Thesis Proposal

Near-Optimal Scheduling: Towards a Unified Theory

Ziv Scully

Carnegie Mellon University
Computer Science Department

November 19, 2020

Thesis Committee

Mor Harchol-Balter, CMU (co-chair)
Guy E. Blelloch, CMU (co-chair)
Alan Scheller-Wolf, CMU
Anupam Gupta, CMU
Adam Wierman, Caltech
Balaji Prabhakar, Stanford

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

Contents

1 Motivation and Overview

Computer systems are rife with jobs waiting in queues: HTTP requests waiting for responses in web servers, packet flows waiting for transmission in network switches, SQL queries waiting for results in databases, and more. Every queue has an accompanying *scheduling policy* which decides the order in which to serve jobs. Choosing the right scheduling policy can significantly improve many aspects of a system's performance, such as its response time.[1] As such, the question of what scheduling policy to use is very important, having been studied extensive from both theoretical [15, 20, 26, 33, 46] and practical [4, 5, 10, 49] perspectives.

In some simple settings, theorists have found optimal scheduling policies. For example, suppose our goal is to minimize mean response time of a preemptive M/G/1 queue, which is a classic single-server queueing model. When the scheduler knows each job's exact size,[2] the *Shortest Remaining Processing Time* (SRPT) policy, which always serves the job of least remaining size, minimizes mean response time [35]. Under much more general conditions, such as when job sizes are unknown or only partially known, a generalization of SRPT called the *Gittins* policy is optimal [2, 15].[3]

While proving Gittins's optimality in the idealized setting described above was a major accomplishment of queueing theory, it is by no means the end of the optimal scheduling story. In practice, optimal scheduling requires addressing many questions that fall outside the realm of the idealized setting.

- The M/G/1 is a single-server model, but multiserver computer systems are the norm. This is true at all scales, from CPUs with multiple cores to datacenters with multiple machines. How do we optimally schedule in multiserver systems?
- Gittins minimizes mean response time, but other objectives can be more important in practice. It may be okay to increase mean response time if one can decrease the probability of especially large response times? How do we optimally schedule for other objectives?
- The idealized M/G/1 model assumes that there is no downside to using advanced scheduling policies like Gittins, but this ignores a number of implementation details that are important in practice.
  - Implementing Gittins requires solving an optimization problem which can be complicated or even intractable to solve in general.
  - Gittins's optimality assumes that switching between jobs is instant, but switching may take nonnegligible time.
  - Gittins works by assigning jobs a priority that can be any positive real number, but some computer systems, such as network switches, are limited to a finite number of priority levels.

[1] A job's *response time*, also known as its *sojourn time* or *latency*, is the time between its arrival and its completion.

[2] A job's *size*, also known as its *service requirement*, is the total amount of service it requires to complete.

[3] The relationship between SRPT and Gittins is worth clarifying. Gittins is defined in a general way that depends on the information the scheduler knows about each job's size. In the special case where the scheduler has perfect job size information, Gittins is equivalent to SRPT.

How do we model these and other practical concerns, and how do we optimally schedule in light of them?

These practical scheduling questions are not new, but they have long resisted theorists' efforts, so solving them exactly is likely intractable. As such, system designers turn to heuristic scheduling policies. In the realm of heuristics, theory has several roles to play: inventing new heuristics; evaluating heuristics' performance, which enables choosing between heuristics or tuning a heuristics parameters; and, when possible, proving that heuristics achieve near-optimal performance. However, at least two major obstacles prevent theory from playing this role.

- Analyzing the response time of heuristic scheduling policies is, for the most part, an open problem. The state of the art is ad-hoc analyses of a small set of simple policies. To be able to evaluate and tune policies in the wide space of possible heuristics, we need more general analysis techniques.
- Proving that a heuristic has near-optimal performance requires bounding the optimal performance, but this is an open problem even in the simplest case. For example, although we know that Gittins minimizes mean response time in an idealized setting, we do not know what mean response time it actually achieves. To be able to evaluate whether heuristics achieve near-optimal mean response time, we need a better understanding of Gittins.

1.1 Contributions

I propose to solve the theoretical obstacles outlined above and apply the resulting solutions to answer practical scheduling questions. The two main theoretical themes are as follows:

- I will develop a unified theory of M/G/1 scheduling, called *SOAP*, that is powerful enough to analyze a vast space of heuristic scheduling policies (Chapter 2).
- I will present an in-depth study of Gittins, including multiple ways of understanding and computing its mean response time (Chapter 3).

Using these insights, I will present a number of novel results that address practical scheduling questions, including the following:

- I will show that Gittins has near-optimal mean response time in the M/G/k, the multiserver analogue of the M/G/1 (Section 3.3).
  - Specifically, I will show that the difference between Gittins's M/G/k and M/G/1 mean response times grows at most logarithmically with load. This implies optimality in heavy traffic for most finite-variance job size distributions.
- I will show that for heavy-tailed job size distributions, Gittins is also asymptotically optimal for minimizing the probability of especially large response times in the M/G/1 (Section 2.2.3).
- I will present a new policy, called *M-SERPT*, which is simpler to implement than Gittins but still has near-optimal mean response time when jobs have unknown sizes (Section 2.2.2).
  - Specifically, I will show that M-SERPT's mean response time is at most a constant factor worse than Gittins's.
- I will use the SOAP response time analysis to evaluate and tune heuristics for practical concerns like preemption costs and limited priority levels (Section 2.2.1).

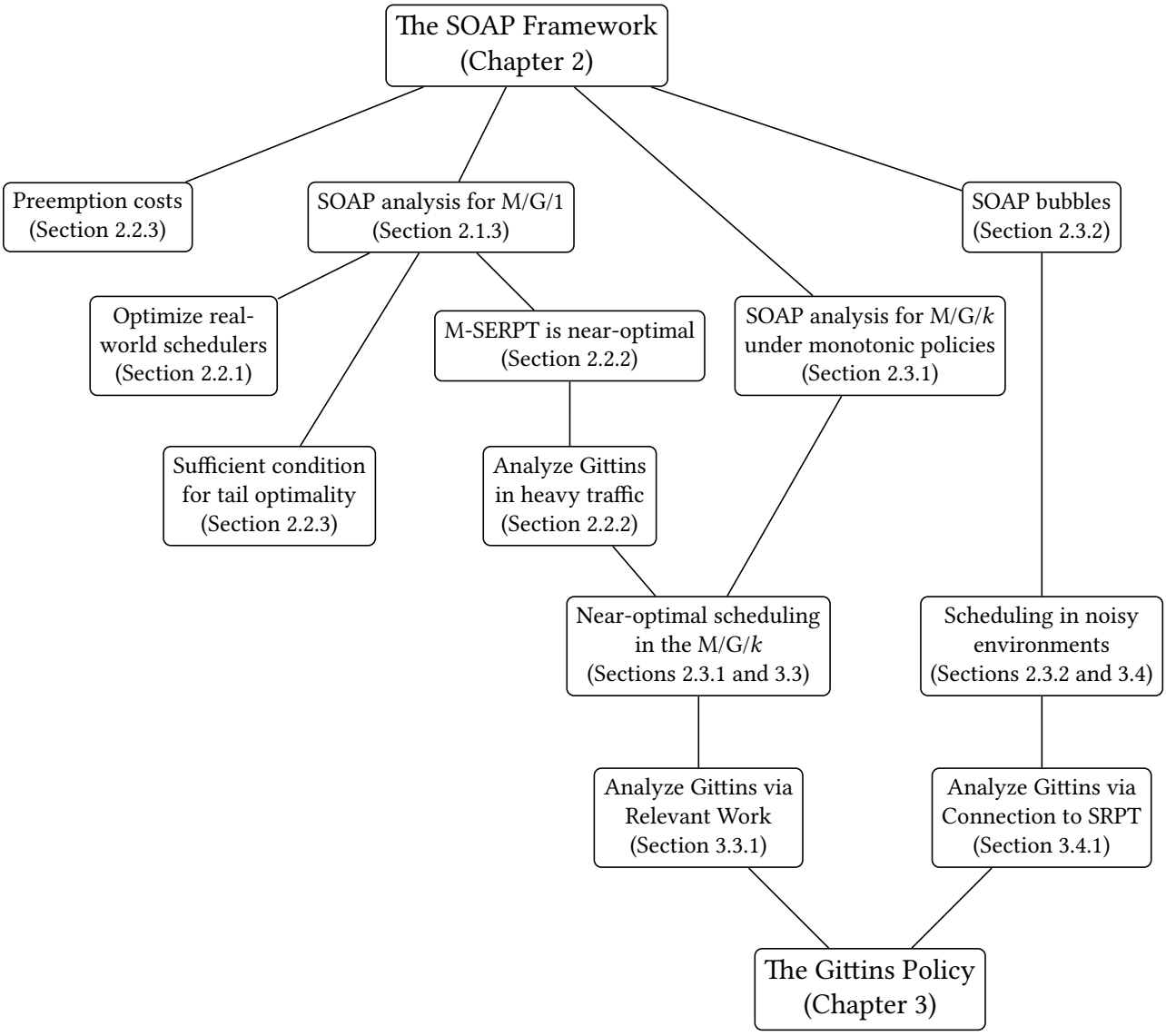


Figure 1.1.1. Map of Proposed Work

If there is time, I hope to also include progress on the following additional problems:

- We now have a simple alternative to Gittins, namely M-SERPT, with near-optimal performance when jobs have unknown sizes. Are there simple near-optimal alternatives to Gittins for more complicated settings, such as when each job gives the scheduler a noisy size estimate (Section 3.4)?
- Precisely modeling preemption costs is actually outside the realm of SOAP, so the results on preemption costs mentioned above use a simplified model. Can we extend the SOAP analysis with a detailed and realistic model of preemption costs (Section 2.3.3)?

Figure 1.1.1 summarizes the problems we study and the new techniques we propose to solve them.

1.2 Outline

The next two chapters of this proposal describe the above proposed work in much more detail. Each chapter focuses on one of the major theoretical themes and proposes solving several problems within that theme. Figure 1.1.1 summarizes the following outline as a picture.

Chapter 2 covers the *SOAP framework*. SOAP is a new way of describing and analyzing a wide variety of scheduling policies. It enables near-optimal scheduling by allowing us to analyze the response time of a much wider space of scheduling heuristics than was previously possible. After introducing SOAP (Section 2.1), we discuss how to apply SOAP to solve several open problems (Section 2.2). We then discuss some scenarios where we can extend SOAP to settings beyond the standard M/G/1, such as systems with multiple servers (Section 2.3).

Chapter 3 covers the *Gittins policy*. While Gittins has long been known to minimize mean response time in the M/G/1, many aspects of Gittins are poorly understood. For example, despite knowing its optimality, the mean response time achieved by Gittins in the M/G/1 is not known in the general case. While SOAP provides one way of analyzing Gittins's mean response time, it turns out that Gittins in particular admits a variety of analyses, each of which provides different insights. After introducing Gittins (Section 3.2), we describe two new ways of analyzing its mean response time, each of which helps us solve a different open problem. First, we prove Gittins is nearly optimal in the M/G/k (Section 3.3). Second, we use construct a simple policy for scheduling in systems with noisy size estimates (Section 3.4).

Throughout this document, I highlight each problem I propose to solve. Each problem is followed by a summary of the progress I have made towards solving it. An example follows.

Problem 1.2.1. Write a PhD thesis.

Progress: 1%. This proposal is just the beginning....

2 The SOAP Framework

Queueing theorists have been analyzing the response time of scheduling policies for more than half a century, from classic works [14, 24–26, 34, 36, 43] to recent advances [1, 8, 9, 23, 27, 32, 47]. Here are some examples of common scheduling policies:

- *First-come, first-served* (FCFS) serves jobs nonpreemptively in the order they arrive.
- *Class-based priority* serves the job of highest priority class, possibly preemptively and possibly nonpreemptively.
- *Shortest remaining processing time* (SRPT) preemptively serves the job with the least remaining time.
- *Foreground-background* (FB) preemptively serves the job of least age, namely the job that has received the least service so far.
- *Processor sharing* (PS) concurrently serves all jobs in the system at the same rate.

In just these few examples we see a variety of features represented: preemptible jobs, nonpreemptible jobs, prioritizing by class, prioritizing by job size, and prioritizing by age. Each policy requires a custom response time analysis that takes into account its particular combination of features.

Although there has been much success in analyzing the response time of specific scheduling policies in the M/G/1 setting, such as those listed above, results are ad-hoc and *limited to relatively simple policies*. Analyzing variants of the above simple policies, let alone fundamentally different policies, is an open problem. For instance, none of the following scenarios have been analyzed before:

- Suppose we have exact size information for some "sized" jobs but not other "unsized" jobs. We run a mixture of two policies, using SRPT on sized jobs and FCFS on unsized jobs (Example 2.1.1).
- Suppose we have jobs that are neither fully preemptible nor fully nonpreemptible but instead preemptible only at specific "checkpoint" ages. We run a preemptive policy, such as SRPT or FB, but only preempt jobs when they reach checkpoint ages (Example 2.1.2).
- The *Gittins policy* [2, 15], long known to minimize mean response time in the M/G/1 queue, has only been analyzed in certain special cases [22, 32]. In general, the Gittins policy can have a complex priority scheme [3] which, while known to perform optimally, has not been analyzed before in its general form (Example 2.1.3).

Approaching the above examples with state-of-the-art techniques, if possible at all, would require an ad-hoc analysis for each scenario. We seek *general principles and techniques* for response time analysis that apply to not just the above examples but to as many scheduling policies as possible, even those not yet imagined.

2.1 What is SOAP?

SOAP, which stands for *Schedule Ordered by Age-based Priority*, is a new framework for defining and analyzing scheduling policies in great generality. The SOAP framework consists of two pieces.

- The first piece of SOAP is a *broad class of scheduling policies* (Sections 2.1.1 and 2.1.2). We say "SOAP policies" or "SOAP class" to refer to this class of policies.
- The second piece of SOAP is a *universal response time analysis* that applies to all SOAP policies (Section 2.1.3). We say "SOAP analysis" to refer to this universal analysis.

2.1.1 The Class of SOAP Policies

The major challenge in defining the SOAP class is coming up with a general enough framework to express a wide variety of scheduling policies. Looking at the examples at the start of Chapter 2, we see that aside from FCFS and PS, all of the policies can be seen as working in the same general way: each job receives a priority, and the system serves the job with the best priority. The SOAP class formalizes this idea, specifying what a job's priority is allowed to depend on and what to do when ties occur.

Every SOAP policy is described by a *rank function* which assigns a numerical *rank*, or priority, to each job.[1] All SOAP policies have one scheduling rule:

> Every moment in time, serve the job of *minimum rank*, breaking ties in FCFS order.[2]

A rank function r takes the following two inputs:

- A job's *age* a, which is the amount of service it has received so far.
- A job's *metadata* m, which can contain any additional static information about the job. For example, a job's metadata might specify a size, size estimate, or priority class. The main restriction is that metadata is static, meaning a job's metadata does not change over time.

We write the rank of a job with age a and metadata m as $r_m(a)$. For *blind* policies, namely those that do not use metadata, we omit the subscript.[3] The class of SOAP policies contains all policies that can be encoded by a rank function.

2.1.2 Examples of SOAP Policies

The SOAP class includes a huge variety of policies, including classic policies like SRPT, complex policies like Gittins, and an infinite number of ways to mix and match policies.

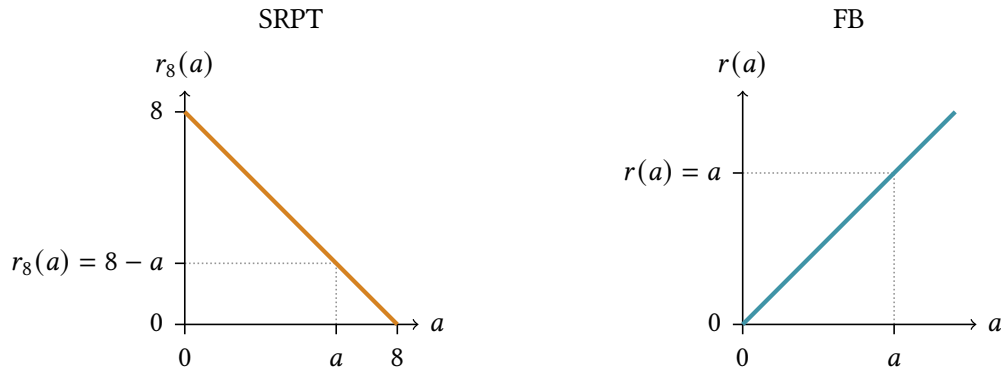
Below are some examples of classic scheduling policies in the SOAP class, along with their rank functions. Figure 2.1.1 illustrates two examples.

- FCFS is a blind SOAP policy with rank function $r(a) = 0$. That is, all jobs are always tied, so we always follow the FCFS tiebreaking rule.
- Both preemptive and nonpreemptive class-based priority are SOAP policies. Suppose the priority classes ordered from best to worst priority are $1, \dots, n$. Here a job's metadata is its class $i \in \{1, \dots, n\}$.
  - Preemptive class-based priority has rank function $r_i(a) = m$.

[1] One can use an ordered set of possible ranks other than the real numbers, but using the reals suffices for all of the examples we discuss.

[2] SOAP can actually be generalized to use FCFS, LCFS, or hybrids between them as tiebreaking rules. For ease of presentation, we focus on the most practical case, namely FCFS tiebreaking.

[3] Strictly speaking, we represent this case as all jobs having the same "dummy" metadata, which we omit from the notation to reduce clutter.



Left: the SRPT rank function, shown for a job of size 8. *Right:* the FB rank function.

Figure 2.1.1. Examples of Rank Functions of Classic Scheduling Policies

  - Nonpreemptive class-based priority has rank function

$$r_i(a) = \begin{cases} i & \text{if } a = 0 \\ -1 & \text{otherwise.} \end{cases}$$

    That is, if all jobs are at age 0, meaning no job is in service, we start serving the job in the best priority class. But once a job starts service, its rank drops to -1, so it remains in service even if a job in a better priority class arrives later.
- SRPT is a SOAP policy. A job's metadata is its size s, and the rank function is $r_s(a) = s - a$
- FB is a blind SOAP policy with rank function $r(a) = a$.

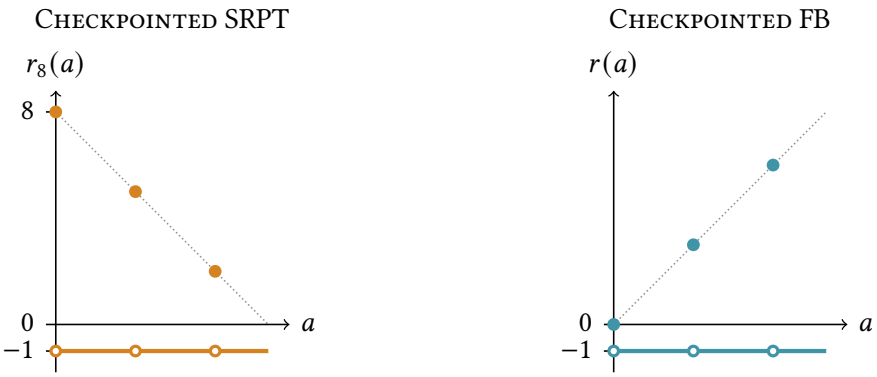
That is, aside from PS, all of the policies mentioned at the start of Chapter 2 are SOAP policies. The remaining examples in this section are of scheduling policies that could not be analyzed prior to SOAP.

Example 2.1.1 (Mixture of Two Policies)**.** Consider a system with two job classes: humans and robots.

- Humans, unpredictable and easily offended, have unknown sizes, are nonpreemptible, and insist on FCFS service relative to other humans. We denote humans with metadata human.
- Robots, precise and ruthlessly efficient, have known sizes, are preemptible, and insist on SRPT service relative to other robots. We denote a robot of size s with metadata robot(s).

A reasonable policy for this system might be to prioritize as follows, from best to worst:

- humans that have started service, then
- robots with small remaining size, then
- humans that have not started service, then finally
- robots with large remaining size.



Left: the checkpointed SRPT rank function with checkpoint ages every 3 time units, shown for a job of size 8. *Right:* the checkpointed FB rank function with checkpoint ages every 3 time units.

Figure 2.1.2. Expressing Preemption Checkpoints with Rank Functions

Letting s_{human} be the threshold between "small" and "large" sizes for robots, this policy is a SOAP policy with rank function

$$r_{\text{human}}(a) = \begin{cases} s_{\text{human}} & \text{if } a = 0 \\ -1 & \text{otherwise,} \end{cases}$$
$$r_{\text{robot}(s)}(a) = s - a.$$

Example 2.1.2 (Preemption Checkpoints)**.** Suppose we want to schedule using SRPT, a preemptive policy, but the jobs in our system are only preemptible at specific checkpoint ages. Adapting SRPT to such a system yields the *checkpointed SRPT* policy, which has rank function

$$r_s(a) = \begin{cases} s - a & \text{if } a \text{ is a checkpoint age} \\ -1 & \text{otherwise.} \end{cases}$$

One can express checkpointed versions of other preemptive policies using similar rank functions. For example, *checkpointed FB* has rank function

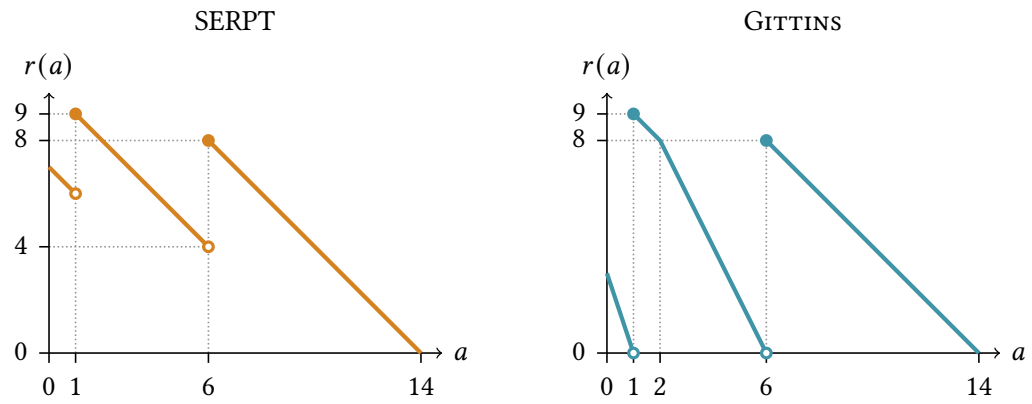
$$r(a) = \begin{cases} a & \text{if } a \text{ is a checkpoint age} \\ -1 & \text{otherwise.} \end{cases}$$

Figure 2.1.2 illustrates both of the above rank functions.

Example 2.1.3 (Using the Size Distribution)**.** Suppose we are scheduling in a system where a job's metadata m does not necessarily determine its exact size. To achieve low response time, we want to bias towards jobs with lower remaining size. One policy that does this is *shortest expected remaining processing time* (SERPT), which serves the job of least expected remaining size. SERPT is a SOAP policy with rank function

$$r_m(a) = \mathbf{E}[S_m - a \mid S_m > a],$$

where S_m is the size distribution of jobs with metadata m.



Left: the SERPT rank function for a blind system in which a job's size S is 1, 6, or 14, each with probability 1/3. *Right:* the Gittins rank function for the same system.

Figure 2.1.3. Examples of the SERPT and Gittins Rank Functions

The above definition of SERPT works for a variety of settings, depending on the available metadata. On one extreme is the blind case, in which $r(a) = \mathbf{E}[S - a \mid S > a]$, where S is the overall job size distribution. See Figure 2.1.3 for an example On the other extreme is the case where each job's metadata is its size s. Here $S_s = s$, and thus SERPT becomes equivalent to SRPT.

Despite its intuitive appeal, SERPT does not in general minimize mean response time. That honor goes to the *Gittins* policy,[4] which is a SOAP policy with rank function

$$r_m(a) = \inf_{b>a} \frac{\mathbf{E}[\min\{S_m - a, b\} \mid S_m > a]}{\mathbf{P}[S_m \le b \mid S_m > a]}.$$

Roughly speaking, one can view Gittins as a refinement of SERPT: if one chooses $b = \infty$ instead of optimizing in the above equation, one obtains the SERPT rank function. Section 3.2 gives additional intuition for the Gittins rank function.

2.1.3 M/G/1 Response Time Analysis

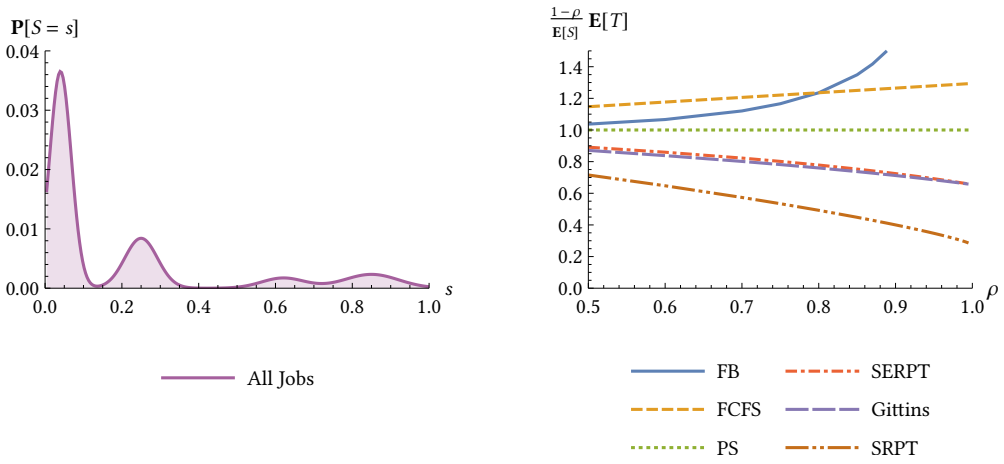
The main technical contribution of the SOAP framework is a universal response time analysis that works for all SOAP policies.

Problem 2.1.1. Derive formulas for the mean and Laplace-Stieltjes transform of response time in the M/G/1 under a general SOAP policy.

Progress: 100%. We derive general formulas in [40]. In certain special cases, such as blind policies with monotonic rank functions, we can give simpler formulas which are useful in some applications [38, 41].

The formulas depend on

[4] Specifically, Gittins minimizes mean response time in the M/G/1 when the information available to the scheduler is just each job's age and metadata.



Left: a job size distribution S. *Right:* mean response time $\mathbf{E}[T]$ as a function of load ρ under several scheduling policies in an M/G/1 with job size distribution S and arrival rate $\lambda = \rho/\mathbf{E}[S]$. Mean response time is normalized to that of PS, which is $\mathbf{E}[S]/(1-\rho)$.

Figure 2.2.1. Comparing SERPT and Gittins to Classic Scheduling Policies

- the rank function r;
- for each possible metadata value m, the size distribution S_m of jobs with metadata m; and
- the arrival rate, or equivalently the load.

The exact formulas and the techniques used to prove them are beyond the scope of this proposal, but I will describe them in detail in the thesis.

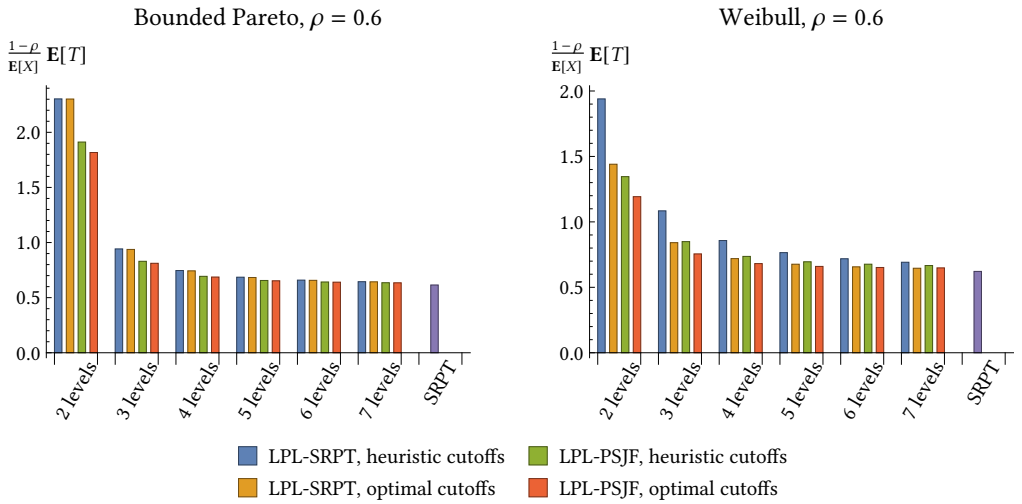
2.2 Solving Open Problems with SOAP

The SOAP analysis immediately solves a number of open problems. For example, because SERPT and Gittins are SOAP policies (Example 2.1.3), we can obtain the mean response times of SERPT and Gittins for any specific M/G/1 system, which was previously intractable. This enables us to create comparisons like Figure 2.2.1 for the first time.

In the rest of this section, we describe how to apply SOAP to solve a number of other open problems. We begin with case studies that show how to directly apply the SOAP analysis to optimize scheduling policies in a variety of computer systems (Section 2.2.1). We then show how to use SOAP to solve several open theoretical problems (Sections 2.2.2–2.2.4). The main difference between the practical and theoretical applications is that while the practical case studies deal with *specific* M/G/1 systems, our theoretical results prove statements about *all* M/G/1 systems satisfying certain properties.

2.2.1 System Design Case Studies

The SOAP analysis is a powerful tool, but it does not by itself does not answer difficult policy design questions. We frame several difficult yet common design problems in terms of SOAP policies,



Mean response time for two different policies, *limited priority level SRPT* (LPL-SRPT) and *limited priority level preemptive shortest job first* (LPL-PSJF), as a function of the number of priority levels. LPL-SRPT groups jobs into priority levels based on their *remaining* size, while LPL-PSJF uses *original* size instead. Surprisingly, LPL-PSJF outperforms LPL-SRPT. For each policy, we consider two ways of choosing size cutoffs between priority levels. We set the *heuristic cutoffs* such that an equal fraction of jobs have original size in each priority level. We set the *optimal cutoffs* by using the SOAP analysis to numerically optimize the cutoffs to minimize mean response time. *Left:* comparison of the previously described policies for a bounded Pareto job size distribution. *Right:* the same for a Weibull job size distribution.

Figure 2.2.2. Achieving Near-Optimal Mean Response Time with a Limited Number of Priority Levels

enabling us to use the SOAP analysis to gain insight into them.

Problem 2.2.1. Gittins minimizes mean response time, but its rank function is complex and potentially difficult to implement. *Is this added complexity worthwhile, or do simpler scheduling policies have comparable mean response time?* How does the answer change depending on the available metadata?

Progress: 80%. We address this in work that we are preparing for publication.

We find that in many cases where job sizes are not exactly known, *Gittins significantly outperforms simple heuristics*. The exception is in scenarios where we can use job metadata to cleanly separate jobs into classes based on approximate size. In such cases, simple class-based priority policies can perform comparably to Gittins.

Problem 2.2.2. Some scheduling systems, such as network switches and the Linux scheduler, have only a limited number of priority levels [21, 29]. *How should we schedule given limited priority levels?* How many priority levels do we need for near-optimal response time?

Progress: 80%. We address this in work that we are preparing for publication.

We find that *limited priority level SRPT* (LPL-SRPT), a version of SRPT that groups jobs into priority levels based on remaining size, achieves mean response time close to that of ordinary SRPT. For highly variable job size distributions, *around six priority levels suffices*. Figure 2.2.2 illustrates this for two versions of LPL-SRPT, which differ in how the choose the size cutoffs between priority levels, as well as two version of a similar policy called LPL-PSJF.

Problem 2.2.3. We considered systems with preemption checkpoints in Example 2.1.2. However, in practice, these preemption checkpoints incur an overhead, which prompts a question: *how frequent should checkpoints be?* Decreasing the interval between checkpoints allows for more flexible preemption and thus smarter scheduling, which decreases mean response time. However, each checkpoint also incurs time overhead, so decreasing the interval between checkpoints adds work to the system, which increases mean response time or even causes instability. How do we balance this tradeoff?

Progress: 80%. We address this in work that we are preparing for publication.

We give a formula for a *minimum safe checkpoint interval* such that any larger checkpoint interval is guaranteed to yield a stable system. The optimal checkpoint interval tends to be slightly larger than this minimum, and we provide a simple rule of thumb that approximates the optimum. The general trend is that the higher the load and the higher the overhead, the more time we need between checkpoints.

2.2.2 A Simple Policy with Near-Optimal Mean Response Time

Recall the SERPT and Gittins policies from Example 2.1.3. The Gittins policy minimize mean response time, but its rank function is determined by a complicated formula that can be difficult to compute. In contrast, SERPT has a simpler rank function formula, but despite its intuitive appeal, there is no guarantee that it achieves near-optimal mean response time. With that said, Figure 2.2.1 shows SERPT achieving mean response time very close to Gittins's.

Problem 2.2.4. Prove that SERPT or a similarly simple policy achieves near-optimal mean response time in the M/G/1.

Progress: 100%. We do this for the blind setting in [41].

While we have not yet managed to prove a result for SERPT, we have shown that a variation of SERPT called *monotonic SERPT* (M-SERPT) is a 5-approximation for mean response time for blind scheduling in the M/G/1. M-SERPT's rank function is

$$r(a) = \max_{0 \le b \le a} \mathbf{E}[S - b \mid S > b],$$

where S is the job size distribution. This rank function is similar to SERPT's, but the running maximum ensures it never decreases.

Some remaining questions include whether M-SERPT's true approximation ratio is actually less than 5, whether ordinary SERPT also has a constant approximation ratio, and whether a similar result holds in non-blind settings where jobs have metadata; but I do not expect to address these in the thesis.

2.2.3 Optimizing the Asymptotic Tail of Response Time

We have thus far focused on minimizing *mean* response time $\mathbf{E}[T]$, but in practice, other objectives can be more important than minimizing the mean. Many practitioners focus instead on the *tail* of response time $\mathbf{P}[T > t]$. While directly studying the tail is difficult to do with existing theoretical

tools, studying its *asymptotic behavior* in the $t \to \infty$ limit is more tractable, at least for simple policies like SRPT, FB, and PS [30]. However, determining the asymptotic tail behavior of more complicated policies, such as SERPT and Gittins, remains an open problem.

For a wide class of heavy-tailed job size distributions S, namely those that are intermediate regularly varying [11], it turns out that SRPT, FB, and PS are all *tail-optimal*, meaning they have asymptotically minimal response time tails. Specifically, they satisfy $\mathbf{P}[T > t] \sim \mathbf{P}[S > (1-\rho)t]$ at load ρ.

This prompts a question: is Gittins tail-optimal? An affirmative answer would be significant, because it would suggest that aiming to optimize mean response time also yields good tail behavior. More generally, we might ask which SOAP policies are tail-optimal.

Problem 2.2.5. Give conditions on a SOAP policy's rank function that determine whether or not it is tail-optimal for heavy-tailed job size distributions. In particular, are SERPT and Gittins tail-optimal?

Progress: 60%. We address the case of blind scheduling in [42], and we are preparing follow-up work for submission.

We give a condition on the rank function and job size distribution that implies tail optimality The simplest version of the condition is roughly as follows.[5] Suppose $\Omega(a^{-\beta}) \le \mathbf{P}[S > a] \le O(a^{-\alpha})$ and $\Omega(a^{\gamma}) \le r(a) \le O(a^{\delta})$. The SOAP policy with rank function r is tail-optimal if

$$\frac{\delta}{\gamma} - \frac{\gamma}{\delta} < \frac{\alpha - 1}{\beta}.$$

In particular, $\gamma = \delta = 1$ suffices, which turns out to imply that *SERPT and M-SERPT are tail-optimal*.

We can also show that *Gittins is tail-optimal* using a more complicated version of the condition. Precisely characterizing this more complicated condition and verifying that Gittins satisfies it is the follow-up work we are preparing for submission.

2.2.4 Gittins in Heavy-Traffic

The SOAP analysis gives a mean response time formula for Gittins. It is straightforward to use the formula to compute Gittins's mean response time for any *specific* M/G/1 system, as in Figure 2.2.1. However, the formula is complicated enough that it does not give a good *general* picture of Gittins's mean response time, even if we restrict to the case of blind scheduling.

As a first step to understanding Gittins's mean response time in general, we might ask how it scales as a function of load ρ in the $\rho \to 1$ limit,[6] also called the *heavy-traffic* limit.

Problem 2.2.6. Characterize the asymptotic scaling of Gittins's mean response time in the heavy-traffic M/G/1.

Progress: 95%. We address the blind setting in [38]. It is possible that techniques we discuss in Section 3.4 will help slightly generalize our results.

[5] Specifically, $-\beta$ and $-\alpha$ should be defined as the lower and upper Matusewska indices [7], respectively, of $a \mapsto \mathbf{P}[S > a]$.

[6] The system becomes unstable as ρ approaches 1.

We derive explicit formulas for the heavy-traffic scaling of Gittins for a wide class of job size distributions. Our analysis makes liberal use of M-SERPT, which has a simpler rank function than Gittins but has the same heavy-traffic scaling, and other techniques that we develop when solving Problem 2.2.4.

2.3 Extending SOAP Beyond the Standard M/G/1

SOAP allows us to analyze the response time of much more realistic scheduling policies than previously possible. However, the SOAP analysis is limited to a standard M/G/1 model: there is a single server (Section 2.3.1), the scheduler has perfect information about each job's state (Section 2.3.2), and preemptions incur no overhead (Section 2.3.3). In the rest of this section, we discuss extensions of SOAP that can handle each of these cases.

2.3.1 M/G/k Response Time Analysis

The multiserver analogue of the M/G/1 is the M/G/k. SOAP policies naturally generalize to the M/G/k: serve the k jobs of minimal rank, breaking ties in FCFS order. This leads to a natural question: can we analyze the response time of SOAP policies in the M/G/k?

Analyzing M/G/k response time even under simple FCFS scheduling is a famously difficult problem. It thus seems like it would be even more difficult to analyze the M/G/k under a SOAP policy that is much more complicated rank function. Fortunately, it turns out that under certain SOAP policies, it is possible to obtain meaningful response time bounds in the M/G/k.

The main restriction on our analysis is that it applies only to *monotonic* SOAP policies. These are policies such that for all metadata m, the rank $r_m(a)$ is a monotonic function of age a, where the monotonicity direction is the same for all m. For example, SRPT is a monotonic SOAP policy because a job's rank, namely its remaining size, always decreases with age (Section 2.1.2).

Problem 2.3.1. Derive explicit bounds for mean response time in the M/G/k under a monotonic SOAP policy.

Progress: 80%. We address the blind setting in [38] and the known-size setting in [19]. We have plans to slightly strengthen these results and to generalize them to arbitrary job metadata.

We bound M/G/k response time by comparing it to the M/G/1 response time formula (Problem 2.1.1). To make the comparison fair, we assume each of the M/G/k's servers speed $1/k$, as shown in Figure 2.3.1 which gives both systems the same total service capacity. We can decompose M/G/1 mean response time $\mathbf{E}[T_1]$ into two pieces denoted $\mathbf{E}[Q]$ and $\mathbf{E}[R]$:

$$\mathbf{E}[T_1] = \mathbf{E}[Q] + \mathbf{E}[R].$$

Roughly speaking, we show that only the second term increases when moving to M/G/k mean response time $\mathbf{E}[T_k]$:

$$\mathbf{E}[T_k] \le \mathbf{E}[Q] + k\mathbf{E}[R] + k\mathbf{E}[R'],$$

where $\mathbf{E}[R']$ is a quantity whose formula is similar to $\mathbf{E}[R]$'s. We further show that $\mathbf{E}[R'] \le \mathbf{E}[R]$ for several important policies, including SRPT, M-SERPT, and *monotonic Gittins* (M-Gittins).[7]

[7] M-Gittins's rank function is defined in terms of Gittins's in the same way that M-SERPT's is defined in terms of SERPT's.

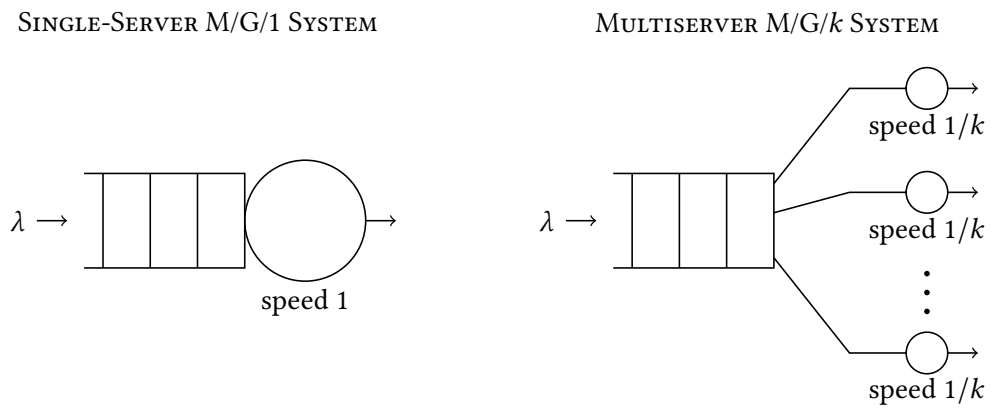


Figure 2.3.1. M/G/1 and M/G/k Server Speeds

Problem 2.3.2. Determine which scheduling policies have optimal mean response time in the heavy-traffic M/G/k.

Progress: 100%. We address the blind setting in [38] and the known-size setting in [19].

We show the following results for the M/G/k in the heavy-traffic limit. Both results apply to "most" finite-variance job size distributions.

- In the blind setting, M-Gittins asymptotically minimizes mean response time.
- In the known-size setting, SRPT and similar policies asymptotically minimize mean response time, provided that the job size distribution S satisfies $\mathbf{E}[S^2(\log S)^+] < \infty$.

These results follow from the solution to Problem 2.3.1, which for the above policies reduces to

$$\mathbf{E}[T_1] = \mathbf{E}[Q] + \mathbf{E}[R],$$
$$\mathbf{E}[T_k] \le \mathbf{E}[Q] + 2k\mathbf{E}[R].$$

It thus suffices to show that $\mathbf{E}[Q]$ dominates $\mathbf{E}[R]$ in the heavy-traffic limit. This is already known for SRPT [27]. We show this for M-Gittins as part of the heavy-traffic analysis in Problem 2.2.6, which applies to M-Gittins as well as Gittins.

Notably missing from the above discussion is Gittins. We address Gittins later in the thesis using techniques not based on SOAP (Section 3.3).

2.3.2 Scheduling with Noisy System State Information

To implement the Gittins policy or any other SOAP policy, the scheduler needs to know each job's exact age at every moment in time. However, in some computer systems the scheduler only knows each job's *approximate age*. For example, in a time-shared system, it can be hard to precisely track the CPU time allocated to a particular job without incurring significant overhead. As another example, in a networked system, a controller may not be on the same machine as the jobs it is scheduling, in which case updating age information over the network incurs a delay or cost.

Problem 2.3.3. Design a scheduling policy that has near-optimal mean response time while being robust to noise in each job's age.

Progress: 100%. We solve this problem in [39].

Our solution is a generic construction that, for any SOAP policy, constructs a new version of that policy which is robust to adversarial noise in jobs' ages. Specifically, given a rank function and a maximum age perturbation Δ, we construct a new *robust* rank function with the following property: if we schedule by plugging each job's noisy age into the robust rank function and serving the job of lowest rank, the mean response time is at most the no-noise mean response time plus a continuous function of Δ. Surprisingly, the construction is robust enough that the result holds even under *adversarial* noise models. To prove these results, we generalize the SOAP analysis to "SOAP bubble" policies, which assign each job a range of ranks instead of a single rank.

2.3.3 Preemption Costs

The SOAP analysis assumes that preempting a job incurs no overhead, but this is not a practical assumption in many computer systems. Accounting for preemption overhead is very difficult and has only been done in a few simple cases [18, 44].

Problem 2.3.4. Account for preemption overhead in the SOAP analysis.

Progress: 40%. In ongoing work, we have analyzed the preemptive class-based priority policy (Section 2.1.2) in a system where starting or stopping a job incurs overhead. We also have some preliminary progress on analyzing SOAP policies with more complicated rank functions.

2.4 Related Work

Many specific SOAP policies have been analyzed before [20, 26, 32, 34, 36, 43]. The SOAP analysis combines ideas from these prior works in a novel way. The resulting analysis is more general, yielding a single response time formula that works for all SOAP policies, and more powerful, covering policies for which no such formula previously existed.

One of the key ideas of SOAP is describing an entire class of policies, namely those describable by a rank function, in a unified way. The SOAP analysis thus builds on a short but notable tradition of analyzing an entire class of policies at once. Two examples are the *SMART* [47] and *multilevel processor sharing* (MLPS) [26] classes.

- The MLPS class, introduced by Kleinrock [26], consists of policies that divide all jobs in the system into echelons based on age. The system serves jobs in the youngest possible echelon. Within each echelon, jobs are served using FCFS, FB, or PS. Some recent work on MLPS policies includes optimally choosing the age echelon cutoffs [6] and connecting MLPS to the Gittins policy [3].
- The SMART class, introduced by Wierman et al. [47], includes all policies that satisfy certain criteria that ensure they prioritize small jobs over large ones, such as SRPT. Some recent work on SMART policies includes analyzing the tail behavior of response time [31] and characterizing the tradeoff between accuracy of size estimates and response time [48].

While the SMART and MLPS classes have nearly no overlap, the SOAP class includes many policies from *both* classes. Specifically, the SMART$*$ subclass of SMART [47] and MLPS policies which do not use PS echelons are all SOAP policies.

3 The Gittins Policy

The question of how to schedule jobs so as to minimize mean response time is a classic question in the queueing literature. In the M/G/1 queue, the answer is the *Gittins* policy. Gittins minimizes mean response time in a huge variety of settings [15]. These include

- the case where we have *perfect information* about each job's size;
- the case "blind" case where we have *zero information* about any job's size, but the job size distribution is known;
- a wide range of cases where we have *partial information* about each job's size; and
- a wide range of cases where we have *dynamically evolving information*, meaning we learn more about a job's size as we serve it.

The way Gittins works is as follows: based on the information the scheduler knows about a job, Gittins assigns a numerical rank to it, and at every moment in time, Gittins serves the job of lowest rank. Roughly speaking, a job has low rank if it is likely to complete soon (Section 3.2). For example, in the case of perfect job size information, a job's gittins rank is its remaining size, which makes Gittins equivalent to SRPT. More generally, if the information known about a job can be encoded as some static metadata and the job's age, then Gittins is a SOAP policy (Example 2.1.3).

Despite the strong results that exist for Gittins in the M/G/1, several major obstacles stop Gittins from being a practical choice. Two such obstacles and the open problems they invite are the following:

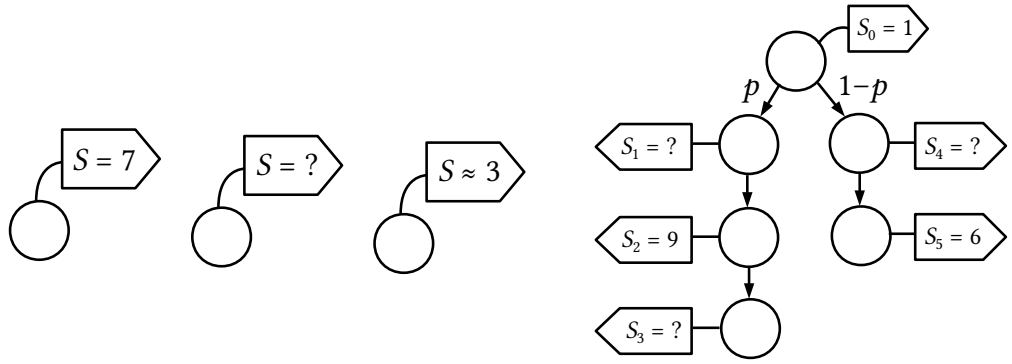
- Many systems have multiple servers, but we do not know how well Gittins performs in multiserver models, such as the M/G/k. Does Gittins have near-optimal performance in the M/G/k?
- Determining a job's Gittins rank requires solving an optimization problem that can be intractable in general. Can we simplify Gittins while still having near-optimal performance?

We note that we have already started to address these problems in Chapter 2, studying the M/G/k in the zero- and perfect-information cases (Problem 2.3.2) and showing that a simple policy, M-SERPT, is near-optimal in the M/G/1 in the zero-information case (Problem 2.2.4). However, both problems remain open in the extremely broad range of cases where the scheduler has partial or dynamically evolving information about each job's size.

It turns out that to solve both of the above problems, we need to *more deeply understand Gittins in the M/G/1*. Specifically, we will derive two new formulas for Gittins's mean response time, each of which leads us to techniques that solve one of the problems. The techniques behind the first formula lead us to a way to bound Gittins's mean response time in the M/G/k, showing it has near-optimal performance (Section 3.3). The techniques behind the second formula lead us to deriving simple policies for some partial-information cases (Section 3.4).

3.1 What is a Job?

Our job model encodes the process by which the scheduler gains information about a job's service requirement. We ensure that we can handle a highly general set of information-revealing scenarios,



From left to right: a job with perfect job-size information (Example 3.1.1); a job with zero job-size information (Example 3.1.2); a job with partial job-size information (Example 3.1.3); and a multistage job where some stages have perfect information, some stages have zero information, and the first stage transition is probabilistic (Example 3.1.4).

Figure 3.1.1. Examples of Scenarios in which Gittins Minimizes Mean Response Time

including four scenarios illustrated in Figure 3.1.1.

The simplest scenarios are *perfect information*, where the scheduler knows each job's service requirement immediately upon the job's arrival to the system; and *blind* or *zero information*, where the scheduler only knows the overall service requirement distribution, but not an individual job's service requirement.

We can also model a variety of *partial information* scenarios. In such a scenario, upon each job's arrival the scheduler is given a service requirement distribution for that job that is more specific than the overall service requirement distribution. For instance, the scheduler could be given a service requirement estimate where the joint distribution of true service requirement and the estimate is known. This model is the size-estimate model considered by Mitzenmacher [28].

However, we need not only consider scenarios where information is only revealed when the job arrives in the system: it may be that we have *dynamically evolving information*, learning more about a job's size as we serve it. For example, a job may be broken up into a series of several stages, where each stage has unknown size but known size distributions, with the scheduler learning when each stage completes. This model is the multistage job model considered by Mitzenmacher [28]. An example of such a job is shown on the right side of Figure 3.1.1.

We unify these scenarios by encapsulating the information known to the scheduler into the *state* of a job. Our model's generality comes from allowing job's state to evolve according to an arbitrary Markov process (Section 3.1.1). It is general enough to encompass all of the examples in Figure 3.1.1 (Section 3.1.2).

3.1.1 Modeling Jobs as Markov Processes

We model jobs as continuous-time absorbing Markov processes. The intuition is that a job's state encodes everything the scheduler knows about the job. A job's state evolves stochastically as it is served. The evolution of the state is not dependent on any decisions by the scheduler, beyond choosing whether to run the job or not.

Each job follows an i.i.d. Markovian trajectory $\{X(s)\}_{s\geq 0}$ through some *job state space* $\mathbb{X}$. Here

s is a job's age and $X(s)$ is the random state of a job at age s. A job's state does not change if it is not in service.

Each arriving job enters the system in an i.i.d. *arrival state* $X_0 = X(0)$, which is the random initial state of the job's trajectory through the state space. A job completes and exits the system when it enters a designated *departure state* $\top \in \mathbb{X}$, which is the unique absorbing state. A job's service requirement, which we assume to have finite expectation, is the amount of service it needs to reach the departure state:

$$S \coloneqq \min\{s \geq 0 \mid X(s) = \top\}.$$

3.1.2 Examples of the Markov Processes Job Model

In this section we show how to model the jobs illustrated in Figure 3.1.1 using the Markov process job model.

Example 3.1.1 (Perfect Information)**.** Consider a system where the scheduler knows each job's exact size and preemption is allowed at any time. To model this, we let a job's state be its remaining size. The job state space is $\mathbb{X} = \mathbb{R}_{\geq 0}$ with departure state $\top = 0$. A job's trajectory is $X(s) = (S - s)^+$. Although the arrival state $X_0 = S$ is stochastic, the dynamics are deterministic: a job's state decreases at rate 1 until it reaches 0.

Example 3.1.2 (Zero Information)**.** Consider a system where the scheduler has zero a priori knowledge of any individual job's size, but the overall size distribution S is known. To model this, we let a job's state be its age. The state space is $\mathbb{X} = \mathbb{R}_{\geq 0} \cup \{\top\}$, and a job's trajectory is

$$X(s) = \begin{cases} s & \text{if } s < S \\ \top & \text{if } s \geq S. \end{cases}$$

Here the arrival state $X_0 = 0$ is deterministic, but the dynamics are stochastic: a job's state increases at rate 1 with stochastic jumps to the departure state $\top$, where the jump intensity is the hazard rate of the size distribution S.

Example 3.1.3 (Partial Information: Noisy Size Estimates)**.** Consider a system where the scheduler receives an estimate of each job's size which is correlated with its true size. Specifically, suppose that each job announces to the scheduler an i.i.d. estimated size M, and suppose the conditional distribution $(S \mid M)$ is known. To model this, we let a job's state be a tuple of its size estimate and its age. The state space is $\mathbb{X} = \mathbb{R}^2_{\geq 0} \cup \{\top\}$, and a job's trajectory is

$$X(s) = \begin{cases} (M, s) & \text{if } s < S \\ \top & \text{if } s \geq S. \end{cases}$$

Here the arrival state $X_0 = (M, 0)$ is stochastic, as are the dynamics: the second element of a job's state increases at rate 1, and there are stochastic jumps to the departure state $\top$ according to the hazard rate of $(S \mid M)$.

We note that although we have focused on the case where M is a job's estimated size, we can use the same technique to model any scenario in which jobs are "tagged" with static information. For example, in a system with multiple job classes, M could be a job's class.

Example 3.1.4 (Dynamically Evolving Information: Multistage Jobs)**.** Consider a system where a job its broken up into stages, and upon completing stage i moves to stage j with probability p_{ij}, or else completes. Suppose the scheduler knows when a job completes a stage, and what stage the job transitions to. Furthermore, suppose the scheduler has zero knowledge of the length of each stage, but the stage size distributions S_i and transition probabilities p_{ij} are known. To model this, we let a job's state be a tuple of its current stage and the amount of service it has received during this stage. There are m possible stages $\{0, \dots, m-1\}$, with jobs always arriving in stage 0. The state space is $\mathbb{X} = (\{0, \dots, m-1\} \times \mathbb{R}_{\geq 0}) \cup \{\top\}$. A job's trajectory depends on the sequence of stages that are sampled. For instance, if a job goes through stages 0, 3, and 5 and then departs, spending $s_0 \leftarrow S_0$, $s_3 \leftarrow S_3$, and $s_5 \leftarrow S_5$ time in each stage, then the job's trajectory would be

$$X(s) = \begin{cases} (0, s) & \text{if } s < s_0 \\ (3, s - s_0) & \text{if } s \in [s_0, s_0 + s_3) \\ (5, s - (s_0 + s_3)) & \text{if } s \in [s_0 + s_3, s_0 + s_3 + s_5) \\ \top & \text{if } s \geq s_0 + s_3 + s_5. \end{cases}$$

Here the arrival state $X_0 = (0, 0)$ is deterministic, but the dynamics are stochastic: the second element of a job's state increases at rate 1 with stochastic jumps to states of the form $(j, 0)$ or $\top$. The jump intensity is determined by the hazard rate of the current stage's size distribution S_i, and the endpoint of the jump is sampled based on the transition probabilities p_{ij}.

Although we have focused on the case where the service requirement distribution of each stage is unknown, we can use the same technique to model any mixture of known, unknown, and partially known distributions.

3.2 What is Gittins?

The heart of the Gittins policy is the *Gittins rank function*

$$\mathsf{rank} : \mathbb{X} \to \mathbb{R}_{\geq 0}.$$

The Gittins rank function assigns to each state x a rank, or priority, $\mathsf{rank}(x)$. Lower rank is better, meaning Gittins always serves the job of minimal rank. The tiebreaking rule for Gittins is left unspecified, as any tiebreaking rule results in the same mean response time.

The Gittins rank function is

$$\mathsf{rank}(x) = \inf_{\tau > 0} \frac{\mathbf{E}[\tau \mid X(0) = x]}{\mathbf{P}[X(\tau) = \top \mid X(0) = x]},$$

where τ is a past-measurable stopping time. The rank function is based on the following optimization problem. We are given a job in state x, which we serve for as long as we like. We can use any strategy for deciding when to stop, with τ denoting the amount of time we spend serving the job before stopping. Our goal is to *minimize the expected time per job completion* of the process: we spend τ time serving the job and complete $\mathbb{1}(X(\tau) = \top)$ jobs. Because "serve the job until it completes" is a possible strategy, $\mathsf{rank}(x)$ is always at most the expected remaining size of a job starting from state x.

3.3 Gittins is Nearly Optimal in the M/G/k

Recall that Gittins minimizes mean response time in the M/G/1. One could imagine using Gittins in the M/G/k by always serving the k jobs of minimal Gittins rank.

Problem 3.3.1. Bound Gittins's mean response time in the M/G/k.

Progress: 95%. We solve this problem in [37], and we have preliminary work that slightly improves and generalizes our results.

Let S be the overall job size distribution, ρ be the system load, T_k be the response time of Gittins in the M/G/k, and $\overline{F}_e^{-1}$ be the inverse of

$$\overline{F}_e(t) = \frac{1}{\mathbf{E}[S]} \int_t^\infty \mathbf{P}[S > t] \, \mathrm{d}s.$$

We show

$$\mathbf{E}[T_k] - \mathbf{E}[T_1] \le (k-1)\,\mathbf{E}[S]\left(\log \frac{\overline{F}_e^{-1}(1-\rho)}{\mathbf{E}[S]} + 5.133\right). \tag{3.3.1}$$

In particular, if $\mathbf{E}[S^\alpha] < \infty$ for some $\alpha > 1$, then

$$\mathbf{E}[T_k] - \mathbf{E}[T_1] \le (k-1)\,O\left(\log \frac{1}{1-\rho}\right).$$

Prior work implies $\mathbf{E}[T_1] = \omega(\log(1/(1-\rho)))$ if $\mathbf{E}[S^2(\log S)^+] < \infty$ [27], in which case Gittins is optimal in the heavy-traffic M/G/k.

The bound in (3.3.1) depends only on load ρ and the overall job size distribution S. We can tighten the bound further by taking into account the details of the underlying job Markov process $\{X(s)\}_{s \ge 0}$.

Note that the techniques we develop to solve Problem 2.3.1 do not directly apply to analyzing Gittins in the M/G/k because a job's Gittins rank need not be a monotonic function of its age.

3.3.1 Computing Mean Response Time via Relevant Work

The technique underlying our solution to Problem 3.3.1 is a new formula for mean response time based on the Gittins rank function. Let a job's *r-relevant work* be the amount of service it needs to either complete or reach a state of Gittins rank at least r, and let $W(r)$ be the steady-state distribution of the total r-relevant work of all jobs in the system. We show in [37] that the mean number of jobs in the system $\mathbf{E}[N]$ can be computed as

$$\mathbf{E}[N] \int_0^\infty \frac{\mathbf{E}[W(r)]}{r^2} \, \mathrm{d}r,$$

from which mean response time $\mathbf{E}[T]$ quickly follows by Little's law. In addition to being a key step to solving Problem 3.3.1, this formula may be useful in other contexts, as it is not specific to the M/G/k.

3.4 Simple Scheduling with Noisy Size Estimates

One of the main reasons preventing SRPT from being widely adopted in computer systems is that few systems provide perfect job size information. However, the rise of machine learning means that there is no shortage of ways to estimate job sizes, so scenarios like Example 3.1.3 are increasingly common. While Gittins minimizes mean response times in such systems, Gittins may be impractically difficult to compute.

Problem 3.4.1. Design a simple scheduling policy with nearly optimal mean response time in an M/G/1 where the scheduler is provided with noisy job size estimates.

Progress: 20%. We have some preliminary work but do not yet know how successful it will be.

We hope to show that SERPT (Example 2.1.3) is a constant-factor approximation for mean response time under certain conditions on the joint distribution between true and estimated job sizes. It remains to be seen exactly what conditions suffice.

3.4.1 Computing Mean Response Time via Analogy with SRPT

Underlying our plan of attack for Section 3.4 is another new formula for Gittins's mean response time. Specifically, we hope to show that Gittins's mean response time can be written in much the same way as the classic formula for SRPT's mean response time. A classic result of Schrage and Miller [36] shows that in an M/G/1 with a continuous job size distributions S, SRPT's mean response time $\mathbf{E}[T]$ is

$$\mathbf{E}[T] = \int_0^\infty \frac{\mathbf{E}[(\min\{S, r\})^2]}{(1 - \lambda \mathbf{E}[S\mathbb{1}(S \le r)])^2} \, \mathrm{d}\mathbf{P}[S \le r] + \int_0^\infty \frac{\mathbf{P}[S > r]}{1 - \lambda \mathbf{E}[S\mathbb{1}(S \le r)]} \, \mathrm{d}r.$$

Our preliminary work suggests that under certain conditions, we can write Gittins's mean response time using a very similar formula. The main difference is that we substitute most instances of the job size distribution S for a different distribution R.[1] This new distribution R is less than or equal to S under the convex ordering, meaning $\mathbf{E}[f(R)] \le \mathbf{E}[f(S)]$ for all convex R, where $R = S$ in the perfect-information case. If this succeeds, we may be able to use methods developed for SRPT by Wierman et al. [47] to study Gittins in the setting of noisy size estimates.

3.5 Related Work

The Gittins policy was originally introduced to solve the multi-armed bandit problem, and there is a large literature on Gittins in this context. Gittins et al. [15] provides an introduction to the field and references to many other works. In principle, Gittins in the multi-armed bandit setting could be more general than Gittins in the queueing setting we consider, namely minimizing mean response time. However, Gittins in the multi-armed bandit setting is typically studied in discrete time with discrete state spaces, and they typically assume a nonzero discount parameter. The result is that much work on Gittins in queueing [16, 17, 45] are restricted to discrete state spaces. Our

[1] Specifically, we swap every instance of S to R except for the second moment in the numerator of the first term, which changes in a more complicated way.

presentation of Gittins uses general continuous-time Markov processes to allow for continuous state spaces.

There are some prior results showing Gittins is nearly optimal in multiserver systems, but they are for job models far less general than ours. The most general such results are the following:

- Glazebrook and Niño-Mora [17] study the multiclass M/M/k with Bernoulli feedback. This can be viewed as a special case of the multistage job model (Example 3.1.4) in which all stage size distributions are exponential.
- Because SRPT is the special case of Gittins with perfect job size information, our results on SRPT in the M/G/k (Problem 2.3.1) are an instance of Gittins being nearly optimal in the M/G/k.
- Glazebrook [16] study the *nonpreemptive* multiclass M/G/k with Bernoulli feedback in discrete time. The results apply only under an assumption on each class's job size distribution that imply they are light-tailed.

The partial information scenario with noisy size estimates has attracted attention in recent years, with several works analyzing and comparing different scheduling heuristics [12, 13, 28]. These heuristics are substantially simpler than Gittins, but whether they attain comparable mean response time is an open question that we hope to address (Problem 3.4.1).

4 Thesis Timeline

I plan to graduate in June 2022. Here is how I will spend my time from now until graduation.

- *Present–February 2021:* complete ongoing work on the following problems:
  - Systems case studies (Problems 2.2.1–2.2.3).
  - Tail optimality of Gittins (Problem 2.2.5).
  - Analyzing preemptive priority with preemption costs (Problem 2.3.4).
- *March 2021–December 2021:* work on the following more ambitious open problems:
  - Analyze general SOAP policies with preemption costs (Problem 2.3.4).
  - Design a simple near-optimal policy for jobs with noisy size estimates (Problem 3.4.1).
  - Queueing versions of the stochastic multi-armed bandit problem (not in this thesis).
- *January 2022–June 2022:* write the thesis (Problem 1.2.1), which will involve tying up loose ends on the following problems:
  - Improving our M/G/k mean response time bounds for monotonic SOAP policies (Problem 2.3.1) and Gittins (Problem 3.3.1).
  - Improving our analysis of Gittins in heavy traffic (Problem 2.2.6).

Bibliography

[1] Samuli Aalto, Urtzi Ayesta, Sem Borst, Vishal Misra, and Rudesindo Núñez-Queija. 2007. Beyond Processor Sharing. *ACM SIGMETRICS Performance Evaluation Review* 34, 4 (March 2007), 36–43.

[2] Samuli Aalto, Urtzi Ayesta, and Rhonda Righter. 2009. On the Gittins Index in the M/G/1 Queue. *Queueing Systems* 63, 1-4 (Dec. 2009), 437–458.

[3] Samuli Aalto, Urtzi Ayesta, and Rhonda Righter. 2011. Properties of the Gittins Index with Application to Optimal Scheduling. *Probability in the Engineering and Informational Sciences* 25, 3 (July 2011), 269–288.

[4] Josh Aas. 2005. Understanding the Linux 2.6.8.1 CPU Scheduler.

[5] Remzi H. Arpaci-Dusseau and Andrea C. Arpaci-Dusseau. 2018. *Operating Systems: Three Easy Pieces* (1.00 ed.). Arpaci-Dusseau Books.

[6] Konstantin Avrachenkov, Patrick Brown, and Natalia Osipova. 2009. Optimal Choice of Threshold in Two Level Processor Sharing. *Annals of Operations Research* 170, 1 (Sept. 2009), 21–39.

[7] Nicholas H. Bingham, Charles M. Goldie, and Jef L. Teugels. 1987. *Regular Variation.* Number 27 in Encyclopedia of Mathematics and Its Applications. Cambridge University Press, Cambridge [Cambridgeshire] ; New York.

[8] Sem Borst, Rudesindo Núñez-Queija, and Bert Zwart. 2006. Sojourn Time Asymptotics in Processor-Sharing Queues. *Queueing Systems* 53, 1-2 (June 2006), 31–51.

[9] Jacqueline Boyer, Fabrice Guillemin, Philippe Robert, and Bert Zwart. 2003. Heavy Tailed M/G/1-PS Queues with Impatience and Admission Control in Packet Networks. In *Proceedings of IEEE INFOCOM 2003. 22nd Annual Joint Conference of the IEEE Computer and Communications Societies*, Vol. 1. IEEE, San Francisco, CA, 186–195.

[10] Francesco Capozzi, Giuseppe Piro, L. Alfredo Grieco, Gennaro Boggia, and Pietro Camarda. 2012. Downlink Packet Scheduling in LTE Cellular Networks: Key Design Issues and a Survey. *IEEE Communications Surveys & Tutorials* 15, 2 (July 2012), 678–700.

[11] Daren B. H. Cline. 1994. Intermediate Regular and Π Variation. *Proceedings of the London Mathematical Society* s3-68, 3 (May 1994), 594–616.

[12] Matteo Dell'Amico. 2019. Scheduling with Inexact Job Sizes: The Merits of Shortest Processing Time First. *arXiv* (July 2019), 8. arXiv:1907.04824

[13] Seyed Emadi, Rouba Ibrahim, and Saravanan Kesavan. 2019. Can "Very Noisy" Information Go a Long Way? An Exploratory Analysis of Personalized Scheduling in Service Systems. *Working paper* (Jan. 2019), 40.

[14] S. W. Fuhrmann and Robert B. Cooper. 1985. Stochastic Decompositions in the M/G/1 Queue with Generalized Vacations. *Operations Research* 33, 5 (Oct. 1985), 1117–1129.

[15] John C. Gittins, Richard Weber, and Kevin D. Glazebrook. 2011. *Multi-Armed Bandit Allocation Indices* (second ed.). Wiley, Hoboken, NJ.

[16] K. D. Glazebrook. 2003. An Analysis of Klimov's Problem with Parallel Servers. *Mathematical Methods of Operations Research* 58, 1 (Sept. 2003), 1–28.

[17] Kevin D. Glazebrook and José Niño-Mora. 2001. Parallel Scheduling of Multiclass M/M/m Queues: Approximate and Heavy-Traffic Optimization of Achievable Performance. *Operations Research* 49, 4 (Aug. 2001), 609–623.

[18] Carmelita Goerg. 1986. Evaluation of the Optimal SRPT Strategy with Overhead. *IEEE Transactions on Communications* 34, 4 (April 1986), 338–344.

[19] Isaac Grosof, Ziv Scully, and Mor Harchol-Balter. 2019. SRPT for Multiserver Systems. *ACM SIGMETRICS Performance Evaluation Review* 46, 2 (Jan. 2019), 9–11.

[20] Mor Harchol-Balter. 2013. *Performance Modeling and Design of Computer Systems: Queueing Theory in Action*. Cambridge University Press, Cambridge.

[21] Mor Harchol-Balter, Bianca Schroeder, Nikhil Bansal, and Mukesh Agrawal. 2003. Size-Based Scheduling to Improve Web Performance. *ACM Transactions on Computer Systems* 21, 2 (May 2003), 207–233.

[22] Esa Hyytiä, Samuli Aalto, and Aleksi Penttinen. 2012. Minimizing Slowdown in Heterogeneous Size-Aware Dispatching Systems. *ACM SIGMETRICS Performance Evaluation Review* 40, 1 (June 2012), 29–40.

[23] Bart Kamphorst and Bert Zwart. 2020. Heavy-Traffic Analysis of Sojourn Time under the Foreground–Background Scheduling Policy. *Stochastic Systems* 10, 1 (March 2020), 1–28.

[24] David G. Kendall. 1953. Stochastic Processes Occurring in the Theory of Queues and Their Analysis by the Method of the Imbedded Markov Chain. *The Annals of Mathematical Statistics* 24, 3 (1953), 338–354.

[25] Leonard Kleinrock. 1967. Time-Shared Systems: A Theoretical Treatment. *J. ACM* 14, 2 (April 1967), 242–261.

[26] Leonard Kleinrock. 1976. *Queueing Systems, Volume 2: Computer Applications*. Wiley, New York, NY.

[27] Minghong Lin, Adam Wierman, and Bert Zwart. 2011. Heavy-Traffic Analysis of Mean Response Time under Shortest Remaining Processing Time. *Performance Evaluation* 68, 10 (Oct. 2011), 955–966.

[28] Michael Mitzenmacher. 2020. Scheduling with Predictions and the Price of Misprediction. In *11th Innovations in Theoretical Computer Science Conference (ITCS 2020) (Leibniz International Proceedings in Informatics (LIPIcs))*, Vol. 151. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, Dagstuhl, Germany, Article 14, 18 pages.

[29] Behnam Montazeri, Yilong Li, Mohammad Alizadeh, and John Ousterhout. 2018. Homa: A Receiver-Driven Low-Latency Transport Protocol Using Network Priorities. In *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication (SIGCOMM 2018)*. ACM, Budapest, Hungary, 221–235.

[30] Rudesindo Núñez-Queija. 2002. Queues with Equally Heavy Sojourn Time and Service Requirement Distributions. *Annals of Operations Research* 113, 1/4 (July 2002), 101–117.

[31] Misja Nuyens, Adam Wierman, and Bert Zwart. 2008. Preventing Large Sojourn Times Using SMART Scheduling. *Operations Research* 56, 1 (Feb. 2008), 88–101.

[32] Natalia Osipova, Urtzi Ayesta, and Konstantin Avrachenkov. 2009. Optimal Policy for Multi-Class Scheduling in a Single Server Queue. In *2009 21st International Teletraffic Congress*. IEEE, Paris, France, 1–8.

[33] Michael Pinedo. 2016. *Scheduling: Theory, Algorithms, and Systems* (fifth ed.). Springer, Cham, Switzerland.

[34] Linus E. Schrage. 1967. The Queue M/G/1 with Feedback to Lower Priority Queues. *Management Science* 13, 7 (March 1967), 466–474.

[35] Linus E. Schrage. 1968. A Proof of the Optimality of the Shortest Remaining Processing Time Discipline. *Operations Research* 16, 3 (June 1968), 687–690.

[36] Linus E. Schrage and Louis W. Miller. 1966. The Queue M/G/1 with the Shortest Remaining Processing Time Discipline. *Operations Research* 14, 4 (Aug. 1966), 670–684.

[37] Ziv Scully, Isaac Grosof, and Mor Harchol-Balter. 2020. The Gittins Policy Is Nearly Optimal in the M/G/k under Extremely General Conditions. *Proceedings of the ACM on Measurement and Analysis of Computing Systems* 4, 3, Article 43 (Dec. 2020), 29 pages.

[38] Ziv Scully, Isaac Grosof, and Mor Harchol-Balter. 2020. Optimal Multiserver Scheduling with Unknown Job Sizes in Heavy Traffic. *Performance Evaluation* (Oct. 2020), 31.

[39] Ziv Scully and Mor Harchol-Balter. 2018. SOAP Bubbles: Robust Scheduling under Adversarial Noise. In *56th Annual Allerton Conference on Communication, Control, and Computing*. IEEE, Monticello, IL, 144–154.

[40] Ziv Scully, Mor Harchol-Balter, and Alan Scheller-Wolf. 2018. SOAP: One Clean Analysis of All Age-Based Scheduling Policies. *Proceedings of the ACM on Measurement and Analysis of Computing Systems* 2, 1, Article 16 (April 2018), 30 pages.

[41] Ziv Scully, Mor Harchol-Balter, and Alan Scheller-Wolf. 2020. Simple Near-Optimal Scheduling for the M/G/1. *Proceedings of the ACM on Measurement and Analysis of Computing Systems* 4, 1, Article 11 (May 2020), 29 pages.

[42] Ziv Scully, Lucas van Kreveld, Onno J. Boxma, Jan-Pieter Dorsman, and Adam Wierman. 2020. Characterizing Policies with Optimal Response Time Tails under Heavy-Tailed Job Sizes. *Proceedings of the ACM on Measurement and Analysis of Computing Systems* 4, 2, Article 30 (June 2020), 33 pages.

[43] L. Takács. 1963. Delay Distributions for One Line with Poisson Input, General Holding Times, and Various Orders of Service. *Bell System Technical Journal* 42, 2 (March 1963), 487–503.

[44] Rocco van Vreumingen. 2019. *Queueing Systems with Non-Standard Service Policies and Server Vacations*. Master's. University of Amsterdam, Amsterdam, the Netherlands.

[45] Peter Whittle. 2005. Tax Problems in the Undiscounted Case. *Journal of Applied Probability* 42, 3 (Sept. 2005), 754–765.

[46] Adam Wierman. 2007. *Scheduling for Today's Computer Systems: Bridging Theory and Practice*. Ph.D. Dissertation. Carnegie Mellon University, Pittsburgh, PA.

[47] Adam Wierman, Mor Harchol-Balter, and Takayuki Osogami. 2005. Nearly Insensitive Bounds on SMART Scheduling. In *Proceedings of the 2005 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems (SIGMETRICS 2005)*. ACM, Banff, Alberta, Canada, 205.

[48] Adam Wierman and Misja Nuyens. 2008. Scheduling despite Inexact Job-Size Information. In *Proceedings of the 2008 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems (SIGMETRICS 2008)*. ACM, Annapolis, MD, USA, 25–36.

[49] Zhi-Hui Zhan, Xiao-Fang Liu, Yue-Jiao Gong, Jun Zhang, Henry Shu-Hung Chung, and Yun Li. 2015. Cloud Computing Resource Scheduling and a Survey of Its Evolutionary Approaches. *Comput. Surveys* 47, 4, Article 63 (July 2015), 33 pages.