many results about scheduling policies start to change. For instance, SRPT is not optimal for mean slowdown, though it is still within a factor of two of optimal [83]. As a result of these differences, much is left to understand about the performance of scheduling policies under weighted response time measures.

Other models

In this thesis we focused on two generalizations of the M/GI/1 queue – interactive users and multiserver systems – but there are many other important generalizations to consider. Let us highlight two others that are of particular interest: user impatience and stochastic service rates.

User impatience is an issue that affects almost every computer system. When users become frustrated by delay they tend to abandon their requests, e.g. hitting the refresh button in their browser or killing the process that is hanging. This abandonment can be a major design issue. For example, nearly 20% of internet traffic due to aborted service requests [251]. This high level of user abandonment has a large, negative impact on the behavior of the time-sharing policies like PS that are used in many computer systems because resources are wasted on users that later abandon due to impatience. However, since users with small requests tend to become impatient quickly while users with large requests tend to be more patient, it seems that applying the heuristic of prioritizing small jobs may limit the amount of wasted service. But, the benefits of such an approach are not understood.

Stochastic service rates are of fundamental importance for studying wireless networks and when studying power management. In wireless networks, channel conditions change over time, and thus the bandwidth available is stochastic. In such settings, scheduling policies that ignore the changes in service rates can pay a significant price in terms of throughput. Thus, scheduling policies need to be opportunistic and choose jobs that have high service rates when possible. Opportunistic scheduling is commonplace in wireless networks [102, 136], but the analysis of such policies is almost non-existent in the scheduling literature. In power management, the issue is not that the environment affects the service rate. Instead, the goal is to adjust the service rate in a way that conserves power. In particular, by scaling down or turning off the processor during periods of light load, the system can achieve significant power savings. Such techniques are increasingly important in CPU scheduling, data centers, and mobile devices [46, 152, 226], but there is very little analytic work studying these techniques.

Bibliography

[1] Webjamma world wide web (www) traffic analysis tools. http://research.cs.vt.edu/chitra/webjamma.html.

[2] The PSC's Cray J90's. http://www.psc.edu/machines/cray/j90/j90.html, 1998.

[3] The U.S. Geological Survey. http://www.usgs.gov, 2002.

[4] Carnegie Mellon School of Computer Science. http://www.cs.cmu.edu/, 2005.

[5] S. Aalto, U. Ayesta, and E. Nyberg-Oksanen. M/G/1 MLPS compared to M/G/1 PS. *Oper. Res. Letters*, 33:519–524, 2004.

[6] S. Aalto, U. Ayesta, and E. Nyberg-Oksanen. Two-level processor-sharing scheduling disciplines: Mean delay analysis. In *Proc. of ACM Sigmetrics-Performance*, 2004.

[7] J. Abate and W. Whitt. Numerical inversion of Laplace transforms of probability distributions. *ORSA J. on Computing*, 7(1):36–43, 1995.

[8] J. Abate and W. Whitt. A unified framework for numerically inverting laplace transforms. *INFORMS J. on Computing*, 18(4):408–421, 2006.

[9] B. Abi-Itzhak and A. Halfin. Server sharing with a limited number of service positions and symmetric queues. *J. Appl. Prob.*, 24:990–1000, 1987.

[10] M. A. Aczel. The effect of introducing priorities. *Oper. Res.*, 8:730–733, 1960.

[11] J. Almeida and P. Cao. Wisconsin proxy benchmark 1.0. http://www.cs.wisc.edu/cao/wpb1.0.html, 1998.

[12] W. Almesberger. Linux network traffic control — implementation overview. White paper available at http://diffserv.sourceforge.net/, 1999.

[13] C. Amza, E. Cecchet, A. Chanda, A. Cox, S. Elnikety, R. Gil, J. Marguerite, K. Rajamani, and W. Zwaenepoel. Specification and implementation of dynamic web site benchmarks. In *Workshop on Workload Characterization*, 2002.

[14] V. Anantharam. How large delays build up in a GI/G/1 queue. *Queueuing Sys.*, 5:345–368, 1989.

[15] M. Arlitt and T. Jin. A workload characterization study of the 1998 world cup web site. In *IEEE Network*, 2000.

[16] M. Arlitt and C. Williamson. Web server workload characterization: the search for invariants. In *Proc. of ACM Sigmetrics*, 1996.

[17] S. Asmussen. *Applied Probability and Queues*. Springer, 2003.

[18] K. Athreya and P. Ney. *Branching processes*. Springer, 1972.

[19] B. Avi-Itzhak and H. Levy. On measuring fairness in queues. *Adv. of Appl. Prob.*, 36(3):919–936, 2004.

[20] B. Avi-Itzhak, H. Levy, and D. Raz. Quantifying fairness in queueing systems: Principles, approaches and applicability. *Prob. in the Eng. and Info. Sciences*, in press.

[21] A. Baltrunas, D. Daley, and C. Kluppelberg. Tail behaviour of the busy-period of a gi/g/1 queue with subexponential service. *Under submission*, 2006.

[22] G. Banga and P. Druschel. Measuring the capacity of a Web server under realistic loads. *World Wide Web*, 2(1-2):69–83, 1999.

[23] N. Bansal. On the average sojourn time under M/M/1/SRPT. *Oper. Res. Letters*, 22(2):195–200, 2005.

[24] N. Bansal and D. Gamarnik. Handling load with less stress. *Queueing Systems*, 54(1):45–54, 2006.

[25] N. Bansal and M. Harchol-Balter. Analysis of SRPT scheduling: Investigating unfairness. In *Proc. of ACM Sigmetrics*, 2001.

[26] N. Bansal and A. Wierman. Competitive analysis of M/GI/1 queueing policies. Technical Report CMU-CS-02-201, Carnegie Mellon University, December 2002.

[27] N. Bansal and A. Wierman. Competitive analysis of M/GI/1 queueing policies. Technical Report CMU-CS-02-201, Carnegie Mellon University, 2002.

[28] P. Barford and M. Crovella. Generating representative web workloads for network and server performance evaluation. In *Proc. of ACM Sigmetrics*, 1998.

[29] P. Barford and M. Crovella. The surge traffic generator: Generating representative web workloads for network and server performance evaluation, 1998.

[30] M. Bender, S. Chakrabarti, and S. Muthukrishnan. Flow and stretch metrics for scheduling continous job streams. In *Proc. of the 9th Annual ACM-SIAM Symposium on Discrete Algorithms*, 1998.

[31] D. Bertsimas and D. Nakazato. The distributional Little's Law and its applications. *Operations Research*, 43(2):298–310, 1995.

[32] N. Bingham and R. Doney. Asymptotic properties of supercritical branching processes i: the Galton-Watson process. *Adv. in App. Prob.*, 6:711–731, 1974.

[33] N. Bingham, C. Goldie, and J. Teugels. *Regular Variation*. Cambridge University Press, 1987.

[34] T. Bonald and L. Massoulie. Impact of fairness on internet performance. In *Proc. of ACM Sigmetrics*, 2001.

[35] T. Bonald and A. Proutiere. Insensitivity in processor-sharing networks. *Performance Evaluation*, 49(1-4):193–210, 2002.

[36] T. Bonald and A. Proutiere. Insensitive bandwidth sharing in data networks. *Queueing Sys.*, 44:69–100, 2003.

[37] T. Bonald and A. Proutiere. On performance bounds for the integration of elastic and adaptive streaming traffic. In *Proc. of ACM Sigmetrics-Performance*, 2004.

[38] A. Bondi and J. Buzen. The response times of priority classes under preemptive resume in M/G/m queues. In *ACM Sigmetrics*, pages 195–201, August 1984.

[39] S. Borst, O. Boxma, J. Morrison, and R. Nunez-Queija. The equivalence between processor sharing and service in random order. *Oper. Res. Let.*, 31:254–262, 2003.

[40] S. Borst, O. Boxma, R. Nunez-Queija, and B. Zwart. The impact of the service discipline on delay asymptotics. *Performance Evaluation*, 54:175–206, 2003.

[41] S. Borst, R. Nunez-Queija, and B. Zwart. Sojourn time asymptotics in processor-sharing queues. *Queueing Sys.*, 53(1-2), 2006.

[42] D. Botvich and N. Duffield. Large deviations, economies of scale, and the shape of the loss curve in large multiplexers. *Queueing Systems*, 20:293–320, 1995.

[43] O. Boxma and V. Dumas. The busy period in the fluid queue. *Perf. Eval. Rev.*, 26:100–110, 1998.

[44] Bradford L. Barrett. The Webalizer. http://www.mrunix.net/webalizer, 2005.

[45] L. Bright and P. Taylor. Calculating the equilibrium distribution in level dependent quasi-birth-and-death processes. *Stochastic Models*, 11:497–514, 1995.

[46] D. Brooks, P. Bose, S. Schuster, H. Jacobson, P. N. Kudva, A. Buyuktosunoglu, J. Wellman, V. Zyuban, M. Gupta, and P. Cook. Power-aware microarchitecture: Design and modeling challenges for next-generation microprocessors. *IEEE Micro*, 20(4):26–44, 2000.

[47] P. Brown. Comparing FB and PS scheduling policies. *Perf. Eval. Rev.*, 34(3):18–20, 2006.

[48] J. Buzen and A. Bondi. The response times of priority classes under preemptive resume in M/M/m queues. *Operations Research*, 31:456–465, 1983.

[49] T. Cain, M. Martin, T. Heil, E. Weglarz, and T. Bezenek. Java TPC-W implementation. http://www.ece.wisc.edu/ pharm/tpcw.shtml, 2000.

[50] J. Calabrese. Optimal workload allocation in open queueing networks in multiserver queues. *Management Science*, 38:1792–1802, 1992.

[51] X. Chao and C. Scott. Several results on the design of queueing systems. *Operations Research*, 48:965–970, 2000.

[52] L. Cherkasova. Scheduling strategies to improve response time for web applications. In *High-performance computing and networking: international conference and exhibition*, pages 305–314, 1998.

[53] R. Chinchilla, J. Hoag, D. Koonce, H. Kruse, S. Ostermann, and Y. Wang. The trafgen traffic generator, 2002.

[54] V. Chistyakov. A theorem on sums of independent, positive random variables and its applications to branching processes. *Thry. of Prob. and its App.*, 9:640–648, 1964.

[55] J. Chover, P. Ney, and S. Waigner. Functions of probability measures. *J. d'Analyse Mathématique*, 26:255–302, 1973.

[56] D. Cline. Intermediate regular and π variation. *Proc. of the London Math. Soc.*, 68:529–557, 1994.

[57] D. Cline and G. Samorodnitsky. Subexponentiality of the product of two random variables. *Stoch. Proc. and their App.*, 49:75–98, 1994.

[58] A. Cobham. Priority assignment in waiting line problems. *Operations Research*, 2:70–76, 1954.

[59] A. Cockcroft. Watching your web server. http://www.theunixinsider.com, 1996.

[60] J. Cohen. Some results on regular variation for distributions in queueing and fluctuation theory. *J. Appl. Prob.*, 10:343–353, 1973.

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

[62] M. Crovella and A. Bestavros. Self-similarity in world wide web traffic: Evidence and possible causes. *Trans. on Networking*, 5(6):835–846, 1997.

[63] R. Davis. Waiting-time distribution of a multi-server, priority queueing system. *Operations Research*, 14:133–136, 1966.

[64] S. Delas, R. Mazumdar, and C. Rosenberg. Tail asymptotics for HOL priority queues handling a large number of independent stationary sources. *Queue. Sys. Thry. and App.*, 40(2):183–204, 2002.

[65] B. Dellart. How tolerable is delay? consumers evaluations of internet web sites after waiting. *J. of Interactive Marketing*, 13:41–54, 1999.

[66] A. Demers, S. Keshav, and S. Shenkar. Analysis and simulation of a fair queueing algorithm. *Journal of Internetworking*, 1:3–26, 1990.

[67] L. Dowdy and M. Chopra. On the applicability of using multiprogramming level distributions. In *Proc. of ACM Sigmetrics*, 1985.

[68] A. B. Downey. A parallel workload model and its implications for processor allocation. In *Proc. of High Performance Distributed Computing*, pages 112–123, August 1997.

[69] A. B. Downey. Evidence for long-tailed distributions in the internet. In *Proc. of ACM SIGCOMM Internet Measurment Workshop*, 2001.

[70] S. Drekic and J. E. Stafford. Symbolic computation of moments in priority queues. *J. on Computing*, 14:261–277, 2002.

[71] N. Duffield, W. Massey, and W. Whitt. A nonstationary offered-load model for packet networks. *Telecommunication Systems*, 13:271–296, 2001.

[72] R. Egorova, B. Zwart, and O. Boxma. Sojourn time tails in the M/D/1 processor sharing queue. *Probability in the engineering and informational sciences*, to appear, 2006.

[73] A. Feldmann, A. C. Gilbert, P. Huang, and W. Willinger. Dynamics of IP traffic: A study of the role of variability and the impact of control, 1999.

[74] H. Feng, V. Misra, and D. Rubenstein. PBS: a unified priority-based cpu scheduler. In *Proc. of ACM Sigmetrics*, 2007.

[75] W. Feng, M. Kawada, and K. Adachi. Analysis of a multiserver queue with two priority classes and (M,N)-threshold service schedule ii: preemptive priority. *Asia-Pacific Journal of Operations Research*, 18:101–124, 2001.

[76] L. Flatto. The waiting time distribution for the random order of service M/M/1 queue. *Ann. of App. Prob.*, 7:382–409, 1997.

[77] S. B. Fredj, T. Bonald, A. Proutiere, G. Regnie;, and J. W. Roberts. Statistical bandwidth sharing: a study of congestion at flow level. *SIGCOMM Comput. Commun. Rev.*, 31(4):111–122, 2001.

[78] E. Friedman and S. Henderson. Fairness and efficiency in web server protocols. In *Proc. of ACM Sigmetrics*, 2003.

[79] E. Friedman and G. Hurley. Protective scheduling. Technical report, Cornell University, 2003.

[80] J. Fulmer. Siege. http://joedog.org/siege.

[81] H. Gail, S. Hantler, and B. Taylor. Analysis of a non-preemptive priority multiserver queue. *Advances in Applied Probability*, 20:852–879, 1988.

[82] H. Gail, S. Hantler, and B. Taylor. On a preemptive Markovian queues with multiple servers and two priority classes. *Mathematics of Operations Research*, 17:365–391, 1992.

[83] J. Gehrke, S. Muthukrishnan, R. Rajaraman, and A. Shaheen. Scheduling to minimize average stretch online. In *40th Annual symposium on Foundation of Computer Science*, pages 433–443, 1999.

[84] S. Gigandet, A. Sudarsanam, and A. Aggarwal. The inktomi climate lab: an integrated environment for analyzing and simulating customer network traffic. In *Proc. ACM SIGCOMM Workshop on Int. Meas.*, pages 183–187, 2001.

[85] G.L.Choudhury and W. Whitt. Heavy-traffic asymptotic expansions for the asymptotic decay rates in the BMAP/G/1 queue. *Stochastic Models*, 10:453–498, 1994.

[86] M. Gong. *Quantifying Unfairness in Web Scheduling*. PhD thesis, University of Calgary, 2003.

[87] M. Gong and C. Williamson. Quantifying the properties of SRPT scheduling. In *IEEE/ACM Symposium on Mod., Anal., and Sim. of Comp. and Telecomm. Sys. (MASCOTS)*, 2003.

[88] M. Gong and C. Williamson. Simulation evaluation of hybrid SRPT scheduling policies. In *Proc of IEEE MASCOTS*, 2004.

[89] I. S. Gradshteyn and I. M. Ryzhik. *Tables of Integrals, Series, and Products*. Academic Press, 2000.

[90] F. Guillemin, P. Robert, and B. Zwart. Tail asymptotics for processor sharing queues. *Adv. in Appl. Prob.*, 36:525–543, 2004.

[91] M. Harchol-Balter. *Exploiting process lifetime distributions for dynamic load balancing*. PhD thesis, University of California, Berkeley, 1996.

[92] M. Harchol-Balter. Task assignment with unknown duration. *Journal of the ACM*, 49(2), 2002.

[93] M. Harchol-Balter, M. Crovella, and C. Murta. On choosing a task assignment policy for a distributed server system. *IEEE Journal of Parallel and Distributed Computing*, 59:204 – 228, 1999.

[94] M. Harchol-Balter, C. Li, T. Osogami, A. Scheller-Wolf, and M. Squillante. Task assignment with cycle stealing under central queue. In *International Conference on Distributed Computing Systems*, pages 628–637, 2003.

[95] M. Harchol-Balter, T. Osogami, A. Scheller-Wolf, and A. Wierman. Multiserver queueing systems with multiple priority classes. *Queueing Sys.*, 51:331–360, 2005.

[96] M. Harchol-Balter, B. Schroeder, M. Agrawal, and N. Bansal. Size-based scheduling to improve web performance. *ACM Transactions on Computer Systems*, 21(2), May 2003.

[97] M. Harchol-Balter, K. Sigman, and A. Wierman. Asymptotic convergence of scheduling policies with respect to slowdown. *Performance Evaluation*, 49(1-4):241–256, 2002.

[98] M. Harchol-Balter, K. Sigman, and A. Wierman. Asymptotic convergence of scheduling policies with respect to slowdown. In *Proc. of* Performance '02, 2002.

[99] M. Harchol-Balter, K. Sigman, and A. Wierman. Understanding the slowdown of large jobs in an M/GI/1 system. *Performance Evaluation Review*, 30(3):9–11, 2002.

[100] G. Hasegawa, T. Matsuo, M. Murata, and H. Miyahara. Comparisons of packet scheduling algorithms for fair service among connections on the internet. In *Proc. of IEEE INFOCOMM 2000*, 2000.

[101] P. E. Heegaard. Gensyn - generator of synthetic internet traffic. http://www.item.ntnu.no/ poulh/GenSyn/gensyn.html.

[102] M. Hu, J. Zhang, and J. Sadowsky. A size-aided opportunistic scheduling scheme in wireless networks. In *Globecom*, 2003.

[103] ITA. The Internet traffic archives. Available at `http://town.hall.org/Archives/pub/ITA/`, 2002.

[104] R. Jain. *The Art of Computer Systems Performance Analysis*. Wiley & Sons, 1991.

[105] P. Jelenkovic and P. Momcilovic. Resource sharing with subexponential distributions. In *Proc. of Infocom*, 2002.

[106] K. Kant, V. Tewari, and R. Iyer. Geist: Generator of e-commerce and internet server traffic. http://kkant.ccwebhost.com/geist/.

[107] E. Kao and K. Narayanan. Computing steady-state probabilities of a nonpreemptive priority multiserver queue. *Journal on Computing*, 2:211–218, 1990.

[108] E. Kao and K. Narayanan. Modeling a multiprocessor system with preemptive priorities. *Management Science*, 2:185–97, 1991.

[109] E. Kao and S. Wilson. Analysis of nonpreemptive priority queues with multiple servers and two priority classes. *European Journal of Operational Research*, 118:181–193, 1999.

[110] A. Kapadia, M. Kazumi, and A. Mitchell. Analysis of a finite capacity nonpreemptive priority queue. *Computers and Operations Research*, 11:337–343, 1984.

[111] O. Kella and U. Yechiali. Waiting times in the non-preemptive priority m/m/c queue. *Stochastic Models*, 1:257 – 262, 1985.

[112] O. Kella, B. Zwart, and O. Boxma. Some time-dependent properties of symmetric M/G/1 queues. *J. Appl. Prob.*, 42:223–234, 2005.

[113] F. Kelly. *Reversibility and Stochastic Networks*. John Wiley & Sons, 1979.

[114] F. Kelly, A. Maulloo, and D. Tan. Rate control for communication networks: shadow prices, proportional fairness, and stability. *J. of the Op. Res. Soc.*, 49, 1998.

[115] D. Kendall. Stochastic processes occurring in the theory of queues and their analysisi by the method of the imbedded Markov chain. *Annals of Math. Stat.*, 24:338–354, 1953.

[116] M. Kendall. *The Advanced Theory of Statistics*. Griffin, London, 1945.

[117] A. Kherani. Sojourn times in (discrete) time shared systems and their continuous time limits. In *Proc. of ValueTools*, 2006.

[118] A. A. Kherani and R. Nunez-Queija. TCP as an implementation of age-based scheduling: fairness and performancs. In *IEEE Infocom*, 2006.

[119] L. Kleinrock. *Queueing Systems*, volume I. Theory. John Wiley & Sons, 1975.

[120] L. Kleinrock. *Queueing Systems*, volume II. Computer Applications. John Wiley & Sons, 1976.

[121] L. Kleinrock, R. R. Muntz, and J. Hsu. Tight bounds on average response time for processor-sharing models of time-shared computer systems. *Info. Processing*, 71:50–58, 1971.

[122] D. Korshunov. On distribution tail of the maximum of a random walk. *Stoch. Proc. Appl.*, 72:97–103, 1997.

[123] C. Kotopoulos, N. Likhanov, and R. Mazumdar. Overflow asymptotics in GPS systems with heterogeneous longtailed inputs. In *Proc. of IEEE Infocom*, 2001.

[124] V. Kumar, J. Kapur, and O. Hawaleshka. On the interrelationship between semi-open and closed queueing network models for flexible manufacturing system. *J. of. Information and Optimization Sciences*, 8:167–187, 1987.

[125] G. Latouche and V. Ramaswami. *Introduction to Matrix Analytic Methods in Stochastic Modeling*. ASA-SIAM, 1999.

[126] H. Leemans. *The Two-Class Two-Server Queue with Nonpreemptive Heterogeneous Priority Structures*. PhD thesis, K.U.Leuven, 1998.

[127] W. Leland, M. Taqqu, W. Willinger, and D. Wilson. On the self-similar nature of ethernet traffic. In *Proc. of SIGCOMM '93*, pages 183–193, September 1993.

[128] J. Leslie. On the non-closure under convolution of the class of subexponential distributions. *J. of App. Prob.*, 26:58–66, 1989.

[129] Z. Liu, N. Niclausse, and C. Jalpa-Villanueva. Traffic model and performance evaluation of web servers. *Performance Evaluation*, 46(2-3):77–100, 2001.

[130] D. Lu, P. Dinda, Y. Qiao, and H. Sheng. Effects and implications of file size/service time correlation on web server scheduling policies. In *Proc. of IEEE Mascots*, 2005.

[131] D. Lu, H. Sheng, and P. Dinda. Size-based scheduling policies with inaccurate scheduling information. In *Proc. of IEEE Mascots*, 2004.

[132] B. A. Mah, P. E. Sholander, L. Martinez, and L. Tolendino. Ipb: An internet protocol benchmark using simulated traffic, 1998.

[133] A. Mandelbaum and M. Reiman. On pooling in queueing networks. *Management Science*, 44:971–981, 1998.

[134] M. Mandjes and M. Nuyens. Sojourn times in the M/G/1 FB queue with light-tailed service times. *Prob. in the Eng. and Info. Sci.*, 19:351–361, 2005.

[135] M. Mandjes and A. Zwart. Large deviations of sojourn times in processor sharing queues. *Under submission*, 2005.

[136] R. Mangharam, M. Demirhan, R. Rajkumar, and D. Raychaudhuri. Size matters: Size-based scheduling for MPEG-4 over wireless channels. In *SPIE & ACM Proceedings in Multimedia Computing and Networking*, pages 110–122, 2004.

[137] T. Matis and R. Feldman. Using cumulant functions in queueing theory. *Queueing Sys.*, 40:341–353, 2002.

[138] D. McWherter, B. Schroeder, N. Ailamaki, and M. Harchol-Balter. Priority mechanisms for OLTP and transactional web applications. In *Int. Conf on Data Engineering*, 2004.

[139] D. McWherter, B. Schroeder, N. Ailamaki, and M. Harchol-Balter. Improving preemptive prioritization via statistical characterization of OLTP locking. In *Int. Conf on Data Engineering*, 2005.

[140] D. Menasce and V. Almeida. *Scaling for E-Buisness: technologies, models, performance, and capacity planning*. Prentice Hall, 2000.

[141] A. D. Meyer and J. Teugels. On the asymptotic behaviour of the distributions of the busy period and the service time in M/G/1. *J. App. Prob.*, 17:802–813, 1980.

[142] Microsoft. The arts and science of web server tuning and internet information servers 5.0. microsoft technet - insights and answers for it professionals. http://www.microsoft.com/technet/, 2001.

[143] Microsoft IIS 6.0 Resource Kit Tools. Microsoft web capacity analysis tool (wcat) version 5.2.

[144] Microsoft TechNet. Ms web application stress tool (wast). http://www.microsoft.com/technet/itsolutions/intranet/ downloads/webstres.mspx.

[145] D. Miller. Steady-state algorithmic analysis of M/M/c two-priority queues with heterogeneous servers. In R. L. Disney and T. J. Ott, editors, *Applied probability - Computer science, The Interface, volume II*, pages 207–222. Birkhauser, 1992.

[146] Mindcraft. The authmark benchmark. http://www.mindcraft.com/authmark/.

[147] I. Mitrani. *Probabilistic Modeling*. Cambridge University Press, 1998.

[148] I. Mitrani and P. King. Multiprocessor systems with preemptive priorities. *Performance Evaluation*, 1:118–125, 1981.

[149] P. Molinero-Fernandez, K. Psounis, and B. Prabhakar. Systems with multiple servers under heavy-tailed workloads, 2003 – Manuscript.

[150] P. Morse. *Queues, Inventories, and Maintenance*. John Wiley and Sons, 1958.

[151] D. Mosberger and T. Jin. httperf: A tool for measuring web server performance, 1998.

[152] T. Mudge. Power: A first-class architectural design constraint. *Computer*, 34(4):52–58, 2001.

[153] E. Nahum, M. Rosu, S. Seshan, and J. Almeida. The effects of wide-area conditions on WWW server performance. In *Proc of* ACM SIGMETRICS, pages 257–267, 2001.

[154] R. Nelson. *Probability, Stochastic Processes, and Queueing Theory*. Springer-Verlag, 1995.

[155] M. Neuts. Moment formulas for the Markov renewal branching process. *Advances in Applied Probabilities*, 8:690–711, 1978.

[156] B. Ngo and H. Lee. Analysis of a pre-emptive priority M/M/c model with two types of customers and restriction. *Electronics Letters*, 26:1190–1192, 1990.

[157] T. Nishida. Approximate analysis for heterogeneous multiprocessor systems with priority jobs. *Performance Evaluation*, 15:77–88, 1992.

[158] R. Nunez-Queija. Queues with equally heavy sojourn time and service requirement distributions. *Ann. Oper. Res*, 113:101–117, 2002.

[159] M. Nuyens. *The Foreground-Background Queue*. PhD thesis, University of Amsterdam, 2004.

[160] M. Nuyens and A. Wierman. The foreground-background queue: A survey. *Performance evaluation*, in press.

[161] M. Nuyens, A. Wierman, and B. Zwart. Preventing large sojourn times using SMART scheduling. *Operations Research*, in press.

[162] M. Nuyens and B. Zwart. A large-deviations analysis of the GI/GI/1 SRPT queue. *Under submission*, 2005.

[163] T. Osogami. *Analaysis of multi-server systems via dimensionality reduction of Markov Chains*. PhD thesis, Carnegie Mellon University, 2005.

[164] T. Osogami and M. Harchol-Balter. A closed-form solution for mapping general distributions to minimal PH distributions. In *Modelling Tools and Techniques for Comp. and Comm. System Perf. Eval.*, 2003.

[165] T. Osogami, M. Harchol-Balter, and A. Scheller-Wolf. Analysis of cycle stealing with switching cost. In *ACM Sigmetrics 2003*, pages 184–195, 2003.

[166] T. Osogami, A. Wierman, M. Harchol-Balter, and A. Scheller-Wolf. A recursive analysis technique for multi-dimensionally infinite Markov chains. *Perf. Eval. Rev.*, 32(2):12–13, 2004.

[167] T. J. Ott. The sojourn-time distribution in the M/G/1 queue with processor sharing. *J. of App. Prob.*, 21:360–378, 1984.

[168] V. S. Pai, P. Druschel, and W. Zwaenepoel. Flash: An efficient and portable web server. In *Proc. of USENIX*, 1999.

[169] A. Pakes. On the tails of waiting-time distributions. *J. App. Prob.*, 12:555–564, 1975.

[170] Z. Palmowski and T. Rolski. On busy period asymptotics in the GI/GI/1 queue. *Under submission*, 2004.

[171] A. Parekh and R. Gallager. A generalized processor sharing approach to flow control in integrated services networks: the single node case. *IEEE/ACM Transactions on Networking*, 1:344–357, 1993.

[172] K. Park and W. Willinger. *Self-similar network traffic and performance evaluation*. John Wiley & Sons, 2000.

[173] V. Paxson and S. Floyd. Wide area traffic: The failure of Poisson modeling. *Trans. on Networking*, pages 226–244, 1995.

[174] D. L. Peterson. Data center I/O patterns and power laws. In *CMG Proc.*, December 1996.

[175] T. E. Phipps. Machine repair as a priority waiting-line problem. *Oper. Res.*, 4:76–85, 1956.

[176] M. Pinedo. *Scheduling: Theory, algorithms, and systems*. Prentice-Hall, Inc., 2002.

[177] PostgreSQL. `http://www.postgresql.org`.

[178] Y. Qiao, D. Lu, R. Bustamante, and P. Dinda. Looking at the server side of peer-to-peer systems. Technical Report NWU-CS-04-37, Northwestern University, 2004.

[179] I. A. Rai, G. Urvoy-Keller, and E. Biersack. Analysis of LAS scheduling for job size distributions with high variance. In *Proc. of ACM Sigmetrics*, 2003.

[180] I. A. Rai, G. Urvoy-Keller, M. Vernon, and E. W. Biersack. Performance modeling of LAS based scheduling in packet switched networks. In *Proc. of ACM Sigmetrics-Performance*, 2004.

[181] K. Ramanan and A. Stolyar. Largest weighted delay first scheduling: large deviations and optimality. *Ann. of App. Prob.*, 11:1–48, 2001.

[182] M. Rawat and A. Kshemkalyani. SWIFT: Scheduling in web servers for fast response time. In *Symp. on Net. Comp. and App.*, 2003.

[183] J. Rawls. *A theory of social justice*. Harvard University Press, 1971.

[184] D. Raz, B. Avi-Itzhak, and H. Levy. Fairness considerations in multi-server and multi-queue systems. In *Proc. of Valuetools*, 2006.

[185] D. Raz, H. Levy, and B. Avi-Itzhak. A resource-allocation queueing fairness measure. In *Proc. of ACM Sigmetrics-Performance*, 2004.

[186] S. Resnik and G. Samorodnitsky. Activity periods of an infinite server queue and performance of certain heavy tailed fluid queues. *Queueing Sys.*, 33:43–71.

[187] R. Rhodes, M. P. Battin, and A. Silvers. *Medicine and social justice*. Oxford University Press, 2002.

[188] R. Righter and J. Shanthikumar. Scheduling multiclass single server queueing systems to stochastically maximize the number of succesful departures. *Prob. in the Eng. and Info. Sci.*, 3:967–978, 1989.

[189] R. Righter, J. Shanthikumar, and G. Yamazaki. On external service disciplines in single stage queueing systems. *J. of Applied Probability*, 27:409–416, 1990.

[190] L. Rizzo. Dummynet: a simple approach to the evaluation of network protocols. *ACM Computer Communication Review*, 27(1), 1997.

[191] J. W. Roberts. A survey on statistical bandwidth sharing. *Comput. Networks*, 45(3):319–332, 2004.

[192] S. Ross. *Stochastic Processes*. John Wiley & Sons, 1996.

[193] S. Ross. *Introduction to Probability Models*. Academic Press, 1997.

[194] A. Rousskov and D. Wessels. High performance benchmarking with web polygraph. *Software - Practice and Experience*, 1:1–10, 2003.

[195] W. Sandmann. A discrimination frequency based queueing fairness measure with regard to job seniority and service requirement. In *Proc. of Euro NGI Conf. on Next Generation Int. Nets*, 2005.

[196] W. Sandmann. Analysis of a queueing fairness measure. In *GI/ITG Conf. on Measurement, Modeling, and Eval. of Comp. and Comm. Sys.*, 2006.

[197] R. Schassberger. A new approach to the M/G/1 processor sharing queue. *Adv. in Appl. Prob.*, 16:802–813, 1984.

[198] P. Schatte. On conditional busy periods in n/M/GI/1 queues. *Math. Operationsforsh. u. Statist. ser. Optimization*, 14:455–465, 1983.

[199] P. Schatte. The M/GI/1 queue as limit of closed queueing systems. *Math. Operationsforsh. u. Statist. ser. Optimization*, 15:161–165, 1984.

[200] A. Scheller-Wolf. Necessary and sufficient conditions for delay moments in FIFO multiserver queues with an application comparing s slow servers with one fast one. *Operations Research*, 51:748–758, 2003.

[201] L. E. Schrage. A proof of the optimality of the shortest remaining processing time discipline. *Operations Research*, 16:678–690, 1968.

[202] L. E. Schrage and L. W. Miller. The queue M/G/1 with the shortest remaining processing time discipline. *Operations Research*, 14:670–684, 1966.

[203] B. Schroeder and M. Harchol-Balter. Web servers under overload: How scheduling can help. In *International Teletraffic Congress (ITC 2003)*, 2003.

[204] B. Schroeder, M. Harchol-Balter, A. Iyengar, E. Nahum, and A. Wierman. How to determine a good multi-programming level for external scheduling. In *Proc. of IEEE ICDE*, 2006.

[205] B. Schroeder, M. Harchol-Balter, A. Iyengar, E. Nahum, and A. Wierman. Providing QoS using external scheduling. 2007.

[206] B. Schroeder, A. Wierman, and M. Harchol-Balter. Open versus closed: A cautionary tale. In *Proc. of NSDI*, 2006.

[207] G. Shahkar and H. Tareghian. Designing a production line through optimization of m/g/c using simulation. *Mathematical Engineering in Industry*, 8:123–126, 2001.

[208] A. Shaikh, J. Rexford, and K. Shin. Load-sensitive routing of long-lived IP flows. In *Proc. of ACM SIGCOMM*, pages 215–226, 1999.

[209] S. Shakkottai and R. Srikant. Many-sources delay asymptotics with applications to priority queues. *Queueing Systems: Theory and Applications*, 39:183–200, October 2001.

[210] A. Silberschatz and P. Galvin. *Operating System Concepts, 5th Edition*. John Wiley & Sons, 1998.

[211] A. Sleptchenko. Multi-class, multi-server queues with non-preemptive priorities. Technical Report 2003-016, EURANDOM, Eindhoven University of Technology, 2003.

[212] A. Sleptchenko, A. van Harten, and M. van der Heijden. An exact solution for the state probabilities of the multi-class, multi-server queue with preemptive priorities, 2003 – Manuscript.

[213] sourceforge.net. Deluge - a web site stress test tool. http://deluge.sourceforge.net/.

[214] sourceforge.net. Hammerhead 2 - web testing tool. http://hammerhead.sourceforge.net/.

[215] W. Stallings. *Operating Systems, 2nd Edition*. Prentice Hall, 1995.

[216] Standard Performance Evaluation Corporation (SPEC). SFS97_R1 (3.0) benchmark. `http://www.specbench.org/osg/web99/`.

[217] Standard Performance Evaluation Corporation (SPEC). SPECJ2EE benchmark. `http://www.specbench.org/osg/web99/`.

[218] Standard Performance Evaluation Corporation (SPEC). SPECmail2001 benchmark. `http://www.specbench.org/osg/web99/`.

[219] Standard Performance Evaluation Corporation (SPEC). SPECweb99 benchmark. `http://www.specbench.org/osg/web99/`.

[220] S. Stidham. On the optimality of single-server queueing systems. *Operations Research*, 18:708–732, 1970.

[221] L. Takács. A single-server queue with poisson input. *Oper. Res.*, 10:388–397, 1962.

[222] H. Takagi. *Queueing Analysis: Volume 1: Vacation and Priority Systems*. North-Holland, 1991.

[223] A. Tanenbaum. *Modern Operating Systems*. Prentice Hall, 1992.

[224] W. Tang, Y. Fu, L. Cherkasova, and A. Vahdat. Medisyn: A synthetic streaming media service workload generator. In *Proc. of 13th NOSSDAV*, 2003.

[225] The Apache softward foundation. The Apache web server.

[226] V. Tiwari, D. Singh, S. Rajgopal, G. Mehta, R. Patel, and F. Baez. Reducing power in high-performance microprocessors. In *Proc. of Design Automation Conference*, pages 732–737, 1998.

[227] Transaction Processing Performance Council. TPC benchmark C. Number Revision 5.1.0, December 2002.

[228] Transaction Processing Performance Council. TPC benchmark W (web commerce). Number Revision 1.8, February 2002.

[229] G. Trent and M. Sake. WebStone: the first generation in HTTP server benchmarking. Technical report, MTS Silicon Graphics, February 1995.

[230] VeriTest. NetBench 7.0.3. http://www.etestinglabs.com/benchmarks/netbench/.

[231] VeriTest. WebBench 5.0. http://www.etestinglabs.com/benchmarks/webbench/.

[232] L. von Ahn and L. Dabbish. Labeling images with a computer game. In *CHI '04: Proc. of the SIGCHI conference on Human factors in computing systems*, pages 319–326, 2004.

[233] A. Ward and W. Whitt. Predicting response times in processor-sharing queues. In *Proc. of the Fields Institute Conf. on Comm. Networks*, 2000.

[234] W. Whitt. A review of $L = \lambda W$ and extensions. *Queueing Systems*, 9:235–268, 1991.

[235] A. Wierman. On the effect of inexact size information in size based policies. *Performance Evaluation Review*, 34(3):21–23, 2006.

[236] A. Wierman. Fairness and classifications. *Performance Evaluation Review*, page in press, 2007.

[237] A. Wierman, N. Bansal, and M. Harchol-Balter. A note comparing response times in the M/GI/1/FB and M/GI/1/PS queues. *Operations Research Letters*, 32:73–76, 2003.

[238] A. Wierman and M. Harchol-Balter. Classifying scheduling policies with respect to unfairness in an M/GI/1. In *Proc. of ACM Sigmetrics*, 2003.

[239] A. Wierman and M. Harchol-Balter. Formalizing SMART scheduling. In *Workshop on MAthematical performance Modeling and Analysis (MAMA)*, 2004.

[240] A. Wierman and M. Harchol-Balter. Classifying scheduling policies with respect to higher moments of response time. In *Proc. of ACM Sigmetrics*, 2005.

[241] A. Wierman, M. Harchol-Balter, and T. Osogami. Nearly insensitive bounds on SMART scheduling. In *Proc. of ACM Sigmetrics*, 2005.

[242] A. Wierman, T. Osogami, M. Harchol-Balter, and A. Scheller-Wolf. How many servers are best in a dual-priority M/PH/k system. *Performance Evaluation*, 63(12):1253–1272, 2006.

[243] A. Wierman, T. Osogami, and J. Olsen. Modeling TCP-Vegas under on/off traffic. *Perf. Eval. Rev.*, 31(2):6–8, 2003.

[244] A. Wierman, T. Osogami, and J. Olsen. A unified framework for modeling TCP-Vegas, TCP-SACK, and TCP-Reno. In *Proceedings of MASCOTS*, 2003.

[245] A. Wierman, J. Salzman, M. Jablonski, and A. Godbole. An upper bound for the pebbling threshold of the path. *Discrete Mathematics*, 275:367–373, 2004.

[246] A. Wierman, E. Winands, and O. Boxma. Scheduling in polling systems. 2007.

[247] R. W. Wolff. *Stochastic Modeling and the Theory of Queues*. Prentice Hall, 1989.

[248] C. Yang, A. Wierman, S. Shakkottai, and M. Harchol-Balter. Tail asymptotics for policies favoring short jobs in a many-flows regime. In *Proc. of ACM Sigmetrics*, 2006.

[249] C. W. Yang and S. Shakkottai. Delay asymptote of the SRPT scheduler. In *Proc. of the IEEE Conference on Decision and Control*, December 2004.

[250] S. Yang and G. de Veciana. Enhancing both network and user performance for networks supporting best effort traffic. *Trans. on Networking*, 12(2):349–360, 2004.

[251] S. Yang and G. de Vecianca. Bandwidth sharing: the role of user impatience. In *Proc of Globecom*, 2001.

[252] S. Yashkov. Processor sharing queues: Some progress in analysis. *Queueing Sys.*, 2:1–17, 1987.

[253] S. Yashkov. Mathematical problems in the theory of shared-processor systems. *J. of Soviet Mathematics*, 58:101–147, 1992.

[254] M. Yuksel, B. Sikdar, K. S. Vastola, and B. Szymanski. Workload generation for ns simulations of wide area networks and the internet, 2000.

[255] M. Zhou and L. Zhou. How does waiting duration information influence customers' reactions to waiting for services. *J. of Applied Social Psychology*, 26:1702–1717, 1996.

[256] A. Zwart and O. Boxma. Sojourn time asymptotics in the M/G/1 processor sharing queue. *Queueing Sys.*, 35:141–166, 2000.

[257] B. Zwart. *Queueing systems with heavy tails*. PhD thesis, Technische Universiteit Eindhoven, 2001.

Afterward

I joined the Computer Science Department Ph.D. program at Carnegie Mellon University in the fall of 2001 and almost immediately began working with my advisor Mor Harchol-Balter.

The first piece of this thesis was actually the result of work from the winter of my first year. Together with Mor and Karl Sigman, we proved that all common preemptive policies are "fair" to large job sizes and, in fact, large job sizes are treated equivalently under most common scheduling policies (see Section 7.2). This work appeared as a short abstract at the MAMA workshop in the summer of 2002 [99] and then as a full paper at the IFIP Performance conference in 2002 [98], where I gave my first conference talk. This paper served as the starting point for an extended look at the fairness of scheduling policies that has lasted the rest of my graduate career resulting in papers with Mor at Sigmetrics both in 2003 [238] (see Section 7.1), which won the best paper award, and in 2005 [240] (see Section 7.3 and 7.4). Recently, I was invited to put together a survey of these, and other, recent papers studying fairness [236].

In addition to studying the fairness of scheduling policies, the paper at Sigmetrics in 2003 also began another branch of this thesis: the study of classifications. In the Sigmetrics 2003 paper we introduced the technique-based classifications from Chapter 5. Building from there, I developed and analyzed the SMART classification with Mor and Takayuki Osogami (see Section 4.1). The SMART classification first appeared as an extended abstract at the MAMA workshop in the summer of 2004 [239] and then as a full paper at Sigmetrics 2005 [241]. In the following year, I generalized the SMART classification to obtain the SMART$_\epsilon$ classification (see Section 4.2). This work has so far only appeared in the MAMA workshop as an extended abstract [235], but a full version of the paper is under preparation.

The SMART classification has spurred collaborations with a number of fellow researchers. In particular, it served as the beginning of my collaboration with the researchers at the EURANDOM institute in the Netherlands. With Bert Zwart and Misja Nuyens, we were able to derive a number of results about the response time distribution under SMART policies (see Section 6.3). Our collaboration also extended to a study of the FB policy. Many of these results are still forthcoming, but one paper has already been accepted to Operations Research [161] and another has been accepted to Performance Evaluation [160]. In addition to collaborations with Misja and Bert, the SMART classification also led to a collaboration with Chang-Woo Yang and Sanjay Shakkottai where we analyzed the distribution of response time in the many sources regime. This work appeared in Sigmetrics 2006 [248].

In parallel with work on the SMART classification, I was working with Mor, Takayuki, and Alan Scheller-Wolf to develop techniques for analyzing priority scheduling in multiserver systems (Chapter 9). This collaboration was extremely fruitful and resulted in a paper at the MAMA workshop [166] and two very interesting journal papers that have appeared in QUESTA [95] and Performance Evaluation [242].

Finally, over the last year I have worked with Mor and Bianca Schroeder to characterize the impact of interactive users on the effectiveness of scheduling (Chapter 8). This is an interesting case of "theory meets practice," and it resulted in the most applied piece of the thesis – a publication at NSDI 2006 [206].

Though many of the pieces of the thesis have appeared already as workshop, conference, or journal publications, a number of sections in the thesis include results that have not appeared outside of the thesis. In particular, the FOOLISH classification is new to this thesis (Section 4.3). In addition, the results characterizing the tail behavior of response time under PROTECTIVE policies are new to this thesis (Sections 6.3 and 7.2). Further, the notion of "politeness" introduced in Section 7.5 and the analysis of this quantity are new to this thesis.

Apart from my thesis work, I also had the opportunity to take part in a number of other research projects during my graduate career. While the majority of my graduate studies were devoted to analytic work, I also spent some time on performance modeling for specific application domains, e.g. studying TCP dynamics [244, 243] and database scheduling [204, 205]. Also, there were a number of other more theoretical side projects that I worked on, including extending work from my undergrad years on graph pebbling [245] and comparing FB and PS [27, 237]. In addition, over the last year I have spent time visiting the EURANDOM institute and become in involved with a number of projects there. One project with Onno Boxma and Erik Winands has already led to a paper studying the effect of scheduling in polling systems [246], and a other collaborations with Ivo Adan, Marcel van Vuuren, Pascal Etman, and Ad Kock are also proving to be quite interesting.

About the Author

Adam Wierman received a BS with University Honors in Computer Science and Mathematics with minors in Psychology and Statistics from Carnegie Mellon University in 2001. He then continued at Carnegie Mellon University to receive his Masters degree in 2004 and his Ph.D. in 2007. For a brief period in 2004 and an extended period in 2006, he served as a visiting researcher at the EURANDOM institute in the Netherlands. He is a recipient of an NSF Graduate Research Fellowship, a Siebel Scholars Fellowship, and a best student paper award at the ACM Sigmetrics conference. In addition, he has received the Carnegie Mellon University Graduate Student Teaching Award and the Alan J. Perlis Student Teaching Award.

His research applies analytic models and tools that are traditionally used in the operations research community, in particular stochastic modeling and queueing theory, in order to evaluate the impact of design decisions for computer systems. His main focus is on understanding the impact of scheduling heuristics on response time and fairness measures.