A New Toolbox for Scheduling Theory

Ziv Scully
PhD affiliation: Carnegie Mellon University, Computer Science Department
Current affiliation: Cornell University, School of Operations Research and Information Engineering

ABSTRACT

Queueing delays are ubiquitous in many domains, including computer systems, service systems, communication networks, supply chains, and transportation. Queueing and scheduling theory provide a rigorous basis for understanding how to reduce delays with scheduling, including evaluating policy performance and guiding policy design. Unfortunately, state-of-the-art theory fails to address many practical concerns. For example, scheduling theory seldom treats nontrivial preemption limitations, and there is very little theory for scheduling in multiserver queues.

My thesis presents two new, broadly applicable tools that greatly expand the reach of scheduling theory, using each to solve multiple open problems. The first tool, called "SOAP", is a new unifying theory of scheduling in single-server queues, specifically the M/G/1 model. SOAP characterizes the delay distribution of a broad space of policies, most of which have never been analyzed before. Such policies include the Gittins index policy, which minimizes mean delay in low-information settings, and many policies with preemption limitations. The second tool, called "WINE", is a new queueing identity that complements Little's law. WINE enables a new method of analyzing complex queueing systems by relating them to simpler systems. This results in the first delay bounds for Shortest Remaining Processing Time (SRPT) and the Gittins policy in multiserver queues, specifically the M/G/k model.

This abstract gives a brief overview of my thesis, describing what the SOAP and WINE tools do, the key ideas underlying them, and the open problems they help solve.

1. INTRODUCTION

How can system designers reduce the queueing delay jobs experience? My thesis [14] focuses on one of the main tools for doing so: *scheduling*, namely altering the strategy by which we allocate resources to clients. Scheduling is appealing in that it is virtually free, and it can be done with the resources and know-how one already has.

Given the potential benefits of smart scheduling, it should come as no surprise that queueing theorists have studied scheduling for more than half a century. Why, then, do we need more scheduling theory? The issue is—and has always been, and will always be—that today's scheduling theory does not adequately match scheduling practice. Some examples of areas where scheduling theory is lacking are:

- Scheduling under *uncertainty*, particularly regarding how much service a job needs.
- Scheduling with *multiple servers*.
- Scheduling with practical *preemption constraints*, meaning restrictions on the server's ability to switch from serving one job to serving another.

We would like to develop scheduling theory for these and other practical concerns. However, each of these concerns makes theoretically analyzing and optimizing scheduling policies, an already difficult endeavor, even more complicated.

My thesis [14] develops *two new queueing-theoretic tools* that enable the study of concerns like uncertainty, multiple servers, and preemption constrains. The two tools are called *SOAP* (§2) and *WINE* (§3). My thesis applies SOAP and WINE to prove several previously intractable results.

2. SOAP: UNIFYING THEORY OF SINGLE-SERVER SCHEDULING

One of the main aims of scheduling theory is to guide system design by analyzing the performance impact of using a given scheduling policy. For example, consider the Shortest Remaining Processing Time (SRPT) policy. In single-server queueing systems, SRPT, which always serves whichever job has the least remaining work, is known to minimize mean *response time*, namely the mean time jobs spend in the system [12].

While the above result implies SRPT is good for reducing mean response time, there are other questions we might want to answer before deploying SRPT in practice.

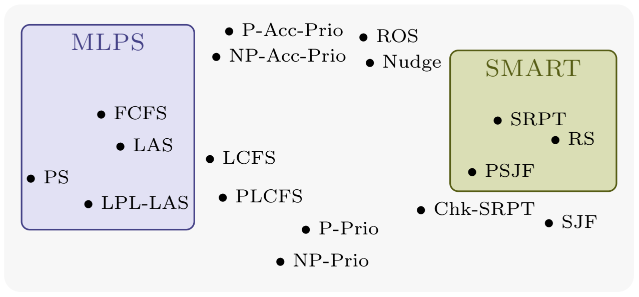
- How much better is SRPT's mean response time than simpler alternatives, such as First-Come, First-Served (FCFS)? Is the benefit worth the extra complexity?
- How well does SRPT perform on response time metrics beyond the mean? Does improving mean performance come at the price of degrading tail performance?
- How are SRPT's response time benefits distributed across different jobs? Is SRPT unfair to long jobs?

Fortunately, we can answer these and other questions about SRPT using its *queueing theoretic analysis*. This analysis, due to Schrage and Miller [13], exactly characterizes SRPT's response time distribution in the M/G/1 queueing model. Queueing theorists use Schrage and Miller's analysis as a foundation for answering many questions about SRPT, including those above. See my thesis [14, Chs. 2 and 3] for further discussion.

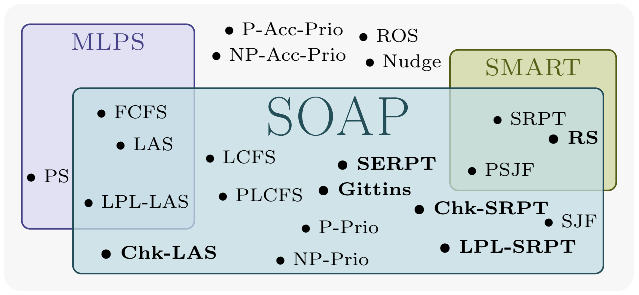
2.1 Problem: Can Analyze Only a Limited Set of Scheduling Policies

Since the 1960s, queueing theorists have analyzed many scheduling policies in the M/G/1. See Figure 2.1(a) and my thesis [14, Ch. 2] for an inexhaustive list. But for the most part, these analyses proceed one-by-one, with a paper analyzing a single policy or handful of related policies. Each new policy seems to require a new analysis.

Two notable exceptions to this state of affairs are analyses of two *classes* of policies, Multi-Level Processor Sharing (MLPS) [7] and SMAll Response Times (SMART) [15]. In both cases, a single analysis applies to all policies of a particular form. However, both MLPS and SMART are relatively

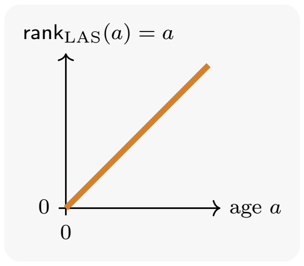


(a) Scheduling policies analyzed in the M/G/1 without SOAP.

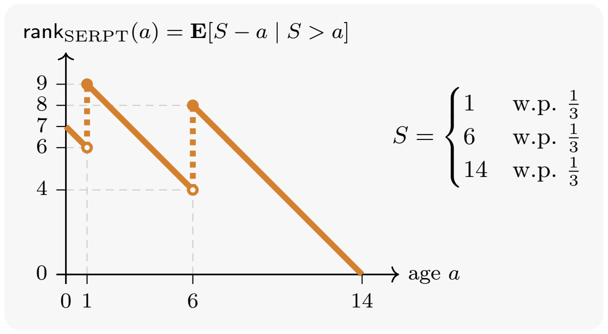


(b) Scheduling policies analyzed in the M/G/1, including those analyzed with SOAP, which are highlighted in **bold**. This includes two policies, Chk-SRPT and RS, for which mean response time, but not distribution of response time, was previously known.

Figure 2.1. SOAP expands the set of policies we know how to analyze in the M/G/1, a single-server queueing model. (a) Prior to SOAP, most policies were analyzed one-by-one, with a few relatively small classes of policies analyzed. (b) SOAP unifies and generalizes the state of the art with a *single universal analysis* for a broad class of policies, subsuming much of what was already known while also analyzing many policies for the first time.



(a) Rank function of Least Attained Service (LAS). The function is monotonic, so LAS was tractable to analyze prior to SOAP [11].



(b) Rank function of SERPT for an example service time distribution S. The function is *nonmonotonic*, so SERPT was *intractable* to analyze prior to SOAP.

Figure 2.2. Two examples of rank functions, each describing a SOAP policy. A rank function determines a job's priority (lower number is better) as a function of its age (a.k.a. attained service). For example, the LAS policy, shown in (a), always serves the job that has been served the least so far.

limited in terms of the policies they cover.

Why is it a problem that the set of scheduling policies we can analyze is limited? Because many important scheduling concerns lie outside of what we can currently analyze. As just one example, consider scheduling under *uncertainty* about jobs' service times. We cannot use SRPT, because we cannot determine each job's remaining work. Instead, we might prioritize jobs using *expected* remaining work, yielding a policy called *SERPT* (E for "expected").

Does SERPT have good mean response time? SERPT is intuitively appealing, but it turns out there is another more complicated policy, called the *Gittins* policy [1, 2], that minimizes mean response time when service times are uncertain. So SERPT certainly has worse mean response time than Gittins. But neither SERPT nor Gittins has been analyzed, so we do not know how big the performance gap between them is. Is Gittins necessary for good mean response time under uncertainty, or does the simpler SERPT suffice?

2.2 Key Idea: Unifying Language for Policies Enables a Universal Analysis

Despite the fact that most prior work analyzes policies one by one, there are common ideas that appear across multiple analyses. For example, part of Harchol-Balter [5], an introductory queueing text, is devoted to analyzing ten different scheduling policies in the M/G/1. But *nine out of ten* of the analyses follow a similar overall strategy, albeit with different details. Can we unify the analyses of these nine policies? If so, can that help us analyze even more policies?

SOAP, which stands for *Schedule Ordered by Age-based Priority*, answers both questions affirmatively. SOAP has two parts:

- *SOAP policies:* a broad class of scheduling policies described in a single unifying language. The class includes many policies which have never been analyzed before.
- *SOAP analysis:* a single analysis that applies to all SOAP policies. The analysis characterizes response time in the M/G/1, yielding exact formulas for the mean and Laplace-Stieltjes transform of response time.

SOAP thus unifies and generalizes prior analyses, as shown in Figure 2.1(b).

The key idea behind SOAP is the unifying language it uses to represent scheduling policies: *rank functions.* A rank function gives each job a *rank*, i.e. priority, as a function of its *age*, i.e. how much service it has received so far. Figure 2.2 illustrates two examples. The SOAP class contains all policies that can be represented by a rank function.

The flexibility of rank functions makes the class of SOAP policies very broad. SOAP includes both classic policies that were analyzed decades ago, such as FCFS and Least Attained Service (LAS) (Fig. 2.2(a)), and policies that have never been analyzed before, such as SERPT (Fig. 2.2(b)) and Gittins. Part of the breadth of SOAP comes from the fact that rank functions can also take as input certain "static" job characteristics, such as service time. For instance, we can represent SRPT by the rank function $\mathsf{rank}_{\mathrm{SRPT}}(s, a) = s - a$: a job's priority is its remaining work, which is its service time s minus its age a. My thesis [14, Chs. 3 and 6] contains numerous examples of SOAP policies and their rank functions.

Having defined SOAP policies, the question of analyzing SOAP policies in the M/G/1 remains. This boils down to analyzing a policy's response time in terms of its rank function. The primary challenge turns out to be handling *nonmonotonic* rank functions. In fact, with just one exception [3], all previously analyzed SOAP policies have monotonic rank functions. But SERPT, Gittins, and many other policies have nonmonotonic rank functions (Fig. 2.2(b)). My thesis [14, Chs. 3 and 7] describes the obstacles nonmonotonicity poses and how the SOAP analysis overcomes them.

2.3 Impact: Broad Class of Policies Analyzed for the First Time in the M/G/1

The end result of the SOAP analysis is an exact characterization of the response time distribution of any SOAP policy in the M/G/1. This yields the first analysis of SERPT and Gittins, both of which can have nonmonotonic rank functions, as well as many other policies.

The SOAP analysis alone, with no additional theory, is already valuable for guiding scheduling design. My thesis numerically applies the SOAP analysis to compare SERPT and Gittins [14, Ch. 10]. One finding is that Gittins's mean response time can often be nearly matched by SERPT or another simple alternative. My thesis also applies SOAP to gain insight into scheduling under two types of preemption limitations [14, Ch. 9].

In addition to being useful numerically, the SOAP analysis is also useful as a foundation for further theory. For example, motivated by the numerical observation that SERPT nearly matches Gittins's mean response time in many examples, we might wonder whether we can prove that SERPT is always in some sense near-optimal for mean response time. My thesis proves such a guarantee for a slight modification of SERPT [14, Ch. 11]. Going beyond means, my thesis also theoretically characterizes the *asymptotic tail behavior* of SERPT's and Gittins's response time distributions [14, Ch. 13].

3. WINE: NEW QUEUEING IDENTITY TO AID MULTISERVER ANALYSIS

Multiserver queueing systems are ubiquitous in practice. Virtually all computer systems today have multiple processing units, from smartphones with multiple cores to datacenters with thousands of machines. Unfortunately, while queueing theorists have studied scheduling in single-server systems for decades, there is currently very little queueing theory that can help us analyze or optimize scheduling policies in multiserver systems.

For concreteness, consider the problem of minimizing mean response time in a system with known service times. In the M/G/1, it has long been known that SRPT is optimal [12]. But the general idea of serving jobs that will complete soon seems wise even we have multiple servers. While it turns out perfect optimality is intractable with multiple servers [8], we might still ask whether SRPT is in some sense near-optimal in multiserver systems. We focus on the M/G/k, an analogue of the M/G/1 in which $k \geq 2$ servers are connected to a single central queue.

(Hereafter, we append a "-1" or "-k" to a policy's name when discussing its M/G/1 or M/G/k version, respectively. For instance, SRPT-1 always serves the single job of least remaining work, while SRPT-k always serves the k jobs with the k least amounts of remaining work, or all jobs if there are fewer than k.)

3.1 Problem: M/G/1 Analysis Techniques Fail in the M/G/*k*

Analyzing response time of the M/G/k is famously intractable, even under the seemingly simple FCFS-k, and even if we are willing to settle for bounds or approximations. See Gupta et al. [4], Kingman [6], Li and Goldberg [9], and my thesis [14, Ch. 2] for further discussion.

Given the difficulty of analyzing FCFS-k, analyzing SRPT-k seems out of reach. Gittins-k seems harder still, seeing as Gittins-1 was only analyzed for the first time using SOAP. But this prompts a question: can we generalize SOAP from the M/G/1 to cover the M/G/k as well?

Unfortunately, the M/G/1 analysis technique underlying SOAP seems unlikely to work in general in the M/G/k. One of the key properties of the M/G/1 that makes it tractable to analyze is that the single server acts as a "choke point" through which all jobs must pass. This makes it easy to determine how long one job A spends waiting behind another job B, which is a key step of the SOAP analysis.

The story is more complicated in the M/G/k. With multiple servers, there is no longer a single "choke point" through which all jobs pass. This makes it much harder to determine how long one job A spends waiting behind another job B. Even if job A has worse rank than job B, it might happen that both A and B get to enter service at the same time. This is just one of several complications that make a SOAP-style analysis of the M/G/k difficult.

3.2 Key Idea: Relate Response Time to Work, a Much Simpler Quantity

Given that existing M/G/1 techniques do not work well in the M/G/k, we need a new approach. One source of inspiration comes from *Little's law* [10], a queueing identity which relates a system's mean response time $\mathbf{E}[T]$ to the mean number of jobs $\mathbf{E}[N]$. While there are few problems that Little's law solves by itself, it is a ubiquitous tool, in large part because it applies to essentially any queueing system. Are there other similarly general identities that could help us analyze scheduling in the M/G/k?

WINE, which stands for *Work Integral Number Equality*, answers this question affirmatively. WINE is a new queueing identity that relates a system's number of jobs N to a quantity called *r-work*, denoted $W(r)$. Like Little's law, WINE applies to essentially any queueing system, including M/G/k. By combining WINE with Little's law and one more ingredient, we obtain mean response time bounds on SRPT-k and Gittins-k (Fig. 3.1).

Before explaining how WINE helps us analyze scheduling in the M/G/k, let us pin down exactly what WINE and r-work are. We focus on SRPT for simplicity, but the same story works for Gittins, albeit with more complicated details.

For any rank $r \geq 0$, r-work $W(r)$ is, roughly speaking, "work with rank better than r". Under SRPT, this becomes

$$\text{“SRPT-flavored” } r\text{-work:} \quad W(r) = \sum_{j=1}^{N} X_j \, \mathbb{1}(X_j < r),$$

where X_j is the remaining work of job j. That is, we sum up the remaining work X_j of all jobs j whose rank—which,

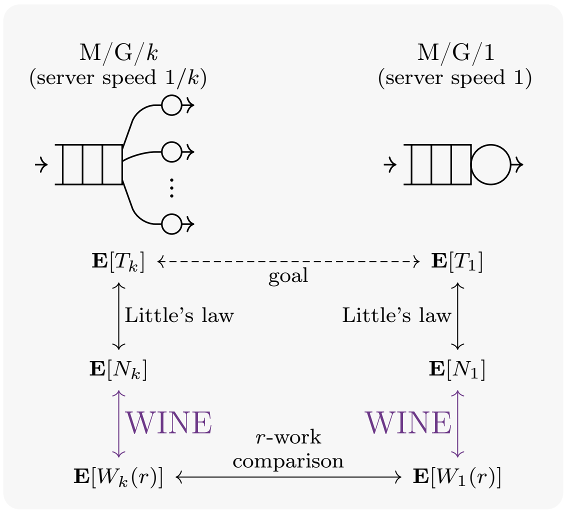


Figure 3.1. Using WINE to analyze the mean response time of SRPT-k and Gittins-k. Both analyses use the same approach: combine WINE and Little's Law to both the M/G/k and M/G/1 to relate response time T to r-work $W(r)$, then use work decomposition to relate r-work in the M/G/k to r-work in the M/G/1.

under SRPT, is also X_j—is better than r.

How is the number of jobs N related to r-work $W(r)$? For any single rank r, there is no relationship that holds between N and $W(r)$. But it turns out that if we take an *integral* of $W(r)$ over all ranks $r \geq 0$, we obtain the number of jobs. One can verify that for any values of $X_1, \dots, X_N$, we have

$$\text{“SRPT-flavored” WINE:} \qquad N = \int_0^\infty \frac{W(r)}{r^2}\,\mathrm{d}r.$$

My thesis [14, Chs. 3 and 15] geometrically interprets this equation and generalizes it to “Gittins-flavored” WINE.

Why is WINE helpful? As outlined in Figure 3.1, combining WINE with Little's law reduces the problem of analyzing mean response time to that of analyzing mean r-work. But is analyzing mean r-work any easier? While an exact analysis is intractable, remarkably, it is possible to prove a bound. The key idea is to bound the r-work in an M/G/k in terms of the r-work in an M/G/1, scaling server speeds to give both systems the same total service capacity. For instance, in Figure 3.1, we compare an M/G/k with servers of speed $1/k$ to an M/G/1 with server speed 1, giving both systems total service capacity 1. This results in a bound on SRPT-k's mean response time in terms of that of SRPT-1.

A question remains: how is it possible to compare M/G/k r-work to M/G/1 r-work, and why is that easier than directly comparing response times? The basic idea is that we can reason about r-work as a continuous quantity, showing that r-work “drains” at similar rates in the M/G/k and M/G/1 [14, Chs. 8 and 17]. Directly comparing response times involves reasoning discretely about jobs, which seems more difficult.

3.3 Impact: First Response Time Bounds for SRPT-*k*, Gittins-*k*, and More

The main motivation for WINE was analyzing SRPT-k and Gittins-k. Following the outline in Figure 3.1, my thesis proves a mean response time bound for SRPT-k and Gittins-k [14, Ch. 17]. The bound is tight enough to imply *heavy traffic optimality*. That is, as the job arrival rate increases, the ratio between SRPT-k's mean response time and the minimum achievable mean response time approaches 1, and similarly for Gittins-k under uncertain service times.

In addition to the multiserver results, it turns out WINE is also useful for proving theorems about single-server scheduling. For example, my thesis uses a cocktail of SOAP and WINE to prove that in the M/G/1, variations of SRPT that have access to only noisy service time estimates can still achieve mean response time close to true SRPT [14, Ch. 12].

References

[1] 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.

[2] John C. Gittins, Kevin D. Glazebrook, and Richard R. Weber. 2011. *Multi-Armed Bandit Allocation Indices* (second ed.). Wiley, Chichester, UK.

[3] Carmelita Goerg. 1990. Further Results on a New Combined Strategy Based on the SRPT-principle. *IEEE Transactions on Communications* 38, 5 (May 1990), 568–570.

[4] Varun Gupta, Mor Harchol-Balter, J. G. Dai, and Bert Zwart. 2010. On the Inapproximability of M/G/K: Why Two Moments of Job Size Distribution Are Not Enough. *Queueing Systems* 64, 1 (Jan. 2010), 5–48.

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

[6] John F. C. Kingman. 2009. The First Erlang Century—and the Next. *Queueing Systems* 63, 1 (Nov. 2009), 3.

[7] Leonard Kleinrock and Richard R. Muntz. 1972. Processor Sharing Queueing Models of Mixed Scheduling Disciplines for Time Shared System. *J. ACM* 19, 3 (July 1972), 464–482.

[8] Stefano Leonardi and Danny Raz. 2007. Approximating Total Flow Time on Parallel Machines. *J. Comput. System Sci.* 73, 6 (Sept. 2007), 875–891.

[9] Yuan Li and David A. Goldberg. 2017. Simple and Explicit Bounds for Multi-Server Queues with Universal $1/(1-\rho)$ Scaling. arXiv:1706.04628.

[10] John D. C. Little. 2011. Little's Law as Viewed on Its 50th Anniversary. *Operations Research* 59, 3 (June 2011), 536–549.

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

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

[13] 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.

[14] Ziv Scully. 2022. *A New Toolbox for Scheduling Theory.* Ph. D. Dissertation. Carnegie Mellon University, Pittsburgh, PA.

[15] Adam Wierman, Mor Harchol-Balter, and Takayuki Osogami. 2005. Nearly Insensitive Bounds on SMART Scheduling. *ACM SIGMETRICS Performance Evaluation Review* 33, 1 (June 2005), 205–216.

Author Biography

Ziv Scully completed his PhD in Computer Science at Carnegie Mellon University in 2022, advised by Prof. Mor Harchol-Balter and Prof. Guy Blelloch. He will join Cornell University's School of Operations Research and Information Engineering (ORIE) as an assistant professor in 2023.

Ziv researches the theory of decision making under uncertainty, resource allocation, and performance evaluation. His particular focus has been analyzing and optimizing computer systems and algorithms from a stochastic perspective, studying topics including job scheduling, load balancing, and stochastic combinatorial optimization. Ziv's work has been recognized by awards from ACM SIGMETRICS, IFIP PERFORMANCE, and INFORMS, including winning the 2022 George Nicholson Student Paper Competition.