The most common queueing theory questions asked by computer systems practitioners

Mor Harchol-Balter*
Carnegie Mellon University
harchol@cs.cmu.edu

Ziv Scully
Carnegie Mellon University
zscully@cs.cmu.edu

ABSTRACT

This document examines five performance questions which are repeatedly asked by practitioners in industry: *(i)* My system utilization is very low, so why are job delays so high? *(ii)* What should I do to lower job delays? *(iii)* How can I favor short jobs if I don't know which jobs are short? *(iv)* If some jobs are more important than others, how do I negotiate importance versus size? *(v)* How do answers change when dealing with a closed-loop system, rather than an open system? All these questions have simple answers through queueing theory. This short paper elaborates on the questions and their answers. To keep things readable, our tone is purposely informal throughout. For more formal statements of these questions and answers, please see [14].

Keywords

queueing theory; performance; industry; practice; scheduling; variability; closed systems; SRPT; SERPT, Gittins, SITA; pooling; $c\mu$-rule; job value; holding cost.

1. MY SYSTEM UTILIZATION IS LOW, SO WHY ARE JOB DELAYS SO HIGH?

How can one have low system utilization, but high delay? To make this question more concrete, consider a single-server queue shown in Figure 1, where jobs are processed in First-Come-First-Served (FCFS) order. Here λ jobs/sec is the rate at which jobs arrive. The jobs have different *sizes (service requirements)*, where S is a random variable representing job size (in seconds). The *utilization (load)*, denoted by ρ, is the long-run fraction of time that the server is busy, where $\rho = \lambda \cdot \mathbf{E}[S]$. The *response time* ($T$) of a job is the time from when a job arrives until it completes. The *delay* (D) of a job, a.k.a. its *queueing time*, is just $T - S$, namely the job's response time minus its size.

It seems intuitive that if the server is only utilized $\rho = 20\%$ of the time, then the average job delay should be low. However this intuition is false! In fact, the mean job delay in the case of a single-server queue depends on *three* factors: *(i)* the utilization ρ; *(ii)* the variability of job service times; *(iii)* the variability of job inter-arrival times. Kingman's approximation [20] states that:

$$\mathbf{E}[\text{Delay}] \approx \frac{\rho}{1-\rho} \cdot \left(\frac{C_S^2 + C_A^2}{2} \right) \cdot \mathbf{E}[S].$$

Here C_S^2 represents the squared coefficient of variation of the job service time, also known as the normalized variance, where $C_S^2 = \mathbf{Var}(S)/\mathbf{E}[S]^2$. Likewise C_A^2 is the corresponding quantity for job inter-arrival times. Variability in the job service times leads to high delays because it means that short jobs end up waiting behind long ones, thus "inheriting" the delays of the long jobs. Likewise variability in the inter-arrival times is problematic because it means that jobs arrive in bursts.

In computing systems, C_S^2 in particular tends to be quite high, often in the hundreds. In fact, a recent study of jobs run by the Google Borg Scheduler showed $C_S^2 = 23,000$ [29]. This study also found that the largest 1% of jobs were responsible for 99% of the total load (total work). Such high values of C_S^2 can easily dwarf the effect of low ρ, resulting in high mean delay, despite low utilization.

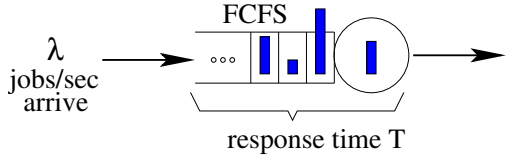

Figure 1: ***Single server FCFS queue. The rectangles represent jobs of different sizes, where the height of the rectangle is the job's size. The response time is the time from when the job arrives until it completes.***

2. WHAT ARE TECHNIQUES FOR LOWERING JOB DELAY?

We present three very different solutions for lowering job delay.

2.1 Scheduling to Favor Short Jobs

The first solution is scheduling to favor short jobs. Ideally, one should schedule jobs in Shortest-Remaining-Processing-Time (SRPT) order, at all times preemptively running the job that will finish soonest. While SRPT is optimal for any arrival stream, it turns out that simpler variants of SRPT work almost as well. For example, one can run SRPT with only 3 size buckets and obtain similar mean delay [18, 21]. Likewise one could run Preemptive-Shortest-Job-First (PSJF)

*This work was supported by: NSF-CMMI-1938909, NSF-CSR-1763701, and a Google 2020 Faculty Research Award.

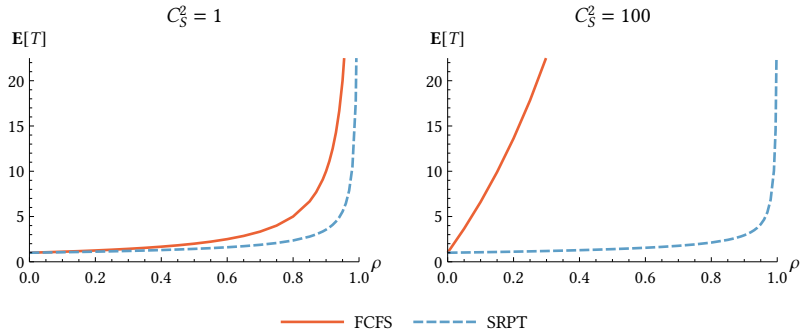

Figure 2: ***Mean response time, $\mathbf{E}[T]$, as a function of load, under SRPT scheduling compared with FCFS scheduling. (a) shows a queue with lower job size variability, $C_S^2 = 1$ (M/G/1 queue where S is Exponential with mean 1). (b) shows a queue with higher job size variability, $C_S^2 = 100$ (M/G/1 queue where S is two-phase Hyperexponential with mean 1 and balanced means across the phases). The "M/G/1" notation refers to a single-server queue with Poisson arrival process and Generally-distributed job sizes.***

which performs almost as well. One can also extend SRPT to a k-server setting, where at all times one runs the k jobs with shortest remaining processing time [11].

SRPT scheduling has big advantages over FCFS when the job service requirements have high variability, as shown in Figure 2, because it ensures that short jobs don't get stuck waiting behind long ones. However practitioners are wary of using SRPT, because they fear that long jobs will be unduly penalized. In truth, long jobs are not treated unfairly compared with short ones, provided that the utilization is not too high (see [5,30]); however these fears are deep-rooted, and hence SRPT scheduling is not used as often as one would expect.

2.2 Dispatching to Isolate Short Jobs

An alternative solution is to physically isolate the short jobs from the long ones. For example, the Size-Interval-Task-Assignment (SITA) scheme proposed in [7, 15] dispatches short jobs to one set of servers and long jobs to a second set, as shown in Figure 3.

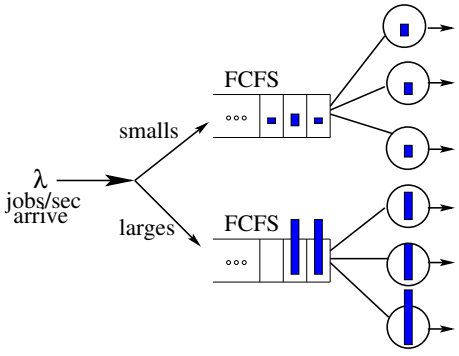

Figure 3: ***SITA dispatching physically separates short and long jobs.***

SITA greatly decreases mean delay because short jobs don't have to wait behind long jobs. In particular, short jobs only experience the job size variability (and load) of the other short jobs. The only question that comes up is how to choose the cutoff defining "short" versus "long." One idea is to choose the cutoff to equalize the load between shorts and longs, as in SITA-E. For typical computing workloads, where the largest 1% of jobs comprise most of the load, SITA-E would send fewer than 1% of jobs to the long-job queue. One can further optimize the cutoff based on the job size distribution, see [19].

2.3 Pooling Resources

A well-known trick from queueing theory for reducing delay is simply to pool resources. Figure 4 (left) shows k separate queues, each with arrival rate λ, mean job size $\mathbf{E}[S]$, and load $\rho = \lambda \mathbf{E}[S]$. Figure 4(right) shows the k queues pooled into a single queue with arrival rate $k\lambda$, where the single queue is served by the k servers. Pooling does not change the system load, which is now $\rho = \frac{k\lambda \mathbf{E}[S]}{k} = \lambda \mathbf{E}[S]$, which is the same as in the original system. However the delay of the pooled system is *far* lower than that of the original system. For example, when job sizes follow an Exponential distribution, then (see [14, p.270]),

$$\mathbf{E}\left[\text{Delay of jobs that queue in pooled system}\right] = \frac{1}{k\lambda} \cdot \frac{\rho}{1-\rho}.$$

The point is that, since ρ and λ are constants, the delay in the pooled system drops multiplicatively with k. Pooling can offer even more benefit for generally-distributed job sizes, see [22].

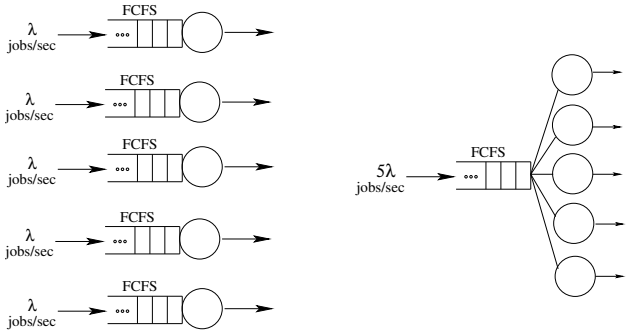

Figure 4: ***On the left we see $k = 5$ separate queues versus. On the right we see a single pooled queue.***

To understand why pooling reduces delay so much, observe that although each server is still busy ρ fraction of the time on average, the chance that *all* servers are busy at the same time goes down. Thus the chance that a job has to queue drops a lot, and the delay of those jobs that do queue is very small. Pooling of resources is a very useful tool in systems design. It not only helps to mitigate delay but also helps to mitigate many overheads like setup times, see [8, 9].

3. WHAT IF I DON'T KNOW JOB SIZE?

In Section 1, we saw that delay is largely related to job size variability, which can be very high. In Section 2 we saw that providing short jobs some isolation from long jobs can be very helpful in mitigating the effect of high job size variability. This isolation for short jobs can be obtained either by giving shorts priority (as in Section 2.1) or by giving short jobs their own dedicated resources (as in Section 2.2). But what can we do to minimize delay if we don't know job sizes a priori? We first discuss alternative scheduling policies and then alternative dispatching policies for situations when job size is not known.

There are many scheduling alternatives to Section 2.1 which don't require knowledge of job size, yet still combat job size variability. The simplest idea is *Processor-Sharing (PS)* scheduling, which simply time-shares the server among all jobs in the queue. By allowing all jobs to receive some quantum of service, shorter jobs will obviate waiting behind longer jobs. However we can go further than PS by using knowledge the *age* of a job, namely the amount of service that a job has received so far, as a proxy for its remaining size. Specifically, it is often the case that jobs with lower age are more likely to complete sooner, while jobs with higher age are more likely to have high remaining times – this is a property called Decreasing Hazard Rate (DHR). In the case of DHR, it pays to rank jobs in terms of their age, giving priority to jobs with lower age [1]. The algorithm that does this is called *Least Attained Service (LAS)*, see [13]. If the job size distribution does not exhibit DHR, one can instead use a job's age, together with knowledge of the job size distribution, to rank jobs in terms of their *expected* remaining processing time. Specifically, if S is a random variable denoting a job's size, and a is a job's current age, then one can compute the job's expected remaining size as:

$$\mathbf{E}\left[S - a \mid S > a\right],$$

and then schedule to at all times (preemptively) run the job with the Smallest Expected Remaining Processing Time (SERPT). The *optimal* scheduling policy when job sizes are not known is the Gittins Index policy, which ranks jobs based on a combination of the job's expected remaining size and its probability of completing in the next moment. While optimal [2, 3, 10, 26] the Gittins Index policy is quite complicated, and, in practice, SERPT typically works just as well. The exact response time for all these policies (in an M/G/1 setting) is derived in [28]. All of the above policies can be generalized to settings with k servers (the M/G/k), see [11, 24, 25].

Figure 5(top) shows a job size distribution, where jobs can have sizes ranging from 0 to 16, with different densities. An individual job's size is *not* known, but what *is* known is a job's age, namely the service that a job has received so far. As we see in Figure 5(bottom), a job's expected remaining size changes with its age. At first the expected remaining size is small, because most jobs have small size (they lie in the first "hump" of the distribution); but once the age of a job passes that first hump in the job size distribution, the expected remaining size increases, and it increases again when we pass the second hump.

Figure 6 shows the mean response time under different scheduling policies (FCFS, PS, SERPT, Gittins, and SRPT) when job sizes come from the distribution in Figure 5. FCFS, PS, SERPT, and Gittins assume that job size is not known. SERPT and Gittins obtain their good performance by using the expected remaining job size, given in Figure 5(bottom). As we see, the SERPT policy has performance very close to that of Gittins, which is optimal. See [27] for more discussion.

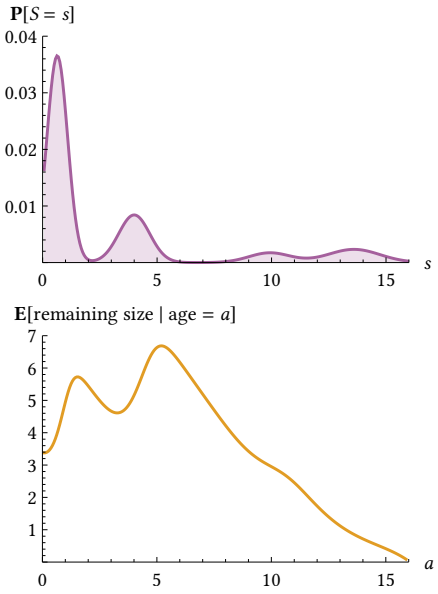

Figure 5: ***(top) A job size distribution. (bottom) The expected remaining size of a job as a function of its age.***

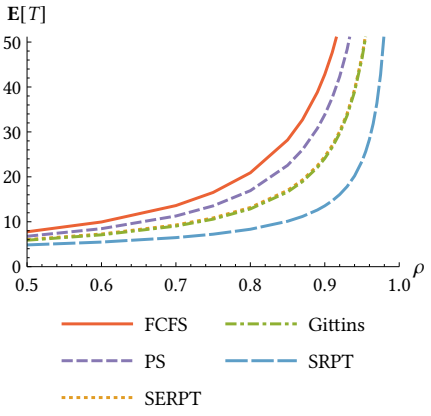

Figure 6: ***Mean response time as a function of load, under different scheduling policies. FCFS, PS, SERPT, and Gittins assume that job size is not known. We compare with SRPT scheduling, which makes use of knowing job size.***

There are also dispatching alternatives to the SITA policy from Section 2.2 which don't require options which don't require knowledge of job size. An example is the Task-Assignment-by-Guessing-Size (TAGS) dispatching policy [4, 12]. TAGS assumes that all jobs are short, and thus initially sends all jobs to the queue for short jobs. However TAGS caps the running time of all jobs at the short job queue; if the job doesn't complete within the "short cap" limit, then the job is restarted at the long job queue. In this way, TAGS limits the pain that long jobs can inflict on short ones.

Of course the pooling option from Section 2.3 is also an alternative to SITA that does not require knowing job size. See [16] and [17] for a discussion on how pooling compares with SITA-based algorithms.

4. HOW DO I NEGOTIATE IMPORTANCE VERSUS SIZE?

If certain jobs are more important (more valuable) than others, the standard solution is to create a priority queue structure, where jobs of higher importance have higher priority and can preempt those of lower importance. But what do we do if the most important jobs all have large remaining size, while some other less important jobs have very small remaining size? Are we still getting the most value over time by strictly favoring the more important jobs?

This question of how to trade off importance and size has been studied in queueing theory. We define the notion of a *holding cost* for a job, which is a rate, in the form of dollars/second, that the system incurs for not having yet completed the job. That is, the holding cost of a job is a rate of money that we're burning by *not* doing the job. Jobs with high holding cost are more "valuable."

The *total holding cost* at time t over all jobs is:

$$\text{TotalHoldingCost}(t) = \sum_{\substack{\text{jobs } j \text{ in the} \\ \text{system at time } t}} (\text{holding cost of job } j) .$$

Our goal is to minimize *expected holding cost*, the time-average of the total holding cost at time t:

$$\mathbf{E}\left[\text{Holding Cost}\right] = \lim_{u\to\infty} \frac{1}{u} \int_0^u \text{TotalHoldingCost}(t)dt .$$

The algorithm for minimizing expected holding cost is known as the $c\mu$-rule [6], which assigns to each job an index, where

$$\text{Index(job)} = \frac{\text{Holding cost of job}}{\text{Remaining size of job}},$$

and where priority is given to the job of *highest* index. In this way, the $c\mu$-rule favors jobs whose holding cost is high or whose remaining size is small. While the $c\mu$-rule is not optimal in a worst-case sense, it is optimal in many settings, including the M/G/1 queue [26].

5. HOW DO ANSWERS CHANGE GIVEN CLOSED-LOOP ARRIVALS?

Thus far we have assumed that the stream of job arrivals into our systems is exogenous (external) to the system, meaning that new arrivals are not affected by what's going on inside the system. By contrast, some systems obey a *closed-loop arrival model*, where new job arrivals are only triggered by job completions (perhaps followed by some delay, called the "thinking" time). That is, only when a job completes does a new job get to enter (after some thinking period). For such closed-loop configurations, there is typically a limit on the number of jobs which can be in the system simultaneously, called the Multi-Programming Level (MPL).

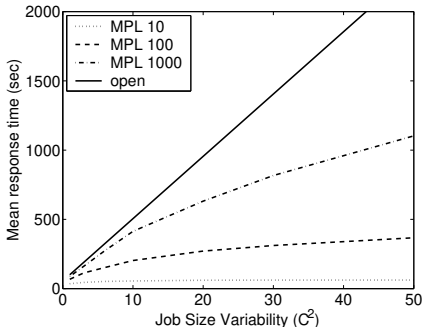

Figure 7: ***Graph (copied from [23]) shows mean response time for an open and a closed FCFS queue system as a function of job size variability. The solid line represents the open system and the dashed lines represent closed systems with different MPLs. In all cases mean job size is*** $\mathbf{E}[S] = 10$ ***and*** $\rho = 0.9$ ***(for the open system, we achieve*** ρ ***by adjusting the arrival rate*** λ***, and for closed systems by adjusting the think time). We see that closed systems are far less sensitive to the effects of job size variability, particularly for lower MPL.***

As explained in [23], the performance of closed-loop systems is very different from open ones, even when both systems are operated under the same load (utilization) ρ. Delays in general are significantly smaller for closed systems. While job size variability has a huge impact on delay in open systems, it has much less of an effect on job delay in closed-loop systems (see Figure 7). Consequently, while scheduling to favor short jobs (or jobs that are likely to be short) is vital in open systems, to combat variability, size-based scheduling is far less important for closed systems. Intuitively, the difference in closed-loop systems and open ones stems from the fact that the number of jobs in the closed system is limited to the MPL. Even if the MPL is high (in the hundreds), the fact that there's a limit at all means that the mean delay is significantly lower in closed systems and the effect of high job size variability is mitigated – when there are fewer jobs total, then there are fewer short jobs stuck behind long ones.

Given the stark differences between closed-loop systems and open ones, it is very important to start every performance discussion by understanding what kinds of system configuration one is dealing with. Likewise, when testing a system via a workload generator, it is also important to be cognizant of whether the workload generator is generating arrivals in a closed loop or exogeneously.

6. REFERENCES

[1] Samuli Aalto and Urtzi Ayesta. Optimal scheduling of jobs with a DHR tail in the M/G/1 queue. In *VALUETOOLS 2008*, October 2008.

[2] Samuli Aalto, Urtzi Ayesta, and Rhonda Righter. On the Gittins index in the M/G/1 queue. *Queueing*

Systems, 63(1):437–458, 2009.

[3] Samuli Aalto, Urtzi Ayesta, and Rhonda Righter. Properties of the Gittins index with application to optimal scheduling. *Probability in the Engineering and Informational Sciences*, 25(3):269–288, 2011.

[4] Eitan Bachmat, Josu Doncel, and Hagit Sarfati. Analysis of the task assignment based on guessing size policy. *Performance Evaluation*, 142, 2020.

[5] Nikhil Bansal and Mor Harchol-Balter. Analysis of SRPT scheduling: Investigating unfairness. In *Proceedings of ACM SIGMETRICS*, pages 279–290, Cambridge, MA, June 2001.

[6] D.R. Cox and W.L. Smith. *Queues*. Kluwer Academic Publishers, 1971.

[7] Mark Crovella, Mor Harchol-Balter, and Cristina Murta. Task assignment in a distributed system: Improving performance by unbalancing load. In *Proceedings of the ACM SIGMETRICS Joint International Conference on Measurement and Modeling of Computer Systems*, pages 268–269, June 1998. Poster Session.

[8] Anshul Gandhi, Sherwin Doroudi, Mor Harchol-Balter, and Alan Scheller-Wolf. Exact analysis of the M/M/k/setup class of Markov chains via Recursive Renewal Reward. In *ACM SIGMETRICS 2013 Conference on Measurement and Modeling of Computer Systems*, pages 153–166, 2013.

[9] Anshul Gandhi and Mor Harchol-Balter. How data center size impacts the effectiveness of dynamic power management. In *49th Annual Allerton Conference on Communication, Control, and Computing*, pages 1164–1169, Urbana-Champaign, IL, September 2011.

[10] John C. Gittins, Kevin D. Glazebrook, and Richard Weber. *Multi-armed Bandit Allocation Indices*. John Wiley & Sons, 2011.

[11] Isaac Grosof, Ziv Scully, and Mor Harchol-Balter. SRPT for multiserver systems. *Performance Evaluation*, 127:154–175, November 2018.

[12] Mor Harchol-Balter. Task assignment with unknown duration. *Journal of the ACM*, 49(2):260–288, March 2002.

[13] Mor Harchol-Balter. Queueing disciplines. Wiley Encyclopedia of Operations Research and Management Science, 2011.

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

[15] Mor Harchol-Balter, Mark Crovella, and Cristina Murta. On choosing a task assignment policy for a distributed server system. In *Lecture Notes in Computer Science, No. 1469: 10th International Conference on Modeling Techniques and Tools for Computer Performance Evaluation*, pages 231–242, September 1998.

[16] Mor Harchol-Balter, Alan Scheller-Wolf, and Andrew Young. Surprising results on task assignment in server farms with high-variability workloads. In *ACM SIGMETRICS 2009 Conference on Measurement and Modeling of Computer Systems*, pages 287–0298, 2009.

[17] Mor Harchol-Balter, Alan Scheller-Wolf, and Andrew Young. Why segregating short jobs from long jobs under high variability is not always a win. In *Forty-seventh Annual Allerton Conference on Communication, Control, and Computing*, pages 121–127, University of Illinois at Urbana-Champaign, October 2009.

[18] Mor Harchol-Balter, Bianca Schroeder, Nikhil Bansal, and Mukesh Agrawal. Size-based scheduling to improve web performance. *ACM Transactions on Computer Systems*, 21(2):207–233, May 2003.

[19] Mor Harchol-Balter and Rein Vesilo. To balance or unbalance load in size-interval task allocation. *Probability in the Engineering and Informational Sciences*, 24:219–244, 2010.

[20] J.F.C. Kingman. Two similar queues in parallel. *Biometrika*, 48:1316–1323, 1961.

[21] Behnam Montazeri, Yilong Li, Mohammad Alizadeh, and John K. Ousterhout. Homa: A receiver-driven low-latency transport protocol using network priorities. In *Proceedings of SIGCOMM 2018*, pages 221–235. ACM SIGCOMM, 2018.

[22] Alan Scheller-Wolf and Rein Vesilo. Structural interpretation and derivation of necessary and sufficient conditions for delay moments in fifo multiserver queues. *Queueing Systems*, 54(3):221–232, 2006.

[23] Bianca Schroeder, Adam Wierman, and Mor Harchol-Balter. Open versus closed: a cautionary tale. In *Proceedings of Networked Systems Design and Implementation (NSDI)*, 2006.

[24] Ziv Scully, Isaac Grosof, and Mor Harchol-Balter. The Gittins policy is nearly optimal in the M/G/k under extremely general conditions. *Proceedings of ACM on Measurement and Analysis of Computer Systems (POMACS/SIGMETRICS)*, 4(3):1–29, 2020. Article 43.

[25] Ziv Scully, Isaac Grosof, and Mor Harchol-Balter. Optimal multiserver scheduling with unknown job sizes in heavy traffic. In *38th International Symposium on Computer Performance, Modeling, Measurement, and Evaluation (IFIP PERFORMANCE 2020)*, Milan, Italy, November 2020.

[26] Ziv Scully and Mor Harchol-Balter. 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 '21)*, Philadelphia, PA, October 2021.

[27] Ziv Scully and Mor Harchol-Balter. How to schedule near-optimally under real-world constraints. *arXiv*, 2021.

[28] Ziv Scully, Mor Harchol-Balter, and Alan Scheller-Wolf. SOAP: One clean analysis of all age-based scheduling policies. *Proceedings of ACM on Measurement and Analysis of Computer Systems (POMACS/SIGMETRICS)*, 2(1):1–30, 2018. Article 16.

[29] Muhammad Tirmazi, Adam Barker, Nan Deng, MD E. Haque, Zhijing Gene Qin, Steven Hand, Mor Harchol-Balter, and John Wilkes. Borg: The next generation. In *Proceedings of the Fifteenth European Conference on Computer Systems (EuroSys '20)*, pages 1–14, Greece, April 2020.

[30] Adam Wierman and Mor Harchol-Balter. Classifying scheduling policies with respect to unfairness in an M/GI/1. In *Proceedings of ACM SIGMETRICS*, pages 238–249, San Diego, CA, June 2003.