Research Statement

Ziv Scully

We need a better theoretical understanding of scheduling for today's computer systems. Today's computer systems are complicated: they have many servers, operate under uncertainty, are subject to a variety of overheads and constraints, and must optimize difficult metrics like tail latency. Each of these factors makes scheduling more difficult, increasing the need for guidance from theory.

While traditional theoretical computer science studies scheduling in worst-case models, for today's large-scale systems, I believe *stochastic* tools, namely *queueing theory*, are more appropriate. Unfortunately, queueing theory cannot yet handle today's systems. Scheduling on multiple servers is nearly absent from queueing theory, and theory for scheduling under uncertainty and with overheads is also lacking.

I aim to develop scheduling theory for today's computer systems. My research spans four main themes: *multiserver systems* (§1), *uncertainty* (§2), *overheads and constraints* (§3), and *tail metrics* (§4). As respective examples, my research provides the first queueing theoretic analyses of multiserver SRPT, the Gittins policy, scheduling under the constraints of network switch hardware, and a new scheduling policy that strictly improves upon FCFS's tail latency. Underlying these "firsts" are *new frontiers in queueing theory* (§5) that my collaborators and I have opened. In the future, I plan to study systems like machine learning training clusters and extend my theoretical insights to domains like reinforcement learning (§6).

1 Multiserver Systems

While there is a large body of queueing theory studying scheduling in single-server[1] systems, there is almost none for multiserver systems. This is unfortunate, because multiserver systems are ubiquitous at all scales, from the multiple cores in a mobile device to the thousands of machines in a datacenter. *How should we schedule in multiserver systems?*

1.1 SRPT in Multiserver Systems For single-server systems, it is well known that the scheduling policy that minimizes mean response time[2] is *Shortest Remaining Processing Time* (SRPT). SRPT always serves whichever job has the least remaining size,[3] preempting the job in service if a smaller one arrives. It is well known that SRPT minimizes mean response time in single-server systems. Unfortunately, it is known that in the worst case, SRPT can perform poorly in multiserver systems: it has unbounded competitive ratio for mean response time [10]. However, SRPT has never been analyzed in the more realistic *stochastic* case, in which arrival times and job sizes are random variables drawn from distributions. *Does SRPT perform well in stochastic multiserver systems?*

We derive the *first response time bounds on multiserver SRPT* [4]. To clarify, multiserver SRPT always serves as many jobs as possible, prioritizing jobs from least to greatest remaining processing time. The bound we prove is a stochastic dominance bound, so it implies bounds on the mean, variance, and higher moments of response time. We use this bound to prove that multiserver SRPT has optimal mean response time in heavy traffic, meaning that as the system's arrival rate increases, the ratio between SRPT's mean response time and the optimal policy's approaches 1.

Our analysis uses a novel combination of stochastic ideas from queueing theory and worst-case ideas from theoretical computer science. This work won the PERFORMANCE 2018 Best Student Paper Award.

1.2 Dispatching to SRPT Queues The above analysis of multiserver SRPT is in a *central-queue* setting, in which all jobs reside in a single queue. However, many systems are better described as *immediate-dispatch*, where each server has its own queue, and jobs must be assigned to one server's queue upon arrival.

[1] The word "server" has a very general meaning in scheduling theory: it is any entity that completes work. Exactly what a sever represents varies. It can be a single core, a single machine, or even a cluster of machines.

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

[3] A job's *size* is its processing time, and a job's *remaining size* is its size minus the amount of time it has been served so far.

We derive the *first response time bounds on policies for dispatching to SRPT queues* [5]. The key idea is to make sure that each queue receives amount of work from small jobs, medium jobs, and large jobs. We formalize this as a set of constraints we call “guardrails”, which are easily satisfied by simple dispatching policies. We prove a stochastic response time bound on any system that pairs a dispatching policy that obeys guardrails with SRPT servers. Our bound implies that guardrails ensure optimal mean response time in heavy traffic. This work won the SIGMETRICS 2019 Best Student Paper Award and was featured as a mini-plenary at STOC 2021 TheoryFest.

1.3 Beyond SRPT Our group has been at the forefront of analyzing scheduling in multiserver systems. Our paper on multiserver SRPT uses the same technique to analyze other size-based scheduling policies [4]. In later work, we develop new techniques for analyzing multiserver policies for unknown job sizes [15–17].

2 Uncertainty

Scheduling using SRPT requires knowing each job’s exact remaining size, but determining exact job sizes is often impossible or impractical. There are many different types of job size uncertainty. The scheduler may have no size information at all, it may have noisy job size estimates, or it may have other types of partial job size information. *How should we schedule in the face of job size uncertainty?*

2.1 No Job Size Information Suppose we wish to minimize mean response time, which involves prioritizing small jobs ahead of big jobs. How can we do this without knowing each job’s size? The key idea is that even if we do not know any individual job’s size ahead of time, we can measure how big jobs are as they complete. Aggregating this data over times gives us a good picture of the *job size distribution.*

The question of how to leverage the job size distribution to minimize mean response time has been well studied. A policy called the *Gittins* policy [3] is optimal in many cases, but several open questions remain.

- All existing proofs of Gittins’s optimality make limiting assumptions on the job size distribution [20]. *Is Gittins optimal even without these limiting assumptions?*
- Even in cases where Gittins is known to be optimal, we do not know what mean response time it actually achieves. *What is Gittins’s mean response time?*
- Gittins is only optimal in single-server systems. *Does Gittins still perform well in multiserver systems?*
- Gittins is a famously complex policy. *Can we match Gittins’s performance with simpler heuristics?*

My research answers the above questions for the first time.

- We give the *first proof of Gittins’s optimality without limiting assumptions* [20].
- We provide the *first response time analysis of Gittins* [23]. This work, which actually covers much more than just Gittins (§5.1), was a finalist of the 2019 INFORMS APS Best Student Paper Prize.
- We prove the *first bounds on Gittins’s response time in multiserver systems* [16]. Our bounds imply that Gittins has optimal mean response time in heavy traffic.
- We give the *first analysis of simpler alternatives to Gittins* [17, 21, 24]. We find that in general, very simple policies like First-Come, First-Served (FCFS) can be much worse than Gittins, but more advanced heuristics that are still simpler than Gittins perform nearly as well.

2.2 Estimated Job Sizes Many systems are able to give the scheduler a noisy estimate of each job’s size. How should we schedule to minimize mean response time with noisy job size estimates? One candidate policy is *SRPT with Estimates* (SRPT-E). SRPT-E is like SRPT, but it uses estimated size instead of true size when computing a job’s priority. Unfortunately, SRPT-E has been observed to be *non-robust*, performing poorly even under low noise [2]. *How do we schedule in a way that is robust to noisy job size estimates?*

We propose the *first scheduling policy that is provably robust to noisy job size estimates* [18]. Our policy, called *SRPT with Bounce* (SRPT-B), has provably near-optimal mean response time under low noise, and its performance degrades gracefully as noise increases.

In contrast to the stochastic setting, near-optimal performance under low noise is impossible in the worst-case setting [1]. This means that worst-case analysis alone would not identify SRPT-B as a good contender, underscoring the importance of stochastic analysis.

2.3 Other Types of Uncertainty While our initial focus on the Gittins policy was in the case where no job size information is available, most of our results extend to cases with various types of *partial job size information* [16, 20]. One example is when jobs have multiple stages, where each stage's size is unknown but the number of stages remaining is known [22]. In other work, we explore *system state uncertainty*, in which we have delayed or otherwise noisy information about the state of the system. This can occur when scheduling over a network. We show how to adapt a wide class of scheduling policies, including Gittins and SRPT-B, to be robust to system state uncertainty [19].

3 Overheads and Constraints

Most queueing theory works under idealistic assumptions that do not match the messy realities of systems.

- Scheduling policies often assume jobs may be freely preempted at any time. But in some systems, jobs might sometimes be nonpreemptible. When preemption is possible, it typically incurs overhead [11].
- Scheduling policies often assign very granular priority levels. But some systems, such as network switches, are limited by hardware or software to a limited number of priority levels [9, 13].

How should we schedule in light of overheads and constraints?

3.1 Preemption Overhead Preemption in general incurs some overhead. When a job is paused, it takes some time to store the state of the job. When a job is later resumed, it takes some time to reload that state and otherwise "warm up" (e.g. refilling caches). There is very little queueing theory on analyzing even very simple scheduling policies with preemption overhead, with none including both pause and resume overheads. *How do pause and resume overheads affect response time?*

We take the first steps towards answering this question by giving the *first stochastic analysis of scheduling with pause and resume overheads* [14]. My mentee Edwin Peng won the undergraduate division of the SIGMETRICS 2021 Student Research Competition with his presentation of this work.

3.2 Limited Priority Levels Network switches, the Linux scheduler, and other systems all have only a limited number of priority levels built into their hardware or software. Adding more levels is typically costly, requiring more expensive hardware or increasing scheduling overhead. *How should we adapt scheduling policies to limited priority levels? How many levels do we need?*

We provide the *first analysis of scheduling with limited priority levels* [21]. We give a heuristic for adapting policies to the limited-priority-level setting, demonstrating that with only five or six priority levels, one can generally achieve mean response time within 20% of the original policy.

3.3 Preemption Checkpoints Many systems only allow preempting jobs at specific *checkpoints* when their state has been saved. More frequent checkpoints mean more opportunities for preemption, and thus more efficient scheduling. But preemption checkpoints typically incur overhead. *What gap between checkpoints balances the needs of smart scheduling with avoiding overhead?*

We provide the *first analysis of scheduling with preemption checkpoints* [21]. We give a rule-of-thumb formula that takes as input the checkpoint overhead and yields the optimal gap between checkpoints.

4 Tail Metrics

In practice, it is the *tail* of response time, not the mean, that is usually most important. The tail can be described with *probabilities* $\mathbf{P}[T > t]$, the chance response time T exceeds a threshold t, or *percentiles* like t_{99}, which is the value such that $1 - \mathbf{P}[T > t_{99}] = 0.99$. *How should we schedule to optimize tail metrics?*

4.1 Improving All Response Time Tail Probabilities and Percentiles SRPT, in addition to minimizing mean response time, has low tail probability $\mathbf{P}[T > t]$ for small and medium thresholds t, but its tail performance is less good for large thresholds t. On the other hand, the classic *First-Come, First-Served*

(FCFS) policy always minimizes the *maximum* response time of any individual job, suggesting its $\mathbf{P}[T > t]$ performance is good for large t. In fact, for a wide class of job size distributions, FCFS has been conjectured to be *asymptotically optimal* for $\mathbf{P}[T > t]$ in the large t limit.

To summarize: SRPT has better $\mathbf{P}[T > t]$ for small and medium t, whereas FCFS is often better for large t. *Can we get the best of both worlds, achieving good tail probability for all thresholds t?*

We combine aspects of SRPT and FCFS to obtain the *first scheduling policy with strictly better response time tail than FCFS* [6]. Our policy, called *Nudge*, serves jobs FCFS by default, then adds just a hint of SRPT: if a small job arrives and the last job in the queue is big, we "nudge" the big job behind the small job, but only if the big job has not been nudged before. We prove that Nudge has smaller $\mathbf{P}[T > t]$ than FCFS for *all* thresholds t, so it also has smaller t_{99} and other response time percentiles. In particular, Nudge's $\mathbf{P}[T > t]$ is multiplicatively smaller than FCFS's in the large t limit, *disproving FCFS's conjectured asymptotic optimality.* This work won the SIGMETRICS 2021 Best Paper Award.

4.2 Optimizing Both Mean and Tail of Response Time Ideally, one would like to achieve both low mean response time and low response time tail. We know that Gittins minimizes mean response time when job sizes are unknown (§2.1), but its tail performance is unknown. *Can we simultaneously achieve good mean and tail of response time? Can we do so using Gittins?*

We perform the *first analysis of Gittins's response time tail* [25, 26]. We find that when the job size distribution is heavy-tailed, Gittins has asymptotically optimal response time tail, making it a good choice for both mean and tail of response time. The story for light-tailed job size distributions is messier: Gittins's tail can be anywhere from asymptotically optimal to asymptotically *pessimal.* To remedy this, we show how to tweak Gittins to avoid pessimal tail behavior while maintaining near-optimal mean response time [25].

5 New Frontiers in Queueing Theory

My goal as a researcher is not just to solve individual problems, but rather to push frontiers of queueing theory as a field so that wide classes of problems become much easier to solve. This philosophy has already led me to *two widely applicable queueing theoretic innovations*, one each in single-server and multiserver scheduling. I have been invited to give tutorials on these techniques at SIGMETRICS 2019 and 2021.

5.1 Unifying Theory of Single-Server Scheduling The state of the art in queueing theory is that each scheduling policy requires its own specifically tailored analysis. Despite some common themes between the analyses of different policies, analyzing a single policy or small group of related policies is still a paper-length task. As such, many more complex policies, such as Gittins (§2.1), have not yet been analyzed.

I have pioneered a new technique, called *SOAP*,[4] which advances the state of the art by giving *a universal analysis of a wide class of scheduling policies* [23]. SOAP simultaneously unifies and generalizes previous analyses. It provides the first analysis of many policies, including Gittins.

5.2 Simpler Approach to Multiserver Scheduling Analyzing multiserver queueing systems is notoriously difficult, even under FCFS [7]. Introducing advanced scheduling only makes things more challenging.

I have pioneered a new technique, called *Work-to-Time*, that *reduces response time analysis to a simpler problem.* The simpler problem is bounding quantities like the amount of work in the system, which is often easier than directly bounding response time. While Work-to-Time was mainly motivated by multiserver systems [15, 16], it has also proven useful for dealing with uncertainty in single-server systems [18, 20, 25].

6 Future Plans

There are many research directions I am excited to pursue. Here I highlight two broad directions. The first direction (§§6.1–6.5) is more systems-motivated: I believe that new tools in queueing theory can be developed to handle a variety of concerns that show up in modern system architectures, ranging from GPUs

[4] SOAP stands for "Schedule Ordered by Age-based Priority", which describes the class of policies the technique can analyze.

to datacenters. The second direction (§6.6) is more theory-motivated: I believe the tools I have developed for studying Gittins in queues may be helpful in settings beyond queueing theory, ranging from reinforcement learning to contact tracing.

6.1 Datacenters Modern datacenters can house hundreds of thousands of machines, and jobs in these datacenters may run on many machines at once. A recent analysis of traces from Google's Borg clusters show jobs requesting anywhere from one to tens of thousands of machines [8]. *How should we schedule when a single job can occupy many servers?*

6.2 Edge Computing Computing on the edge is a unique setting in many ways. One interesting aspect is that while edge devices are relatively limited in terms of their processing and storage capacities, the cloud, which has much greater capacity, is only a few network hops away. Of course, falling back on the cloud uses bandwidth, which is itself a scarce resource. *How should we schedule when we have limited local resources but ample remote resources?*

6.3 Databases Jobs are submitted to databases in the form of queries, which are written in some query language (e.g. SQL). As such, jobs in databases can have a wide variety of structures, but the system has a lot of information about the structure of the jobs it is serving. For example, different phases of query execution exhibit different levels of parallelism, or a query's processing time may be uncertain at the start but become clearer after executing its first few phases. *How should we schedule when we have detailed knowledge of each job's structure?*

6.4 GPUs for Natural Language Processing GPUs and similar architectures perform the same operations on several threads simultaneously. The result is that GPUs run jobs in batches that complete once their largest job has finished. This batching behavior is well-suited to applications like graphics, which produce with images of predictable sizes. But in applications like natural language processing, models can output an a priori unpredictable amount of text. We might benefit from stopping some batches early, finishing the batch's small jobs more quickly but leaving its largest jobs incomplete, though this raises questions about when to stop a batch and how to handle incomplete jobs. *How should we schedule when jobs of different sizes must be served in batches?*

6.5 Training Machine-Learning Models One can view training a machine-learning model as a job. However, unlike jobs in traditional queueing theory, it is not always obvious when training is complete. For example, gradient descent improves a model's quality initially, but it eventually has diminishing returns. But phenomena like epoch-wise double descent [27] make it hard to tell when this diminishing-returns regime has been reached. When training a model, we might wonder whether we should do more iterations of gradient descent or to try a new combination of hyperparameters. *How should we schedule when job completion is not binary, but serving a job has unpredictably diminishing returns?*

6.6 Gittins for Reinforcement Learning and Exploration-Exploitation Tradeoffs So far I have discussed the Gittins policy in the context of scheduling jobs in queues. But Gittins is actually a very general optimization tool for balancing exploration-exploitation tradeoffs. Such tradeoffs are ubiquitous in problems featuring optimization under uncertainty, including reinforcement learning, search problems, and resource allocation. As a topical example, recent work applies Gittins to contact tracing in epidemics [12].

Compared to other algorithms for managing the exploration-exploitation tradeoff (e.g. UCB, Thompson sampling, Exp3), Gittins requires a more precise stochastic model, but in return, Gittins yields *optimal performance* relative to that stochastic model. I believe that in an age where data is readily available, learning or otherwise constructing stochastic models from data and applying Gittins is a promising approach. *Can we leverage Gittins and big data to design better algorithms for reinforcement learning and other problems with exploration-exploitation tradeoffs?*

References

[1] Yossi Azar, Stefano Leonardi, and Noam Touitou. 2021. Flow Time Scheduling with Uncertain Processing Time. In *Proceedings of the 53rd Annual ACM SIGACT Symposium on Theory of Computing (STOC 2021)*. ACM, Rome, Italy, 1070–1080.

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

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

[4] Isaac Grosof, Ziv Scully, and Mor Harchol-Balter. 2018. SRPT for Multiserver Systems. *Perform. Eval.* 127–128 (Nov. 2018), 154–175. Winner of **PERFORMANCE 2018 Best Student Paper Award**.

[5] Isaac Grosof, Ziv Scully, and Mor Harchol-Balter. 2019. Load Balancing Guardrails: Keeping Your Heavy Traffic on the Road to Low Response Times. *Proc. ACM Meas. Anal. Comput. Syst.* 3, 2, Article 42 (June 2019), 31 pages. Winner of **SIGMETRICS 2019 Best Student Paper Award** and featured as a **STOC 2021 TheoryFest Mini-Plenary**.

[6] Isaac Grosof, Kunhe Yang, Ziv Scully, and Mor Harchol-Balter. 2021. Nudge: Stochastically Improving upon FCFS. *Proc. ACM Meas. Anal. Comput. Syst.* 5, 2, Article 21 (June 2021), 29 pages. Winner of **SIGMETRICS 2021 Best Paper Award**.

[7] 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 Syst.* 64, 1 (Jan. 2010), 5–48.

[8] Mor Harchol-Balter. 2021. Open Problems in Queueing Theory Inspired by Datacenter Computing. *Queueing Syst.* 97, 1 (Feb. 2021), 3–37.

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

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

[11] Chuanpeng Li, Chen Ding, and Kai Shen. 2007. Quantifying the Cost of Context Switch. In *Proceedings of the 2007 Workshop on Experimental Computer Science (ExpCS '07)*. ACM, San Diego, CA, 4.

[12] Michela Meister and Jon Kleinberg. 2021. Optimizing the Order of Actions in Contact Tracing. *arXiv* (July 2021), 37 pages. arXiv:2107.09803

[13] 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.

[14] Edwin Peng and Ziv Scully. 2021. Exact Response Time Analysis of Preemptive Priority Scheduling with Switching Overhead. In preparation. Winner of **SIGMETRICS 2021 Student Research Competition**, undergraduate division.

[15] Ziv Scully. 2021. Bounding Mean Slowdown in Multiserver Systems. *ACM SIGMETRICS Performance Evaluation Review* (2021). To appear. Full version in preparation.

[16] Ziv Scully, Isaac Grosof, and Mor Harchol-Balter. 2020. The Gittins Policy Is Nearly Optimal in the M/G/k under Extremely General Conditions. *Proc. ACM Meas. Anal. Comput. Syst.* 4, 3, Article 43 (Nov. 2020), 29 pages.

[17] Ziv Scully, Isaac Grosof, and Mor Harchol-Balter. 2021. Optimal Multiserver Scheduling with Unknown Job Sizes in Heavy Traffic. *Perform. Eval.* 145, Article 102150 (Jan. 2021), 31 pages.

[18] Ziv Scully, Isaac Grosof, and Michael Mitzenmacher. 2022. Uniform Bounds for Scheduling with Job Size Estimates. In *13th Innovations in Theoretical Computer Science Conference (ITCS 2022) (Leibniz International Proceedings in Informatics (LIPIcs))*. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, Berkeley, CA, Article 41, 30 pages.

[19] 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.

[20] Ziv Scully and Mor Harchol-Balter. 2021. The Gittins Policy in the M/G/1 Queue. In *19th International Symposium on Modeling and Optimization in Mobile, Ad Hoc, and Wireless Networks (WiOpt 2021)*. IEEE, Philadelphia, PA, 8 pages.

[21] Ziv Scully and Mor Harchol-Balter. 2021. How to Schedule Near-Optimally under Real-World Constraints. *arXiv* (Oct. 2021), 25 pages. arXiv:1805.06865 In submission.

[22] Ziv Scully, Mor Harchol-Balter, and Alan Scheller-Wolf. 2018. Optimal Scheduling and Exact Response Time Analysis for Multistage Jobs. *arXiv* (Nov. 2018), 31 pages. arXiv:1805.06865

[23] Ziv Scully, Mor Harchol-Balter, and Alan Scheller-Wolf. 2018. SOAP: One Clean Analysis of All Age-Based Scheduling Policies. *Proc. ACM Meas. Anal. Comput. Syst.* 2, 1, Article 16 (April 2018), 30 pages. Finalist of **2019 INFORMS APS Best Student Paper Prize**.

[24] Ziv Scully, Mor Harchol-Balter, and Alan Scheller-Wolf. 2020. Simple Near-Optimal Scheduling for the M/G/1. *Proc. ACM Meas. Anal. Comput. Syst.* 4, 1, Article 11 (May 2020), 29 pages. Winner of **SIGMETRICS 2020 Best Video Award**.

[25] Ziv Scully and Lucas van Kreveld. 2021. When Does the Gittins Policy Have Asymptotically Optimal Response Time Tail? *ACM SIGMETRICS Performance Evaluation Review* (2021). To appear. Full version in submission. arXiv:2110.06326.

[26] 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. *Proc. ACM Meas. Anal. Comput. Syst.* 4, 2, Article 30 (June 2020), 33 pages.

[27] Cory Stephenson and Tyler Lee. 2021. When and How Epochwise Double Descent Happens. *arXiv* (Aug. 2021), 15. arXiv:2108.12006