Strongly Tail-Optimal Scheduling in the Light-Tailed M/G/1

GEORGE YU, Cornell University, USA
ZIV SCULLY, Cornell University, USA

We study the problem of scheduling jobs in a queueing system, specifically an M/G/1 with light-tailed job sizes, to asymptotically optimize the response time tail. This means scheduling to make $\mathbf{P}[T > t]$, the chance a job's response time exceeds t, decay as quickly as possible in the $t \to \infty$ limit. For some time, the best known policy was First-Come First-Served (FCFS), which has an asymptotically exponential tail: $\mathbf{P}[T > t] \sim Ce^{-\gamma t}$. FCFS achieves the optimal *decay rate* γ, but its *tail constant* C is suboptimal. Only recently have policies that improve upon FCFS's tail constant been discovered. But it is unknown what the optimal tail constant is, let alone what policy might achieve it.

In this paper, we derive a closed-form expression for the optimal tail constant C, and we introduce γ-*Boost*, a new policy that achieves this optimal tail constant. Roughly speaking, γ-Boost operates similarly to FCFS, but it pretends that small jobs arrive earlier than their true arrival times. This significantly reduces the response time of small jobs without unduly delaying large jobs, improving upon FCFS's tail constant by up to 50% with only moderate job size variability, with even larger improvements for higher variability. While these results are for systems with full job size information, we also introduce and analyze a version of γ-Boost that works in settings with partial job size information, showing it too achieves significant gains over FCFS. Finally, we show via simulation that γ-Boost has excellent practical performance.

CCS Concepts: • **General and reference** → **Performance**; • **Mathematics of computing** → **Queueing theory**; • **Networks** → **Network performance modeling**; • **Computing methodologies** → *Model development and analysis*; • **Software and its engineering** → *Scheduling*.

Additional Key Words and Phrases: scheduling; response time; sojourn time; tail latency; service level objective (SLO); M/G/1 queue; light-tailed distribution; FCFS; Boost scheduling

ACM Reference Format:
George Yu and Ziv Scully. 2024. Strongly Tail-Optimal Scheduling in the Light-Tailed M/G/1. *Proc. ACM Meas. Anal. Comput. Syst.* 8, 2, Article 27 (June 2024), 33 pages. https://doi.org/10.1145/3656011

1 INTRODUCTION, BACKGROUND, AND KEY IDEAS

Service Level Objectives (SLOs) for practical queueing systems often relate to the *tail* of the system's response time distribution T. The tail is the function that maps an amount of time t to $\mathbf{P}[T > t]$, the probability that a job's response time T exceeds t, where a job's *response time* is the amount of time between the job's arrival and departure.

Motivated by the problem of meeting SLOs, we consider the problem of scheduling jobs to minimize the tail $\mathbf{P}[T > t]$ in the M/G/1 queue. We actually focus on *asymptotically* minimizing the tail, optimizing the decay of $\mathbf{P}[T > t]$ in the $t \to \infty$ limit. This is an extensively studied problem in queueing theory [6, 7, 13–15, 21, 29–32, 34, 39, 40, 44–46, 48] for a number of reasons:

Authors' addresses: George Yu, Cornell University, School of Operations Research and Information Engineering, Ithaca, NY, USA; Ziv Scully, Cornell University, School of Operations Research and Information Engineering, Ithaca, NY, USA.

- Optimizing the tail $\mathbf{P}[T > t]$ for any particular value of t is seldom the sole design objective. Instead, one generally hopes to achieve low $\mathbf{P}[T > t]$ for a range of values of t.
- Because practical SLOs relate to high-quantile response times, meeting those SLOs corresponds to optimizing $\mathbf{P}[T > t]$ for large values of t.
- Optimizing $\mathbf{P}[T > t]$ for fixed finite t appears to be theoretically intractable, but there has been promising recent progress on asymptotic improvements in the $t \to \infty$ limit [21, 45].

In this paper, we study the M/G/1 with light-tailed job size distributions. We propose a new policy, called γ-*Boost*, and prove it has *asymptotically optimal response time tail* in a sense made precise in Section 1.1 below. This resolves a *significant open problem in queueing theory* [7, 46]. Moreover, γ-Boost has excellent practical performance, as illustrated in Fig. 1.2.

The rest of this section gives background on the problem of asymptotically optimal tail scheduling, with discussion of prior work integrated throughout, and describes the main ideas behind our solution. See Section 1.7 for a summary of our contributions and an outline of the rest of the paper.

1.1 Background on weak and strong tail optimality

Consider an M/G/1 with job size distribution S. For now, we primarily focus on the *full-information* setting where job sizes are known to the scheduler, but some of our results apply more broadly to *partial-information* settings (Section 2.2).

Let T_π denote the response time distribution under policy π. Following Boxma and Zwart [7], we say a policy π is *weakly tail-optimal* if there exists a constant $c \geq 1$ such that

$$\sup_{\pi'} \limsup_{t \to \infty} \frac{\mathbf{P}[T_\pi > t]}{\mathbf{P}[T_{\pi'} > t]} = c.$$

If additionally $c = 1$, we say π is *strongly tail-optimal*.

Whether a scheduling policy is weakly tail-optimal depends critically on whether the job size distribution S is heavy-tailed or light-tailed. If S is heavy-tailed, then several preemptive policies like Shortest Remaining Processing Time (SRPT) and Least Attained Service (LAS) are known to be weakly tail-optimal and conjectured to to be strongly tail-optimal [7, 46]. In fact, we observe in Appendix A that a result of Wierman and Zwart [46] implies strong tail optimality of SRPT, LAS, and other policies for an important class of heavy-tailed distributions. The problem of achieving strong tail optimality is thus largely solved in the heavy-tailed case.

In this work, we focus on the case of light-tailed job size distributions S, specifically so-called *class I* distributions [1, 2] (Definition 2.1), for which strong tail optimality is a significant open problem [7, 46]. For some time, the only common policy known to be weakly tail-optimal was First-Come First-Served (FCFS), which has asymptotically exponential response time tail. That is,

$$\mathbf{P}[T_{\text{FCFS}} > t] \sim C_{\text{FCFS}} \exp(-\gamma t),$$

where $\gamma > 0$ is a constant called the *decay rate*, and $C_{\text{FCFS}} > 0$ is a constant we call FCFS's *tail constant*. Both γ and C_{FCFS} depend on S and the system's arrival rate.

It is known that no policy can achieve asymptotic decay rate greater than γ [7, 44], so we can measure the performance of a weakly tail-optimal policy π by its tail constant

$$C_\pi = \lim_{t \to \infty} \exp(\gamma t)\, \mathbf{P}[T > t]. \tag{1.1}$$

The question of finding a strongly tail-optimal policy thus amounts to minimizing C_π over all policies π. Until recently, it was conjectured that FCFS may be strongly tail-optimal, but recent progress has improved upon FCFS's tail constant [21, 45] (Section 1.3). This prompts a question:

What is the smallest possible tail constant C_π, and what policy π achieves it?

1.2 Obstacle: prioritizing short jobs without delaying long jobs

As explained in Section 1.1, optimizing tail asymptotics with light-tailed job sizes is an open problem, in contrast to the heavy-tailed case. Why is the light-tailed case so much more difficult? The main obstacle is that there is a tension between prioritizing short jobs and delaying long jobs. This tension is best illustrated by contrasting two policies, FCFS and SRPT.

Suppose a "tagged" job of random size S arrives to a steady-state system and observes *work* W, meaning the total remaining service time of jobs in the system is W.

- FCFS serves jobs in the order they arrive. This means the tagged job's response time is $T_{\text{FCFS}} = W + S$. In particular, the job's response time is unaffected by future arrivals.
- SRPT always preemptively serves the job of least remaining service time. This means the tagged job may not need to wait for all of the work W to be completed before entering service. But future arrivals of size less than S may be prioritized over the tagged job.

The reason SRPT is good in the heavy-tailed setting is that the amount of work from future arrivals that delays the tagged job, which we denote by $R_{\text{SRPT}}(S)$, has a lighter tail than W. But in the light-tailed setting, $R_{\text{SRPT}}(S)$ is heavier-tailed than W, with a decay rate less than γ. See Nuyens et al. [31], who prove these results for SRPT and a class of related policies, for details.

The takeaway of the above comparison is that for light-tailed job size distributions, strictly prioritizing short jobs delays long jobs too much for good tail performance. But prioritizing short jobs is essentially the only tool we have for improving response times. The question is thus: how should one *partially* prioritize short jobs to improve tail performance?

A number of works have studied scheduling with some sort of partial priority, whether by having just a few priority buckets [10, 23, 28] or by dynamically changing priority over time [11, 12, 43]. While the tail asymptotics of most of these policies have not been formally studied, they seem unlikely to be weakly tail-optimal. This is because they still have the property that a sufficiently long tagged job might be delayed by a constant fraction of future arrivals. A result of Scully and van Kreveld [39, Proposition 9.9] suggests this should lead to worse decay rate, though their result does not directly apply to all of the other policies cited. Scully and van Kreveld [39, Theorem 5.5] also show that no policy in the recently proposed class of "SOAP" policies [34, 36, 38] can improve upon FCFS's tail constant, because all SOAP policies other than FCFS have decay rate worse than γ.[1]

1.3 Nudge: a promising but limited first step

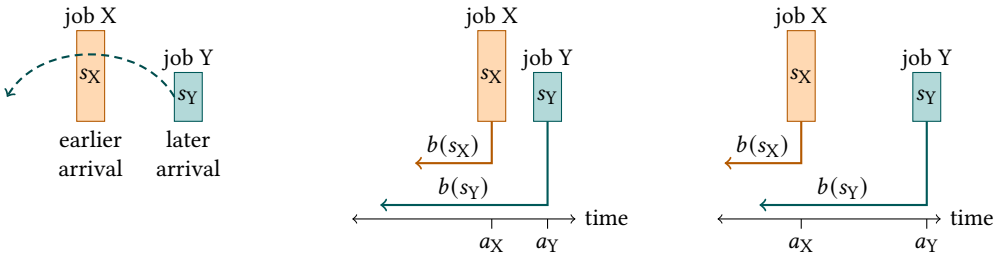
The first improvement upon FCFS's tail constant was through the *Nudge* family of policies, introduced by Grosof et al. [21] and expanded upon by Van Houdt [45] and Charlet and Van Houdt [9]. In its simplest variant, Nudge creates two classes of jobs, *short* and *long*, then runs FCFS with a small modification, illustrated in Fig. 1.1(a):

- When a short job arrives, it is allowed to pass in front of up to K large jobs, where K is a fixed constant.
- Each large job can be passed by a limited number of short jobs. Different variants of Nudge differ in exactly how the limiting works. The two most important variants are the following:
  - *Nudge-K* [45]: Each large job can be passed only once.
  - *Nudge-M* [9]: Small jobs only pass large jobs that are within the K most recent arrivals.

  When $K = 1$, Nudge-K and Nudge-M coincide, and are called simply "Nudge" [21].

Nudge is a family of policies rather than a single policy, because there are many ways to decide which jobs are short and which are long; one can vary the parameter K; and one can choose between Nudge-K, Nudge-M, and other variants [9]. One can think of these parameters as controlling the

[1] Scully and van Kreveld [39] actually consider only a subset of SOAP policies, but one can generalize the relevant part of their argument to cover all SOAP policies.



(a) Nudge always serves X after Y, regardless of their arrival times.

(b) Boost serves X after Y if their arrival times are close together.

(c) Boost serves X before Y if their arrival times are far apart.

Fig. 1.1. Comparison between how Nudge and Boost each handle a long job X arriving before a short job Y. Suppose that Y arrives before X enters service. Nudge decides the order to serve the jobs based only on the arrival order, as shown in (a). In contrast, Boost uses not just the arrival order but also the respective *arrival times*, as shown in (b) and (c). Notation: job i's arrival time is a_i, its size is s_i, and its boost is $b(s_i)$.

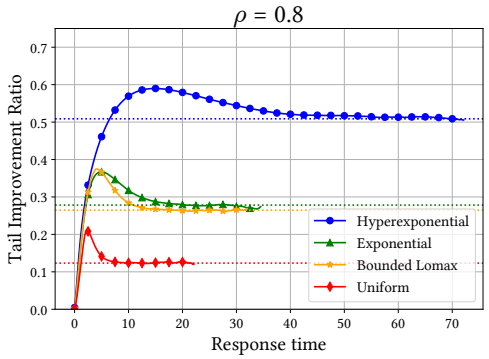
degree to which short jobs are prioritized over long jobs. For instance, larger values of K further prioritize short jobs, and for a fixed value of K, Nudge-K is more conservative about letting short jobs pass long jobs than Nudge-M.

Recent progress on Nudge has yielded several improvements to the best known tail constant. Grosof et al. [21] introduce Nudge with $K = 1$ and show that with appropriate tuning, Nudge achieves $C_{\text{Nudge}} < C_{\text{FCFS}}$, thus demonstrating that FCFS is not strongly tail-optimal. In fact, they show that with sufficiently conservative tuning, Nudge *stochastically improves* upon FCFS, meaning $\mathbf{P}[T_{\text{Nudge}} > t] \le \mathbf{P}[T_{\text{FCFS}} > t]$ for all $t > 0$.

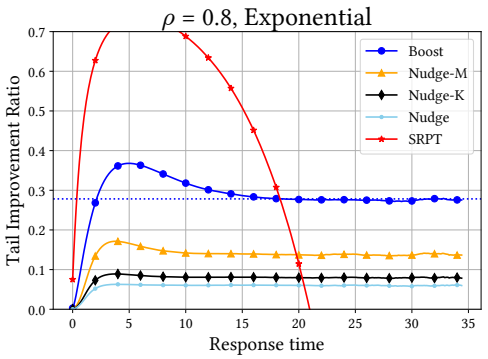
Building on this progress, Van Houdt [45] introduces Nudge-K for $K \ge 2$ and, for any given split between small and large jobs, characterizes the value of K that minimizes $C_{\text{Nudge-K}}$. The optimal value is generally neither $K = 1$ nor $K = \infty$. This reflects the fact that while short jobs should get some priority, giving them too much priority hurts long jobs, and thereby the tail constant $C_{\text{Nudge-K}}$.

Concurrently with this work, Charlet and Van Houdt [9] introduce the Nudge-M variant and show several results about it. The most important of these is that for any given split between small and large jobs, Nudge-M with the optimal value of K achieves the *minimum possible tail constant* out of any variant of Nudge. That is, Nudge-M is strongly tail-optimal among Nudge policies. Charlet and Van Houdt [9] also characterize the value of K that leads to this minimal $C_{\text{Nudge-M}}$, showing that it coincides with the value that minimizes $C_{\text{Nudge-K}}$.

While Nudge is significant due to its improving upon FCFS, there are two reasons to believe that Nudge can also be improved upon. First, it seems likely that it would help to have finer-grained distinctions between job sizes, as opposed to grouping them into just two classes, and it may be beneficial to allow jobs to move many spots in the queue. As an extreme example, if a job were size 0, it would make sense to let it jump straight to the front of the queue. Second, while Nudge makes use of the order in which jobs arrive, it does not make use of the *amounts of time between arrivals*. For instance, suppose a job X arrives before a shorter job Y. If the time between the arrivals is very small, as in Fig. 1.1(b), it may make sense to serve the shorter Y before X. But if there is a long interarrival time between X and Y, as in Fig. 1.1(c), it may make sense to keep X in front of Y, because Y will not have been waiting as long by the time the system starts serving X.



(a) Boost's tail improvement ratio for several job size distributions.



(b) Several policies' tail improvement ratios for an exponential job size distribution.

Fig. 1.2. Empirical performance (higher is better) of Boost, specifically the strongly tail-optimal γ-Boost, (a) on several job size distributions, and (b) compared to two other policies, Nudge (and the K and M variants with optimal parameter K) and SRPT. The plots show *tail improvement ratio* $1 - \mathbf{P}[T_\pi > t]/\mathbf{P}[T_{\text{FCFS}} > t]$ as a function of t. Dotted lines indicate the asymptotic tail improvement ratio $1 - C_\pi/C_{\text{FCFS}}$. The load is $\rho = 0.8$, and the mean job size is $\mathbf{E}[S] = 1$. See Section 6 for additional details on the job size distributions and other simulation parameters.

1.4 Our answer: Boost

Motivated by the limitations of Nudge discussed above, we define *Boost*, a new family of scheduling policies. In the full-information setting where job sizes are known to the scheduler, an instance of Boost is specified by a *boost function* $b : \mathbb{R}_+ \to \mathbb{R}$, where $b(s)$ is called the *boost* of a job of size s. The rough idea is that Boost acts like FCFS, except it pretends that a job of size s arrives $b(s)$ time earlier than it actually does. Specifically, if a job of size s arrives at time a, we define its *boosted arrival time* to be

$$\text{boosted arrival time} = \text{arrival time} - \text{boost} = a - b(s).$$

Boost then follows one scheduling rule: *prioritize jobs from least to greatest boosted arrival time*. See Figs. 1.1(b) and 1.1(c) for an illustration. Notice that Boost, unlike Nudge, takes into account not just the arrival order but also the arrival times.

One can define preemptive and nonpreemptive versions of Boost, depending on whether the priority rule is applied at every moment in time or only when a job completes. The distinction turns out not to affect Boost's tail asymptotics, so our results apply to both versions.

The boost function b determines how Boost balances the tension between prioritizing short jobs and prioritizing jobs that have been waiting a long time. For example, setting $b(s) = 0$ reduces the policy to FCFS, whereas setting $b(s) = r/s$ for a large constant r results in prioritizing jobs nearly entirely based on their size, similar to SRPT. We therefore ask: what boost function is best?

We prove two main theoretical results about Boost. First, we find an *explicit formula* for its tail constant C_{Boost} in terms of the boost function b (Theorem 3.1). Second, we study a particular version of Boost, which we call γ*-Boost*, where the boost function is

$$b_\gamma(s) = \frac{1}{\gamma} \log \frac{1}{1 - \exp(-\gamma s)}. \tag{1.2}$$

We show that γ-Boost is *strongly tail-optimal*, meaning $C_{\gamma\text{-Boost}} \le C_\pi$ for every other scheduling policy π (Theorem 5.1). This solves the open problem of finding a strongly tail-optimal policy, as well as the problem of characterizing the best possible tail constant $\inf_\pi C_\pi$.

Of course, strong tail optimality is a theoretical property, and one would hope that pursuing it as an objective yields a policy with good practical performance. We confirm via simulation that this is indeed the case for γ-Boost. Observe in Fig. 1.2 that γ-Boost's improvement over FCFS is often even better than one would predict from the asymptotic tail constants.

Above, we have focused on the case of full job size information, but Boost and γ-Boost can also be defined for systems with partial job size information. The specific partial-information model we consider has multiple types of jobs, each with a distinct *label*, and the scheduler knows each job's label, but not its size. In this partial-information setting, a job's boost is a function of its label rather than its size. Our analysis of Boost's tail constant also applies to the partial-information setting (Theorem 3.1). We also show that partial-information γ-Boost achieves better tail constant than any other Boost policy (Appendix C) and the previous state-of-the-art, Nudge-M (Appendix D).

1.5 Key idea: relate strong tail optimality to an easier scheduling problem

Where does the boost function in (1.2) come from, and how does one show that the resulting γ-Boost policy is strongly-tail optimal? Our key idea is to relate the problem of minimizing the tail constant C_π to a more traditional scheduling problem involving a type of weighted cost.

We begin by considering the following alternative characterization of C_π, which follows from final value theorem [21, Theorem 4.3]:

$$C_\pi = \lim_{\theta \to \gamma} \frac{\gamma - \theta}{\gamma} \mathbf{E}[\exp(\theta T_\pi)].$$

There is thus a vague sense in which minimizing C_π is equivalent to minimizing $\mathbf{E}[\exp(\gamma T_\pi)]$. This is only an informal statement because, as one can deduce from (1.1), we have $\mathbf{E}[\exp(\gamma T_\pi)] = \infty$ for all policies π, even those that are weakly tail-optimal.

While minimizing the always-infinite quantity $\mathbf{E}[\exp(\gamma T_\pi)]$ is not a well-posed problem in the M/G/1, it is analogous to a well-posed problem in *deterministic single-machine scheduling* [27, 33]. Consider an arbitrary finite batch of jobs $\mathcal{I} = \{(a_1, s_1), \ldots, (a_n, s_n)\}$. Here a_i is the arrival time of job i, and s_i is its size. Additionally, let $d_{\pi,i}$ be the departure time of job i under policy π, and let the *θ-cost* of policy π be $K_\pi(\theta, \mathcal{I}) = \sum_{i=1}^n \exp(\theta(d_{\pi,i} - a_i))$. Minimizing $\mathbf{E}[\exp(\gamma T_\pi)]$ is analogous to minimizing γ-cost $K_\pi(\gamma, \mathcal{I})$ in the deterministic setting.

For $\theta < 0$, minimizing θ-cost is actually a variation of a classic single-machine scheduling problem: minimizing total weighted discounted completion time [27, 33], where job i's weight is $\exp(-\theta a_i)$. This problem is hard, but only because of arrival times. In the *batch relaxation*, in which we allow job i to be served even before time a_i, the optimal policy is an index policy called *Weighted Discounted Shortest Processing Time* (WDSPT) [33, Theorem 3.1.6]. To clarify, the arrival times a_i still matter in the batch relaxation, because they determine the weights $\exp(-\theta a_i)$.

Because $\gamma > 0$, one can view minimizing γ-cost as an instance of minimizing total weighted discounted completion time, but with a *negative discount rate*. To the best of our knowledge, this variant of the problem has not been considered in the literature. Nevertheless, essentially the same proof as in the standard positive-discount case shows that a version of WDSPT is optimal in the negative-discount case.[2] The γ-Boost policy arises from finding a function b_γ such that WDSPT is equivalent to serving jobs in order of increasing boosted arrival time $a_i - b_\gamma(s_i)$.

[2]See, for instance, the interchange argument in Pinedo [33, Theorem 3.1.6]. We believe this result may be folklore, but we sketch a proof in Section 4 for completeness.

Above, we have focused on the full-information case, but nearly the same reasoning works in the partial-information case. The difference is that we base the optimal boost function on Weighted Discounted Shortest *Expected* Processing Time (WDSEPT) [33, Theorem 10.1.3] instead of WDSPT.

1.6 Technical challenge: translating from the batch relaxation to the M/G/1

The fact that γ-Boost minimizes γ-cost in the batch relaxation is a promising sign that it is strongly tail-optimal, meaning $C_{\gamma\text{-Boost}} = \inf_\pi C_\pi$, in the M/G/1. There are two significant obstacles between this intuition and a proof of γ-Boost's strong tail optimality.

The first obstacle is that the batch relaxation allows jobs to be served at any time, whereas in the M/G/1, jobs cannot be served before they arrive. We therefore need to show that adjusting γ-Boost's schedule from the batch relaxation to not serve jobs before they arrive does not significantly degrade its performance. If we consider an arbitrarily long sequence of arrivals, this might not be true, so the first step is to figure out how to split up the M/G/1's infinite sequence of arrivals into finite batches. It turns out that using *busy periods* as batches works well. The main technical challenge then becomes showing that "honest" γ-Boost, which only serves jobs after they arrive, is nearly as good as "cheating" γ-Boost, which is allowed to serve any job in the current busy period, even if it has not arrived yet.

The second obstacle is that minimizing steady-state mean γ-cost $\mathbf{E}[\exp(\gamma T_\pi)]$ is not a well-posed problem in the M/G/1, because the expectation is infinite for all policies. Instead, we must make do with the fact that for any weakly tail-optimal policy π, mean θ-cost $\mathbf{E}[\exp(\theta T_\pi)]$ is finite for all $\theta < \gamma$. We therefore work with $\theta \to \gamma$ limits of θ-cost throughout the paper, as opposed to working directly with γ-cost. The main technical challenge is to show that γ-Boost is near-optimal for minimizing not just mean γ-cost but also mean θ-cost, provided θ is close enough to γ.

Above, we have focused on the full-information case, and for good reason: we have not been able to generalize part of this argument to the partial-information case. The issue has to do with a subtle difference between the traditional stochastic batch setting [33, Section 10.1], which assumes independent job sizes, and the instances that arise from busy periods, which can have subtle dependencies between jobs' sizes (Appendix B). Nevertheless, we show γ-Boost outperforms all other versions of Boost (Appendix C) and Nudge-M (Appendix D) in the partial-information setting.

1.7 Contributions

In this work, we present the *first strongly tail-optimal scheduling policy*, namely γ-Boost, for the M/G/1 with light-tailed job size distributions. This solves a significant open problem in queueing and scheduling theory. We also study Boost more generally in both theory and simulation, making the following specific contributions:

- (Section 2) We propose *Boost*, a new family of scheduling policies that balance the tradeoff between prioritizing short jobs and prioritizing jobs that have been waiting a long time.
- (Section 3) We theoretically analyze Boost, giving an explicit formula for its tail constant C_{Boost} in terms of the boost function used (Theorem 3.1).
- (Section 4) We draw a new connection between minimizing the tail constant in the M/G/1 and a batch scheduling problem with *negative discounting*. We solve the batch scheduling problem using *γ-Boost*, a specific instance of Boost.
- (Section 5) In the full-information setting, we prove γ-Boost is *strongly tail-optimal* in the M/G/1 with light-tailed job size distribution (Theorem 5.1).
- (Section 6) We show in simulation that γ-Boost has excellent practical performance, improving upon FCFS's tail performance by more than 50% in some cases. We observe that γ-Boost's performance is robust to using the wrong value of γ or noisily estimated job sizes.

We also make an observation about strong tail optimality in the heavy-tailed case in Appendix A, though it follows nearly immediately from known results.

2 SYSTEM MODEL AND BOOST POLICIES

We consider an M/G/1 queue with arrival rate λ, job size distribution S, and load $\rho = \lambda \mathbf{E}[S]$. We make the standard assumption that $\rho < 1$, ensuring stability, and we assume that $S > 0$ almost surely to avoid trivial jobs of size 0. We assume that S is light-tailed, considering the following specific class of light-tailed distributions initially identified by Abate et al. [1].

Definition 2.1. A distribution S is *class I* if its moment generating function's leftmost singularity

$$\theta^* = \sup\{\theta \in \mathbb{R} \mid \mathbf{E}[\exp(\theta S) < \infty]\},$$

which may be ∞, satisfies $\theta^* > 0$ and $\lim_{\theta \to \theta^*} \mathbf{E}[\exp(\theta S)] = \infty$. In informal discussion, "light-tailed" is understood to refer to class I unless otherwise stated.

The main metric we are concerned with is *response time*, the amount of time between a job's arrival and departure. We denote the response time distribution under scheduling policy π by T_π. Thanks to the "PASTA" property of Poisson arrivals [47], we can interpret T_π as the response time of a random "tagged" job arriving to a steady-state system. We discuss the details of the scheduling policies π we consider in Section 2.2.

The quantity that has the largest impact on the system's response time is the *work*, the total remaining processing time of jobs currently in the system. We denote the steady-state amount of work in the M/G/1 by W. This amount is the same under all non-idling (aka work-conserving) scheduling policies.

Our main objective is to find the strongly tail-optimal scheduling policy, defined in Section 1.1 and recalled below.

Definition 2.2. A scheduling policy π is *weakly tail-optimal* if there exists finite $c \geq 1$ such that

$$\sup_{\pi'} \limsup_{t \to \infty} \frac{\mathbf{P}[T_\pi > t]}{\mathbf{P}[T_{\pi'} > t]} = c,$$

If additionally $c = 1$, we say π is *strongly tail-optimal*.

The supremum in Definition 2.2 ranges over all preemptive scheduling policies π' that have access to full information about the sizes and arrival times of all arrivals. In particular, π' could in principle use information about *future* arrivals. However, none of the policies we consider use this information (aside from the "cheating" policy introduced in Section 2.4), and we achieve strong tail optimality without it.

2.1 Asymptotic tails

The key property of class I distributions is that they ensure that the work W has asymptotically exponential tail. Specifically, there exist constants $\gamma > 0$ and $C_W > 0$ such that [21, equation (2)][3]

$$C_W = \lim_{t \to \infty} \exp(\gamma t) \mathbf{P}[W > t] = \lim_{\theta \to \gamma} \frac{\gamma - \theta}{\gamma} \mathbf{E}[\exp(\theta W)], \tag{2.1}$$

with the equivalence of the two limits being due to final value theorem [21, Theorem 4.3]. We call γ the *decay rate* and C_W the *tail constant* of the work distribution W. It is known that γ is the least positive real solution to

$$\gamma = \lambda(\mathbf{E}[\exp(\gamma S)] - 1). \tag{2.2}$$

[3]Throughout the paper, $\theta \to \gamma$ limits are understood as being limits from below, seeing as $\mathbf{E}[\exp(\theta W)] = \infty$ for all $\theta \geq \gamma$.

When S is class I, γ is a simple pole of W's moment generating function $\theta \mapsto \mathbf{E}[\exp(\theta W)]$ [1, 2], regardless of the arrival rate λ. Our results likely generalize to other combinations of λ and S for which this is the case.

We define the *tail constant of scheduling policy* π, denoted C_π, in the same way as the tail constant of the work distribution:[4]

$$C_\pi = \lim_{t\to\infty} \exp(\gamma t)\mathbf{P}[T_\pi > t] = \lim_{\theta\to\gamma} \frac{\gamma-\theta}{\gamma}\mathbf{E}[\exp(\theta T_\pi)]. \tag{2.3}$$

As an example, FCFS's tail constant is easily shown to be

$$C_{\text{FCFS}} = C_W \mathbf{E}[\exp(\gamma S)],$$

where finiteness of $\mathbf{E}[\exp(\gamma S)]$ follows from (2.2).

2.2 Scheduling model and what information the scheduler has

We consider both *nonpreemptive* scheduling, where once a job begins service, it will complete without interruption, and *preemptive* scheduling, where jobs may be paused in the middle of service. In the latter case, we assume a standard preempt-resume model in which jobs may be paused and resumed without delay, overhead, or loss of progress.

We wish to study both the *full-information* setting, in which the scheduler learns each job's exact size (aka service time) when it arrives; as well as *partial-information* settings, in which the scheduler has some limited but incomplete information about job sizes. For instance, perhaps there are two types of arrivals, each with its own size distribution, but we do not know the size of any particular job.[5]

To capture a wide range of information settings, we use the flexible *label-size pair* model from recent work on M/G/1 scheduling [34, 38]. In this model, each job has an i.i.d. pair (L, S) of a *label* L and size S. The space of possible labels, denoted $\mathbb{L}$, can be arbitrary, and there may be an arbitrary joint distribution between labels and sizes. For example:

- To model known job sizes, let $\mathbb{L} = \mathbb{R}_+$ and $L = S$.
  - We call this case the *full-information* setting.
- To model a scenario with two types of jobs A and B, where job types are known but job sizes are unknown, let $\mathbb{L} = \{\text{A}, \text{B}\}$, and define the joint distribution such that, for instance, $(S \mid L = \text{A})$ is the size distribution of type A jobs.
  - We call any case where $L < S$ with positive probability, of which the above is one example, the *partial-information* setting.

Of course, one can imagine more complicated label-size pair distributions. As a final example, perhaps some jobs are labeled with their exact size, while others are labeled only type A or type B. This can be modeled using $\mathbb{L} = \mathbb{R}_+ \cup \{\text{A}, \text{B}\}$.

We assume that the scheduler has access to each job's arrival time and label, but that it has no information about each job's size beyond what can be deduced from its label. That is, if a job is labeled l, the scheduler knows its size is distributed as $(S \mid L = l)$, but it does not learn the realization until the job is complete.

[4]The limits below may not exist, so strictly speaking, we should define lower and upper constants using $\liminf_{t\to\infty}$ and $\limsup_{t\to\infty}$ in place of $\lim_{t\to\infty}$. But the limits exist for all policies we consider, so we omit this additional complexity.

[5]If there is no information at all to distinguish different jobs from one another, then, at least among nonpreemptive policies, there is no way to improve upon FCFS, due to T_{FCFS} being minimal in the convex order [41]. Investigating whether preemptive policies could improve upon FCFS in this setting is an interesting future direction.

2.3 Defining the Boost family of policies

We introduce a new scheduling policy called *Boost*. Strictly speaking, Boost is a family of scheduling policies, where an instance of the family is determined by a *boost function* $b : \mathbb{L} \to \mathbb{R}_+$.[6] The boost function maps each label $l \in \mathbb{L}$ to a quantity $b(l)$ called the *boost* of a job with label l.

Boost operates as follows. Suppose a job with label l has arrival time a. We define the job's *boosted arrival time* to be $a - b(l)$. Boost uses the same basic rule with any boost function: *prioritize jobs in order from least to greatest boosted arrival time.*[7] As a trivial example, choosing $b(l) = 0$ reduces Boost to FCFS.

One can define preemptive or nonpreemptive versions of Boost. The preemptive version makes scheduling decisions continuously, always serving the job of least boosted arrival time. The nonpreemptive version, after it serves the first job in each busy period, makes scheduling decisions whenever a job completes, each time choosing the job of least boosted arrival time. One can also define intermediate versions where a job may be preempted by some, but not necessarily all, arrivals with lower boosted arrival time.

All of our theoretical results hold for the preemptive, nonpreemptive, and intermediate versions of Boost (Remark 3.4). As such, we leave the exact preemption rule unspecified throughout our theoretical results. But for concreteness, the reader may safely imagine that nonpreemptive Boost is used throughout, and we use nonpreemptive Boost in our simulations (Section 6).

There is one family of boost functions that is especially important, as they result in strong tail optimality.

Definition 2.3. For any $\theta > 0$, the *θ-Boost* policy for label-size pair distribution (L, S) is the version of Boost with the following boost function, which we call the *θ-optimal boost function*:

$$b_\theta(l) = \frac{1}{\theta} \log \frac{\mathbf{E}[\exp(\theta S) \mid L = l]}{\mathbf{E}[\exp(\theta S) \mid L = l] - 1}.$$

While the definitions of θ-Boost and b_θ depend on the label-size pair distribution (L, S), we leave this implicit in our notation.

The γ-Boost policy alluded to in Section 1 is simply θ-Boost with $\theta = \gamma$. In the full-information case where $L = S$, we have $\mathbf{E}[\exp(\gamma S) \mid L = s] = \exp(\gamma s)$, and so b_γ reduces to the formula in (1.2).

We use the name "Boost" when referring to a version with generic boost function b, and we use the name "θ-Boost" when referring to a version using the θ-optimal boost function b_θ, with $\theta = \gamma$ being the most important case.

If one uses an overly aggressive boost function, such as boosting small jobs too much, then Boost may not be weakly tail-optimal, let alone strongly tail-optimal. Our results (Theorem 3.1) show that as long as

$$\mathbf{E}[b(L)(\exp(\gamma S) - 1)] < \infty, \tag{2.4}$$

then Boost is indeed weakly tail-optimal. There are two important special cases where (2.4) holds.

- It always holds if one uses the γ-optimal boost function b_γ (Lemma 5.3).
- In the full-information case where $L = S$, one can show using finiteness of $\mathbf{E}[\exp(\gamma S)]$ that (2.4) holds if $b(s) \le O(1/s)$ in the $s \to 0$ limit and $b(s) \le O(1)$ in the $s \to \infty$ limit. The intuition behind why we need such a condition is that if small jobs have too large of a boost, then each large job could be overtaken by so many small jobs that we lose weak tail optimality.

[6] One can in principle allow negative boosts, but we focus our analysis on the case where boosts are always nonnegative.
[7] To clarify, at any moment in time, Boost is only aware of jobs whose arrival time a is in the past. There may be jobs that will arrive in the future with boosted arrival time $a - b(l)$ in the past. But Boost is not aware of these jobs yet and will not serve them prior to their arrival. Boost thus does not require knowledge of future arrival times to implement.

To reduce clutter, we let $B = b(L)$ be the boost of a random job, meaning we have a joint distribution of boost-size pairs (B, S). We write B_θ when using the θ-optimal boost function b_θ.

2.4 Lower bounding tool: "cheating" version of Boost

In order to prove that γ-Boost is strongly tail-optimal, we need a lower bound on the possible tail constant C_π (Section 2.1) achievable by any policy π. Our main tool for doing so is to define a "cheating" version of Boost, which we call *Cheat*.

Like Boost, Cheat is defined by a boost function b, it assigns each job a boosted arrival time in the same way, and it also prioritizes jobs in order from least to greatest boosted arrival time. The difference is that we allow Cheat to *serve arrivals from the future*. Specifically, we allow Cheat to serve any job that will arrive in the current busy period, where a *busy period* is a maximal interval of time during which the server is busy [22].[8]

We can view Cheat as being essentially the same policy as Boost for a modified "cheating" M/G/1, which differs from the standard M/G/1. To describe the cheating M/G/1, we distinguish between a job's *arrival time*, which is the time it arrives in the standard system, and a job's *release time*, which is the earliest moment in time at which a job is allowed to be served by the scheduler.

- In the standard M/G/1, a job's release time is its arrival time.
- In the cheating M/G/1, a job's release time is the start of the busy period containing its arrival time.

In the cheating M/G/1, we still define a job's response time to be its departure time minus its arrival time. The difference is that a job's departure time may now be less than its arrival time plus its size, so a job's response time may be less than its size, or even negative.

To clarify, for a given arrival sequence, the standard and cheating M/G/1 systems have the same busy periods. That is, one can imagine first deciding what the busy periods are using the standard M/G/1, and then "retroactively" moving release times to construct the corresponding cheating M/G/1. Note that this implies the cheating M/G/1 is ergodic, because it inherits the renewal cycles of the standard M/G/1.

With the above distinction in mind, strictly speaking, Cheat is best thought of as the same policy as Boost but for a modified system, namely the cheating M/G/1, as opposed to a different scheduling policy for the standard system. But for the purposes of notation, we treat it like a different scheduling policy. For instance, we denote the response time distribution of Cheat by T_{Cheat}. As a reminder, due to how the cheating system works, we can have $T_{\text{Cheat}} < 0$ with positive probability.

We define *θ-Cheat* analogously to θ-Boost (Definition 2.3), namely as the version of Cheat using the θ-optimal boost function b_θ. The significance of θ-Cheat is that, as we show in Theorem 4.3, $\mathbf{E}[\exp(\theta T_{\theta\text{-Cheat}})] \leq \mathbf{E}[\exp(\theta T_\pi)]$ for any policy π for the standard M/G/1.

3 ANALYSIS OF BOOST'S TAIL CONSTANT

In this section, we analyze the tail constants of Boost and Cheat in terms of the boost function b and the system model parameters, namely λ and (L, S).

Our main result is the following characterization of C_{Boost} and C_{Cheat}. Interestingly, we find that cheating has no effect on the tail constant. See Section 3.3 for the proof.

Theorem 3.1. *Consider an M/G/1 with class I job size distribution, and consider the Boost policy with a fixed boost function b. If* (2.4) *holds, then Boost and Cheat both have tail constant*

$$C_{\text{Boost}} = C_{\text{Cheat}} = C_W \mathbf{E}\big[\exp(\gamma(S - b(L)))\big] \exp\big(\lambda \mathbf{E}[b(L)\,(\exp(\gamma S) - 1)]\big).$$

In particular, $C_{\text{Boost}} < \infty$, so Boost is weakly tail-optimal.

[8]This definition is assuming a non-idling scheduling policy, where all such policies lead to the same notion of busy periods.

3.1 Approach: tagged job analysis

To derive the tail constants of Boost and Cheat, we require bounds on T_{Boost} and T_{Cheat}. We obtain these by considering a pair of M/G/1 systems, one standard and one cheating, experiencing the same arrival process. The only difference is that the release times in the standard M/G/1 coincide with arrival times, whereas in the cheating M/G/1, all jobs in a given busy period have release time at the start of the busy period (Section 2.4). We assume both systems are stationary processes.

We use a *tagged job* analysis [22], which is a common technique in analyzing complex scheduling policies [9, 11, 12, 17, 18, 21, 34, 36, 38, 43, 45]. A typical tagged job analysis uses the observation that, due to the "PASTA" property of Poisson arrivals [47], to analyze a policy π's response time distribution T_π, we can analyze the response time of a single "tagged" job which arrives at an arbitrary time, such as time 0. We then interpret T_π as the random response time the tagged job experiences. There are three *independent* sources of randomness that contribute to T_π:

- The tagged job's label-size pair, which is drawn from (L, S).
- Aspects of the M/G/1's state, such as its work W, at time 0, which is drawn from the system's stationary distribution.
- Arrivals that occur after time 0.

In our approach, instead of assuming without loss of generality that the tagged job's arrival time is time 0, we assume the *boosted arrival time* is time 0. The tagged job's response time is still determined by the same three sources of randomness listed above. However, the interpretation of the latter two sources changes, e.g., some of the arrivals after time 0 arrive before the tagged job.

The only subtlety to check is that the system state at time 0, and in particular the work W, still has the stationary distribution. This is indeed the case. Consider the stationary work process W_t as a function of time $t \in \mathbb{R}$. The key observation is that the tagged job's boost is independent of its arrival time. So if the tagged job arrives at time a and has boost $B = b(L)$, the work at the boosted arrival time W_{a-B} is distributed according to the stationary distribution, because $\{W_t\}_{t \le a}$ is stationary and independent of B. As such, we can imagine $a - B = 0$ without loss of generality. One can use the framework of Palm calculus [4] to formalize this argument.

To summarize our approach and notation:

- We analyze the response time of a tagged job with boosted arrival time 0.
  - Abusing notation slightly, we write L, S, $B = b(L)$, and T_π for the tagged job's label, size, boost, and response time under policy π.
  - This means the tagged job's arrival time is B.
- The amount of work in the (standard) M/G/1 at time 0, denoted W, is distributed according to the standard stationary work distribution [22].
  - We follow the convention that W does not include the tagged job's size S.
- After time 0, new Poisson arrivals occur at rate λ with iid label-size pairs.
  - To avoid ambiguity, when we need to discuss the label, size, and boost of a generic future arrival, we write L', S', and B' instead of (the identically distributed) L, S, and B.
  - We call these arrivals "new" because their arrival times are after time 0, even if their arrival times are before the tagged job's arrival time B.

3.2 Bounding Boost's response time using crossing work

A critical step in bounding both T_{Cheat} and T_{Boost} is quantifying how much arriving work will "boost past" the tagged job, which we define formally below. We also define the complementary quantity for the arriving work that doesn't boost past the tagged job.

Definition 3.2.

(a) The *crossing work* arriving in $(0, u)$, denoted $V(u)$, is the amount of work due to jobs that have arrival time in $(0, u)$ and boosted arrival time in $(-\infty, 0]$.
(b) The *non-crossing work* arriving in $(0, u)$, denoted $\bar{V}(u)$, is the amount of work due to jobs that have arrival time in $(0, u)$ and boosted arrival time in $(0, \infty)$.

For example, $V(\infty)$ is all the work that "boosts past" time 0, meaning arriving after 0 but having boosted arrival time before 0. On the other extreme, $V(0)$ is simply 0.

Understanding the amount of crossing work that the tagged job experiences is key to understanding the response time of the tagged job under Boost. However, the exact crossing work is difficult to compute, as it depends on the amount of work W at the tagged job's boosted arrival time, its size, as well as the sizes and labels of future arrivals. In particular, jobs with boosted arrival time before time 0 may still depart after the tagged job, if their arrival times are late enough. Under Cheat, since the release time of all jobs is the beginning of the busy period, the crossing work is given by $V(Z)$, where Z is the random time denoting the end of the busy period. But Z in turn depends on the same arrivals that $V(Z)$ is counting, making this a hard quantity to analyze.

Due to these difficulties, instead of computing crossing work exactly, we find bounds on T_{Boost} and T_{Cheat} which are good enough for computing C_{Boost} and C_{Cheat}.

Lemma 3.3. *The tagged job's response times under Boost and Cheat are both lower-bounded by*

$$T_{\text{Boost}}, T_{\text{Cheat}} \geq W - B + V(W) + S$$

and, for all $u \geq 0$, upper-bounded by

$$T_{\text{Boost}}, T_{\text{Cheat}} \leq (W - \min\{B, u\})^+ + V(\infty) + S + \bar{V}(u)\, \mathbb{1}(W < \min\{B, u\}).$$

Proof. We first recall our conventions from Section 3.1. The tagged job's boosted arrival time is 0, true arrival time is B, and departure time is $T_\pi + B$. There is W work in the system at time 0.

We first treat both bounds on T_{Boost} before handling both bounds on T_{Cheat}. For T_{Boost}, we analyze the work done by the server between the tagged job's boosted arrival time 0 and its departure time $T_{\text{Boost}} + B$. Both the lower and upper bounds use the fact that the server must complete the following work between 0 and $T_{\text{Boost}} + B$:

- W from work present at time 0. This must be completed because all the work present at time 0 has priority over the tagged job. This is because their arrival times, and thereby boosted arrival times, are earlier than 0.
- S from the tagged job itself.
- Some additional work from new arrivals that occur after time 0.

The main task is thus to bound the work from new arrivals.

Lower bound on T_{Boost}. By the first bullet above, work on W must complete before the tagged job can enter service. Therefore, it cannot enter service before time W. In this time, there is at least $V(W)$ work from new arrivals that will have priority over the tagged job. Adding the required components, we have $T_{\text{Boost}} + B \geq W + V(W) + S$, as desired.

Upper bound on T_{Boost}. First, observe that decreasing the tagged job's boost from B to $\hat{B} = \min\{B, u\}$ can only increase its response time, so it suffices to analyze the case with this reduced boost. In the remainder of this argument, $\hat{B}$ plays the role of B.

We first consider the fully preemptive case, in which the tagged job has priority over all jobs with boosted arrival time later than 0. In addition to the required completions of W and S before the tagged job's departure, it will need to complete at most $V(\infty)$ from new arrivals that occur after time 0.

However, we must also account for time the server spends idle. No idling occurs after the tagged job arrives, but the server may be idle for some time during $[0, \hat{B}]$. This idle time is at most $(\hat{B} - W)^+$, so in the fully preemptive case,

$$T_{\text{Boost}} + \hat{B} \le W + V(\infty) + S + (\hat{B} - W)^+. \tag{3.1}$$

We now turn to the case where the server is not fully preemptive. The only change to the argument is that the tagged job may also have to wait for the remaining work of a lower-priority job, if one is in service when the tagged job arrives at time $\hat{B}$. If there is such a job, let R be its total size, and otherwise, let $R = 0$. It is clear that the tagged job's departure time is at most the right-hand side of (3.1) plus R. We bound R with two observations:

- If $W \ge \hat{B}$, then at time $\hat{B}$ when the tagged job arrives, the server is still working on jobs with priority over the tagged job, either from W or from new arrivals. This means $R > 0$ only if $W < \hat{B}$.
- If $R > 0$, then a job with boosted arrival time after 0 is in service at time $\hat{B}$. The job's true arrival time is therefore in $(0, \hat{B})$, so $R \le \bar{V}(\hat{B}) \le \bar{V}(u)$ (Definition 3.2).

Combining these observations implies

$$R \le \bar{V}(u)\, \mathbb{1}(W < \hat{B}).$$

Adding this to the fully preemptive bound in (3.1) yields

$$T_{\text{Boost}} + \hat{B} \le W + V(\infty) + S + (\hat{B} - W)^+ + \bar{V}(u)\, \mathbb{1}(W < \hat{B}),$$

which rearranges to the desired bound.

Lower bound on T_{Cheat}. Recall throughout that W refers to the amount of work in the standard M/G/1 at time 0, whose busy periods affect the release times in the cheating M/G/1 (Section 2.4). We will show that the tagged job cannot begin service before at least time $W + V(W)$, which implies the desired bound. To do this, we first analyze the busy period related to W, the work present at time 0 in the system. This work belongs to a busy period, which we call BP, that started at time $-A < 0$. Let U be the amount of work from jobs with arrival time in $[-A, 0)$. Then we have

$$U - A = W.$$

Therefore, the server will be busy during $[-A, -A + U] = [-A, W]$. As such, at least $V(W)$ work from new arrivals will arrive during $[0, W]$. This means there is at least $U + V(W)$ total work in the busy period with boosted arrival time at most 0.

There are now two cases to consider for the tagged job. First, suppose the system is busy for all of $[0, B]$. Then the tagged job belongs to the same busy period BP as described above. In this period we know that there is at least $U + V(W)$ total work with priority over the tagged job, so it cannot begin service prior to

$$-A + U + V(W) = W + V(W).$$

Second, suppose that the system becomes idle at some time in $[0, B]$. This means that the tagged job belongs to a busy period after BP. Therefore the tagged job's release time, which is the earliest it can enter service, must be after the end of BP. But we know that BP ends at the earliest at $-A + U + V(W) = W + V(W)$, so the tagged job's release time must be at least $W + V(W)$.

Upper bound on T_{Cheat}. First, as in the T_{Boost} upper bound, we reduce the tagged job's boost to $\hat{B} = \min\{B, u\}$. We then imagine a variant of Cheat that treats the tagged job especially poorly, forcing it to begin service at no earlier than its arrival time $\hat{B}$. This clearly only increases the tagged job's response time. From here, the reasoning from the fully preemptive T_{Boost} upper bound also applies to T_{Cheat}.

Remark 3.4. The proof of Lemma 3.3 above works regardless of whether we are considering a preemptive, nonpreemptive, or intermediate version of Boost. As such, our results apply regardless of the precise preemption rule used.

Lemma 3.3 allows us to bound both T_{Boost} and T_{Cheat} with quantities that only depend on the standard M/G/1. It then suffices to show that the lim inf and lim sup of the lower and upper bounds, respectively, converge to the same number, namely the expression given in Theorem 3.1. To do so, we require that the crossing work terms in our bounds have finite moment generating function. This holds under our assumption (2.4), which says, roughly speaking, that the boost isn't too large for too many labels.

Lemma 3.5. *If* (2.4) *holds, then for all* $u \in \mathbb{R}_+ \cup \{\infty\}$,

$$\mathbf{E}[\exp(\theta V(u))] = \exp\big(\lambda \mathbf{E}\big[(\exp(\theta S') - 1)\min\{B', u\}\big]\big) < \infty.$$

Proof. We can consider each new arrival to be a triple (b', s', t) representing its boost b', size s', and arrival time t. Let X be the random set of triples corresponding to arrivals after time 0. We can write the crossing work $V(u)$ as

$$V(u) = \sum_{(b',s',t)\in X} s' \mathbb{1}(t \le \min\{b', u\}).$$

To compute $\mathbf{E}[\exp(\theta V(u))]$, we use Campbell's theorem [25, Section 3.2] for the Laplace functional of a Poisson point process. In our case, the point process is X, and its intensity measure is

$$\mu(\mathcal{B} \times \mathcal{S} \times \mathrm{d}t) = \mathbf{P}[B' \in \mathcal{B}, S' \in \mathcal{S}]\, \lambda\, \mathrm{d}t.$$

Campbell's theorem and a brief computation involving Tonnelli's theorem then imply[9]

$$\begin{aligned}
\mathbf{E}[\exp(\theta V(u))] &= \exp\left(\int \big(\exp\big(\theta s' \mathbb{1}(t \le \min\{b', u\})\big) - 1\big)\, \mu(\mathrm{d}(b', s', t))\right) \\
&= \exp\left(\int (\exp(\theta s') - 1)\, \mathbb{1}(t \le \min\{b', u\})\, \mu(\mathrm{d}(b', s', t))\right) \\
&= \exp\left(\mathbf{E}\left[\int_0^\infty (\exp(\theta S') - 1)\, \mathbb{1}(t \le \min\{B', u\})\, \lambda\, \mathrm{d}t\right]\right) \\
&= \exp\big(\lambda \mathbf{E}\big[(\exp(\theta S') - 1)\min\{B', u\}\big]\big).
\end{aligned}$$

3.3 Tail Constant of Boost

Proof of Theorem 3.1. Let π be one of Boost or Cheat, and let the claimed tail constant be

$$C = C_W \mathbf{E}\big[\exp(\gamma(S - B))\big] \exp\big(\lambda \mathbf{E}[B(\exp(\gamma S) - 1)]\big) = C_W \mathbf{E}\big[\exp(\gamma(S - B))\big]\, \mathbf{E}[\exp \gamma V(\infty)],$$

where Lemma 3.5 implies the second equality. We will show $C_\pi = C$. By Lemma 3.3, for all $u \ge 0$,

$$\begin{aligned}
&\mathbf{E}[\exp(\theta T_\pi)] \ge \mathbf{E}\big[\exp\big(\theta\big(W - B + V(W) + S\big)\big)\big], \\
&\mathbf{E}[\exp(\theta T_\pi)] \le \mathbf{E}\big[\exp\big(\theta\big((W - \min\{B, u\})^+ + V(\infty) + S + \mathbb{1}(W < \min\{B, u\})\bar{V}(u)\big)\big)\big].
\end{aligned}$$

Equation (2.3) thus implies

$$\begin{aligned}
&\gamma C_\pi \ge \liminf_{\theta\to\gamma} (\gamma - \theta)\mathbf{E}\big[\exp\big(\theta\big(W - B + V(W) + S\big)\big)\big], \\
&\gamma C_\pi \le \limsup_{\theta\to\gamma} (\gamma - \theta)\mathbf{E}\big[\exp\big(\theta\big((W - \min\{B, u\})^+ + V(\infty) + S + \mathbb{1}(W < \min\{B, u\})\bar{V}(u)\big)\big)\big].
\end{aligned}$$

[9]The precondition of Campbell's theorem [25, Section 3.2] is satisfied when the right-hand-side expression is finite, as $\exp(\theta s' \mathbb{1}(t \le \min b', u)) - 1 \ge \theta s' \mathbb{1}(t \le \min(b', u))$.

It thus suffices to show that the lower bound is at least γC, and that the infimum over u of the upper bound is at most γC.

We first analyze the lower bound γC_π. For all $w \geq 0$, we have

$$\begin{aligned}
&\liminf_{\theta\to\gamma} (\gamma-\theta)\mathbf{E}\left[\exp\left(\theta\left(W - B + V(W) + S\right)\right)\right] \\
&\quad\geq \liminf_{\theta\to\gamma} (\gamma-\theta)\mathbf{E}[\exp(\theta(W - B + V(W) + S))\mathbb{1}(W > w)] \\
&\quad\geq \liminf_{\theta\to\gamma} (\gamma-\theta)\mathbf{E}[\exp(\theta(W - B + V(w) + S))\mathbb{1}(W > w)] \\
&\quad= \liminf_{\theta\to\gamma} (\gamma-\theta)\mathbf{E}[\exp(\theta W)\mathbb{1}(W > w)]\,\mathbf{E}[\exp(\theta(S - B))]\,\mathbf{E}[\exp(\theta V(w))] \\
&\quad= \gamma C_W \mathbf{E}[\exp(\gamma(S - B))]\,\mathbf{E}[\exp(\gamma V(w))],
\end{aligned}$$

where the last step follows from (2.1). Since this holds for all w, it also holds in the $w \to \infty$ limit. The monotone convergence theorem implies the limit is γC, as desired.

We now turn to the upper bound on γC_π. We have

$$\begin{aligned}
&\limsup_{\theta\to\gamma} (\gamma-\theta)\mathbf{E}\left[\exp\left(\theta\left((W - \min\{B,u\})^+ + V(\infty) + S + \bar{V}(u)\,\mathbb{1}(W < \min\{B,u\})\right)\right)\right] \\
&\quad= \limsup_{\theta\to\gamma} (\gamma-\theta)\mathbf{E}\left[\exp\left(\theta\left(W - \min\{B,u\} + V(\infty) + S\right)\right)\mathbb{1}(W \geq \min\{B,u\})\right] \\
&\qquad+ \limsup_{\theta\to\gamma} (\gamma-\theta)\mathbf{E}\left[\exp\left(\theta\left(V(\infty) + S + \bar{V}(u)\right)\right)\mathbb{1}(W < \min\{B,u\})\right] \\
&\quad\leq \limsup_{\theta\to\gamma} (\gamma-\theta)\mathbf{E}\left[\exp\left(\theta\left(W - \min\{B,u\} + V(\infty) + S\right)\right)\right] \qquad (3.2) \\
&\qquad+ \limsup_{\theta\to\gamma} (\gamma-\theta)\mathbf{E}\left[\exp\left(\theta\left(V(\infty) + S + \bar{V}(u)\right)\right)\right]. \qquad (3.3)
\end{aligned}$$

We now compute the limits in (3.2) and (3.3). We first show the limit in (3.3) vanishes, then we show the limit in (3.2) has the desired value.

Let $A(u) = V(u) + \bar{V}(u) \geq \bar{V}(u)$ be the total amount of work that arrives during $(0, u)$. The limit in (3.3) is bounded by

$$\begin{aligned}
&\limsup_{\theta\to\gamma} (\gamma-\theta)\mathbf{E}\left[\exp\left(\theta\left(V(\infty) + S + \bar{V}(u)\right)\right)\right] \\
&\quad= \limsup_{\theta\to\gamma} \mathbf{E}[\exp(\theta V(\infty))]\,\mathbf{E}[\exp(\theta S)]\,\mathbf{E}[\exp(\theta \bar{V}(u))] \\
&\quad\leq \limsup_{\theta\to\gamma} (\gamma-\theta)\mathbf{E}[\exp(\theta V(\infty))]\,\mathbf{E}[\exp(\theta S)]\,\mathbf{E}[\exp(\theta A(u))] \\
&\quad\leq \limsup_{\theta\to\gamma} (\gamma-\theta)\mathbf{E}[\exp(\gamma V(\infty))]\,\mathbf{E}[\exp(\gamma S)]\,\mathbf{E}[\exp(\gamma A(u))] \\
&\quad= 0,
\end{aligned}$$

where the second line holds by independence of $V(\cdot)$, $\bar{V}(\cdot)$, and S; and the second-to-last line is possible because all three of the factors are finite.

- $\mathbf{E}[\exp(\gamma S)]$ is finite by the definition of γ.
- $\mathbf{E}[\exp(\gamma V(\infty))]$ is finite by (2.4) and Lemma 3.5.
- $\mathbf{E}[\exp(\gamma A(u))] = \exp(\lambda u(\mathbf{E}[\exp(\gamma S)] - 1)) = \exp(\gamma u)$, by a standard M/G/1 result [22] and the definition of γ.

Because the limit in (3.3) vanishes, γC_π is at most the limit in (3.2), so

$$\begin{aligned}\gamma C_\pi &\le \limsup_{\theta\to\gamma} (\gamma-\theta)\mathbf{E}[\exp(\theta(W-\min\{B,u\}+V(\infty)+S))]\\ &= \limsup_{\theta\to\gamma} (\gamma-\theta)\mathbf{E}[\exp(\theta W)]\,\mathbf{E}[\exp(\theta(S-\min\{B,u\}))]\,\mathbf{E}[\exp(\theta V(\infty))]\\ &= \gamma C_W \mathbf{E}[\exp(\gamma(S-\min\{B,u\}))]\,\mathbf{E}[\exp(\gamma V(\infty))],\end{aligned}$$

where the last step follows from (2.1). Because this holds for any u, it holds in the $u\to\infty$ limit. The monotone convergence theorem implies the limit is γC, as desired.

4 A BATCH SCHEDULING PROBLEM RELATED TO TAIL OPTIMALITY

In this section we show that θ-Cheat minimizes $\mathbf{E}[\exp(\theta T_\pi)]$ in the full-information case. The key idea is that in this case, we can treat each busy period as a finite batch of jobs on which to minimize a cost function which corresponds to $\mathbf{E}[\exp(\theta T_\pi)]$. We start by defining what a batch is and what the cost function is.

Definition 4.1. A *batch instance* $\mathcal{I} = \{(a_1,s_1),\dots,(a_n,s_n)\}$ is a finite batch of pairs of arrival times and job sizes.

Definition 4.2. The *θ-cost* of policy π on batch instance $\mathcal{I} = \{(a_1,s_1),\dots,(a_n,s_n)\}$ is

$$K_\pi(\theta,\mathcal{I}) = \sum_{i=1}^n \mathbf{E}[\exp(\theta T_{\pi,i})] = \sum_{i=1}^n \mathbf{E}[\exp(\theta(D_{\pi,i}-a_i))],$$

where $D_{\pi,i}$ is the departure time of job i under policy π. The expectation is over any randomness in the policy.

We now show that θ-Cheat minimizes θ-cost across any finite batch of jobs, across all preemptive policies, and that this in turn minimizes $\mathbf{E}[\exp(\theta T_\pi)]$.

Theorem 4.3. *In the full-information setting, the θ-Cheat policy minimizes θ-cost. Specifically, for any batch instance $\mathcal{I}$,*

$$K_{\theta\text{-Cheat}}(\theta,\mathcal{I}) = \min_\pi K_\pi(\theta,\mathcal{I}), \tag{4.1}$$

and therefore

$$\mathbf{E}[\exp(\theta T_{\theta\text{-Cheat}})] \le \inf_\pi \mathbf{E}[\exp(\theta T_\pi)]. \tag{4.2}$$

Proof. We first show (4.1), namely that θ-Cheat minimizes θ-cost for any batch instance $\mathcal{I}$. This can be seen by observing that θ-Cheat serves jobs in $\mathcal{I}$ in the same order as the Weighted Discounted Shortest Processing Time policy (WDSPT), which is known to minimize θ-cost for any batch instance $\mathcal{I}$. This is because boosted arrival time $a_i - b_\theta(s_i)$ is a monotonic function, namely $1/\theta$ times the negative log, of WDSPT's priority index:

$$a_i - b_\theta(s_i) = -\frac{1}{\theta}\log\left(\exp(-\theta a_i)\frac{\exp(\theta s_i)}{\exp(\theta s_i)-1}\right).$$

The proof is an interchange argument identical to that of Pinedo [33, Theorem 3.1.6], with the signs for the discount rate and objective flipped. Specifically, with negative discounting, we define discounted completion time as $\exp(\theta D_i)-1$ instead of $1-\exp(\theta D_i)$ to keep the sign positive.

Having shown (4.1), we now turn to (4.2). The key idea is to consider busy periods as batch instances. Specifically, by renewal-reward theorem [8, Theorem 10.2.15],

$$\mathbf{E}[\exp(\theta T_\pi)] = \frac{\mathbf{E}[K_\pi(\theta,\mathcal{B})]}{\mathbf{E}[|\mathcal{B}|]},$$

where $\mathcal{B}$ is a random batch instance corresponding to an M/G/1 busy period, and $|\mathcal{B}|$ is the number of jobs in the instance. The intuition is that the average θ-cost per job is the expected total θ-cost of all jobs in a busy period, namely $\mathbf{E}[K_\pi(\theta, \mathcal{B})]$, divided by the expected number of jobs in a busy period, namely $\mathbf{E}[|\mathcal{B}|]$. But $\mathbf{E}[|\mathcal{B}|] = \frac{1}{1-\rho}$ is the same under all scheduling policies, so

$$\begin{aligned}\mathbf{E}[\exp(\theta T_{\theta\text{-Cheat}})] &= \frac{\mathbf{E}[K_{\theta\text{-Cheat}}(\theta, \mathcal{B})]}{\mathbf{E}[|\mathcal{B}|]} \\ &= \frac{\mathbf{E}[\min_\pi K_\pi(\theta, \mathcal{B})]}{\mathbf{E}[|\mathcal{B}|]} \\ &\le \inf_\pi \frac{\mathbf{E}[K_\pi(\theta, \mathcal{B})]}{\mathbf{E}[|\mathcal{B}|]} \\ &= \inf_\pi \mathbf{E}[\exp(\theta T_\pi)].\end{aligned}$$

Remark 4.4. While Theorem 4.3 treats only the full-information setting, an analogue of (4.1) holds in the partial-information setting. The optimality is relative not to all policies, but *nonpreemptive* policies that have access to only labels and arrival times. However, (4.1) no longer implies (4.2) in the partial-information setting. See Appendix B for details.

5 PROOF OF BOOST'S STRONG TAIL OPTIMALITY

In this section, we prove strong tail optimality of γ-Boost in the full-information setting. We do so by computing an explicit lower bound on the optimal tail constant, namely C^* from Theorem 5.1 below, and showing that $C_{\gamma\text{-Boost}} = C^*$. The fact that C^* is a lower bound on the optimal tail constant in the full-information case follows from Theorem 4.3.

Theorem 5.1. *Consider an M/G/1 with class I job size distribution with a fixed label-size pair distribution (L, S), and let*

$$C^* = \liminf_{\theta \to \gamma} \frac{\gamma - \theta}{\gamma} \mathbf{E}[\exp(\theta T_{\theta\text{-Cheat}})].$$

(a) The tail constant of γ-Boost is

$$C_{\gamma\text{-Boost}} = C^* = C_W \mathbf{E}[\exp(\gamma S) - 1] \exp\big(\lambda \mathbf{E}[b_\gamma(L)\,(\exp(\gamma S) - 1)]\big). \tag{5.1}$$

(b) In the full-information setting, namely when $L = S$, γ-Boost is strongly tail-optimal.

Remark 5.2. While the strong tail optimality result is only for the full-information setting, the definition of C^* and the fact that $C_{\gamma\text{-Boost}} = C^*$ extend to the partial-information setting. Of course, the details of what the labels are affects the value of C^*, with the minimum occurring for the full-information setting.

However, based on Remark 4.4, we conjecture that in the partial-information setting, C^* is (a lower bound on) the optimal tail constant achievable with *nonpreemptive* policies that have access to only labels and arrival times. Theorem 5.1 would then imply that γ-Boost achieves this optimal tail constant. As evidence for this conjecture, we show in Appendix C that γ-Boost outperforms all other versions of Boost, and we show in Appendix D that γ-Boost outperforms Nudge-M.

Before proving Theorem 5.1, we introduce some notation for working with the θ-optimal boost function, analogous to the notation used in Section 3:

- Recall that b_θ is the boost function given in (1.2).
- We write $B_\theta = b_\theta(L)$ to mean the boost of the tagged job using boost function b_θ.
  - Similarly $B'_\theta = b_\theta(L')$ is the boost of a generic future arrival.
- We write $V_\theta(u)$ to refer to the crossing work using boost function b_θ.

Recall that for the constant to be well-defined, we require the crossing work $V(\infty)$ to have finite moment generating function. By Lemma 3.5, this amounts to showing (2.4). We therefore first show that (2.4) holds for the γ-optimal boost function b_γ.

Lemma 5.3. *For all $\theta > 0$, for all labels $l \in \mathbb{L}$,*

$$\mathbf{E}[B'_\theta(\exp(\theta S') - 1)] < \frac{1}{\theta}.$$

In particular, taking $\theta = \gamma$ implies the γ-optimal boost function satisfies (2.4).

Proof. Plugging in $B'_\theta = b_\theta(L')$ (Definition 2.3) and rearranging, it suffices to show that with probability 1,

$$\big(\mathbf{E}[\exp(\theta S') \mid L'] - 1\big) \log \frac{\mathbf{E}[\exp(\theta S') \mid L']}{\mathbf{E}[\exp(\theta S') \mid L'] - 1} < 1. \tag{5.2}$$

Letting $x = \mathbf{E}[\exp(\theta S') \mid L'] - 1$, this holds because $x \log(1 + 1/x) \le 1$ for all $x > 0$.

Given Lemma 5.3, we can now employ Theorem 3.1 and Lemma 3.3. The last ingredient we need to prove Theorem 5.1 is to understand how the θ-optimal boost function compares to the γ-optimal boost function. We observe below that as a function of θ, the θ-optimal boost is actually monotonic, with larger θ yielding smaller boosts.

Lemma 5.4. *For all labels $l \in \mathbb{L}$, both $b_\theta(l)$ and $\theta b_\theta(l)$ are decreasing as functions of $\theta > 0$.*

Proof. It suffices to prove that $\theta b_\theta(l)$ is decreasing, which follows from writing it as

$$\theta b_\theta(l) = -\log\left(1 - \frac{1}{\mathbf{E}[\exp(\theta S) \mid L = l]}\right).$$

Proof of Theorem 5.1. We first note that the expression on the right-hand side of (5.1) follows from Theorem 3.1 and plugging in Definition 2.3 with $\theta = \gamma$. Specifically, the second factor from Theorem 3.1 simplifies to

$$\begin{aligned}
\mathbf{E}[\exp(\gamma(S - B_\gamma))] &= \mathbf{E}[\mathbf{E}[\exp(\gamma S) \mid L]\ \exp(-\gamma b_\gamma(L))] \\
&= \mathbf{E}\left[\mathbf{E}[\exp(\gamma S) \mid L] \cdot \frac{\mathbf{E}[\exp(\gamma S) \mid L] - 1}{\mathbf{E}[\exp(\gamma S) \mid L]}\right] \\
&= \mathbf{E}[\exp(\gamma S)] - 1.
\end{aligned}$$

Also, note that in the full-information setting, $C^* \le C_\pi$ for all policies π, so strong tail optimality is implied by $C_{\gamma\text{-Boost}} = C^*$.

It thus remains only to compute C^* to confirm that $C_{\gamma\text{-Boost}} = C^*$. To do so, we consider a system using the θ-Cheat policy, bound $\mathbf{E}[\exp(\theta T_{\theta\text{-Cheat}})]$, then compute the $\theta \to \gamma$ limit. We use the analysis from Section 3.1 with boost function b_θ (Definition 2.3).

From Lemma 3.3, we have $T_{\theta\text{-Cheat}} \ge W - B_\theta + V_\theta(W) + S$. We can lower bound this further using Lemma 5.4: if we were to change *other jobs' boosts* from B'_θ to B'_γ, it would only improve the tagged job's response time. This means $V_\theta(w) \ge V_\gamma(w)$ for all $w \ge 0$, so

$$T_{\theta\text{-Cheat}} \ge W - B_\theta + V_\gamma(W) + S.$$

Therefore, for all $w \ge 0$,

$$\begin{aligned}
\mathbf{E}[\exp(\theta T_{\theta\text{-Cheat}})] &\ge \mathbf{E}\big[\exp\big(\theta\big(W - B_\theta + V_\gamma(W) + S\big)\big)\big] \\
&\ge \mathbf{E}\big[\exp\big(\theta(W - B_\theta + V_\gamma(W) + S)\big)\, \mathbb{1}(W > w)\big] \\
&\ge \mathbf{E}\big[\exp\big(\theta(W - B_\theta + V_\gamma(w) + S)\big)\, \mathbb{1}(W > w)\big] \\
&= \mathbf{E}[\exp(\theta W)\, \mathbb{1}(W > w)]\, \mathbf{E}[\exp(\theta(S - B_\theta))]\, \mathbf{E}[\exp(\theta V_\gamma(w))].
\end{aligned}$$

Taking the $\theta \to \gamma$ limit and applying the monotone convergence theorem, which applies thanks to the monotonicity of θB_θ (Lemma 5.4), yields

$$\begin{aligned}
&\liminf_{\theta\to\gamma} (\gamma-\theta)\mathbf{E}[\exp(\theta T_{\theta\text{-Cheat}})] \\
&\quad\geq \liminf_{\theta\to\gamma} (\gamma-\theta)\mathbf{E}[\exp(\theta W)\mathbb{1}(W>w)]\,\mathbf{E}[\exp(\theta(S-B_\theta))]\,\mathbf{E}[\exp(\theta V_\gamma(w))] \\
&\quad= \left(\lim_{\theta\to\gamma} (\gamma-\theta)\mathbf{E}[\exp(\theta W)\mathbb{1}(W>w)]\right)\left(\lim_{\theta\to\gamma} \mathbf{E}[\exp(\theta(S-B_\theta))]\right)\left(\lim_{\theta\to\gamma} \mathbf{E}[\exp(\theta V_\gamma(w))]\right) \\
&\quad= \gamma C_W \mathbf{E}[\exp(\gamma(S-B_\gamma))]\mathbf{E}[\exp(\gamma V_\gamma(w))], \qquad (5.3)
\end{aligned}$$

where the second line follows from (2.1) and the fact that

$$\lim_{\theta\to\gamma} (\gamma-\theta)\mathbf{E}[\exp(\theta W)\,\mathbb{1}(W\leq w)] \leq \lim_{\theta\to\gamma} (\gamma-\theta)\exp(\gamma w) = 0.$$

Finally, as in the proof of Theorem 3.1, we observe that because (5.3) holds for all w, it holds in the $w \to \infty$ limit, yielding

$$C^* \geq C_W \mathbf{E}\big[\exp\big(\gamma\big(S-B_\gamma\big)\big)\big]\mathbf{E}\big[\exp\big(\gamma V_\gamma(\infty)\big)\big].$$

The right-hand side is exactly the tail constant $C_{\gamma\text{-Boost}}$ from Theorem 3.1, so $C^* \geq C_{\gamma\text{-Boost}}$.

It remains only to show $C^* \leq C_{\gamma\text{-Boost}}$. In the full-information setting, this is immediate from Theorem 4.3, which implies $C^* \leq C_\pi$ for all policies π. In the partial-information setting, Theorem 4.3 does not apply, but by following essentially the same steps as the upper bound in Theorem 3.1, we obtain

$$\begin{aligned}
\gamma C^* &\leq \left(\lim_{\theta\to\gamma} (\gamma-\theta)\mathbf{E}[\exp(\theta W)]\right)\left(\lim_{\theta\to\gamma} \mathbf{E}[\exp(\theta(S-\min\{B_\theta,u\}))]\right)\left(\lim_{\theta\to\gamma} \mathbf{E}[\exp(\theta V_\theta(\infty))]\right) \\
&= \gamma C_W \mathbf{E}[\exp(\gamma(S-\min\{B_\gamma,u\}))]\,\mathbf{E}[\exp(\gamma V_\gamma(\infty))].
\end{aligned}$$

This becomes $\gamma C_{\gamma\text{-Boost}}$ in the $u \to \infty$ limit, as desired. The main difference from Theorem 3.1 is that the second and third limits above involve a boost function that varies with θ. We compute the second limit with Lemma 5.4 and the monotone convergence theorem, and we compute the third limit with Lemma 3.5 and the bounded convergence theorem, which applies thanks to (5.2).

6 SIMULATIONS

We have shown that γ-Boost achieves strong tail optimality, which is an asymptotic property. However, there remain unanswered questions about γ-Boost that are important to practitioners. In this section, we explore the questions below via simulations. We focus by default on the full-information setting, but we address some partial-information settings in the last two questions.

- (Section 6.1) How well does γ-Boost perform in practical regimes? Does it do as well as one would predict from its tail constant $C_{\gamma\text{-Boost}}$?
- (Section 6.2) How does γ-Boost compare to Nudge and SRPT?
- (Section 6.3) In what settings does γ-Boost offer the largest benefit? What role does the variance of the job size distribution play?
- (Section 6.4) Is γ-Boost robust to misspecification, such as being given the wrong value of γ or noisily estimated job sizes?
- (Section 6.5) How well does γ-Boost perform in the partial-information setting, where we have much coarser information about jobs' sizes?

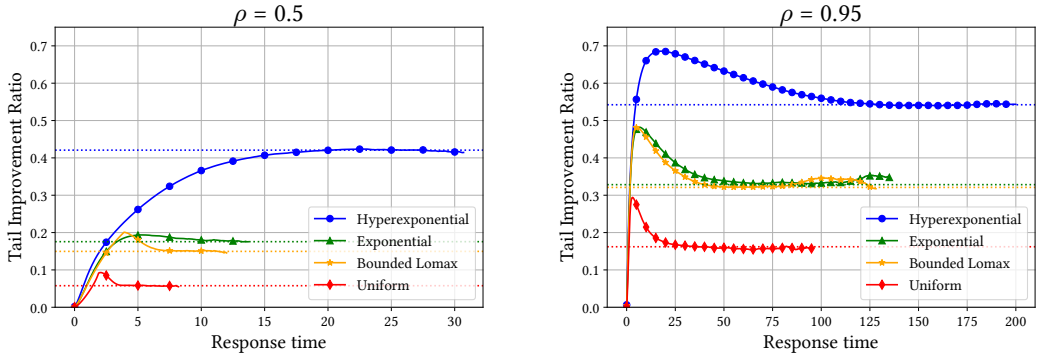


Fig. 6.1. Empirical TIR of γ-Boost over FCFS for several job size distributions S, each with mean $\mathbf{E}[S] = 1$, at loads $\rho = 0.5, 0.95$. See Fig. 1.2(a) for $\rho = 0.8$. We use the same distributions as Grosof et al. [21, Fig. 2], which are: Uniform(0,2), Exponential, Hyperexponential with branches drawn from Exp(2) and Exp(1/3) and first branch probability 0.8, and BoundedLomax with shape parameter $\alpha = 2$ and upper bound 4. The asymptotic TIR is computed with Theorem 3.1 and plotted as a same color dotted line for each distribution. Simulations run with 50 million arrivals.

6.1 Boost in the full-information setting

In Fig. 6.1, we evaluate the performance of the optimal boost policy on common distributions by looking at the empirical Tail Improvement Ratio (TIR) with respect to FCFS, that is, by looking at $\text{TIR}(t) = 1 - \mathbf{P}[T_{\gamma\text{-Boost}} > t]/\mathbf{P}[T_{\text{FCFS}} > t]$. Boost's performance improves upon FCFS's across a variety of job size distributions and loads. In all tested distributions, γ-Boost achieves asymptotic performance equivalent to the TIR that our theory suggests. Moreover, these improvements can be significant: γ-Boost improves the tail constant in the Exponential case by roughly 30%, and improvements exceed 50% for the Hyperexponential distribution case. Another observation of note is that across all loads and distributions, not only does γ-Boost achieve the asymptotic performance suggested by theory, it also improves *stochastically* over FCFS, meaning $\text{TIR}(t) > 0$ for all $t > 0$, not just the $t \to \infty$ limit. Moreover, in many cases the "pre-asymptotic" improvement actually exceeds the asymptotic TIR.

6.2 Boost compared to other policies

We evaluate the performance of boost policies against other policies, namely against Nudge, which is known to have better stochastic performance than FCFS, and SRPT, which is tail-pessimal for Class I distributions. In Figs. 1.2(b) and 6.2, we compare the TIR for all three policies. Following best practices from Grosof et al. [21, Fig. 2, Section 9], we set the small-large threshold for Nudge to be at $\mathbf{E}[S]$, with no medium or extra-large split, and examine performance under a variety of common job size distributions. We find that γ-Boost has larger asymptotic TIR than Nudge and SRPT across all tested job size distributions. Moreover, γ-Boost is stochastically better than Nudge across the distributions tested.

6.3 Variation matters: how CoV affects asymptotic performance

What is important when scheduling for the tail in distributions? If job size distributions are highly variable, tail-pessimal policies for class I distributions that are optimal for heavy-tailed distributions may still perform well for all but the highest tail percentiles. In particular, SRPT demonstrates strong performance for all but the highest tail percentiles. Therefore, while γ-Boost is asymptotically

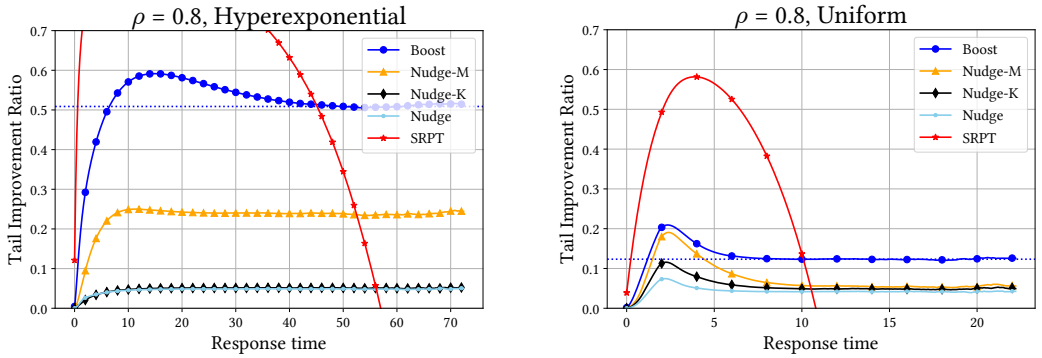


Fig. 6.2. Comparison of empirical TIR of γ-Boost against FCFS and Nudge, for two job size distributions, each with mean 1, and with the same settings as in Grosof et al. [21, Fig. 2]. The left is Hyperexponential with branches drawn from Exp(2) and Exp(1/3), with first branch probability 0.8 and the right is Uniform(0,2). See Fig. 1.2(b) for exponential. In each plot, the dotted horizontal line represents γ-Boost's asymptotic TIR for the respective distribution. For Hyperexponential, we set K to the optimal value of 8 for Nudge-K/M, with type-1 and type-2 jobs set to jobs coming from the Exp(2) and Exp(1/3) branches respectively. (Nudge-K does perform slightly better than Nudge in this case, though this is barely visible on the plot.) For Uniform, we use the same small-large split as Nudge, where type-1 jobs are small (smaller than the mean of the distribution) and type-2 jobs are large, and set K to the optimal value of 3 for Nudge-K/M. Simulations run with 50 million arrivals.

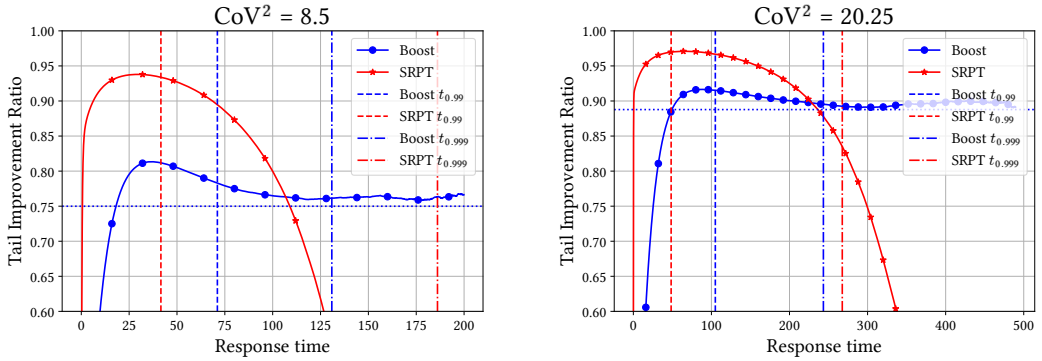
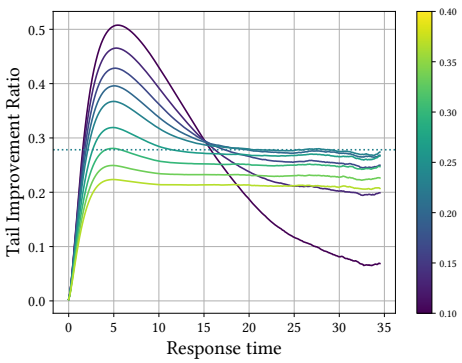
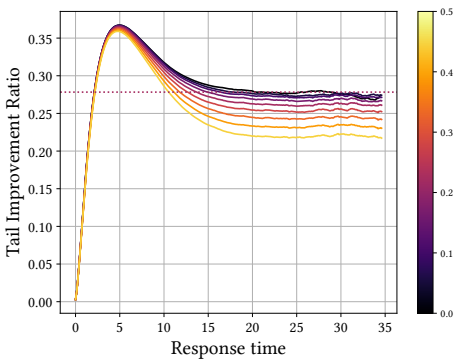


Fig. 6.3. A comparison of γ-Boost and SRPT at high CoV. In both plots, the distributions considered are Hyperexponential, mean 1 distributions. For $\mathrm{CoV}^2 = 8.5$, we choose parameters Exp(4), Exp(1/6), and a first-branch probability of $p = 20/23$. For $\mathrm{CoV}^2 = 20.25$ we choose parameters Exp(8), Exp(1/12), and a first-branch probability of $p = 88/95$. Load is $\rho = 0.8$. Dashed vertical lines mark the $t_{0.99}$ response times of the two policies, and dash-dotted vertical lines mark the $t_{0.999}$ response time. The dotted horizontal line represents the theoretical asymptotic TIR for γ-Boost. Observe how SRPT has lower $t_{0.99}$ response time but higher $t_{0.999}$ response time than γ-Boost. Simulations run with 50 million arrivals.

optimal, a practitioner's choice of policy depends on how sensitive they are to tail response times and the properties of the job size distribution they face. In particular, one may care about tail percentiles that are "pre-asymptotic". In Fig. 6.3, we show how this "pre-asymptotic" performance can depend on the variability of the job size distribution. The main takeaway is that for job size



(a) Sensitivity to misspecified γ.



(b) Sensitivity to noisy size estimates.

Fig. 6.4. Performance of γ-Boost with different types of noise. Job size distribution is Exponential with mean 1, with $\rho = 0.8$. (a) We plot the performance with misspecified γ. The theoretical asymptotic TIR is shown as a dotted horizontal line. We consider a range of $[\gamma/2, 2\gamma]$, where the optimal $\gamma = 0.2$. Observe that underestimating γ leads to significantly worse TIR compared to overestimating γ. (b) We plot the performance with noisy job size estimates. The noise is multiplicative and drawn i.i.d. for each job from a LogNormal distribution with mean 0 and some standard deviation. We consider noise levels (i.e. standard deviations) ranging from 0 to 0.5. While performance is degraded from the theoretically optimal TIR as more noise is added, at 0.5, the TIR is still above 20%. In both (a) and (b), simulations run with 50 million arrivals.

distributions with high variability, SRPT has great performance unless one cares about extremely high threshold (e.g. $t_{0.999}$) response times.

6.4 Robustness of Boost

γ-Boost sets a job's boost using its arrival time, size, and the decay rate parameter γ for the job size distribution. In practice, one might only have access to noisy estimates of the job size distribution and of the job sizes.

If one does not know the exact job size distribution, then the parameter γ may be misspecified. In this case, our takeaways suggest that one can conservatively set the decay rate parameter γ higher than estimated to maintain good performance. For clarity, we denote the optimal policy as γ-Boost, and assume that we set the parameter $\hat{\gamma}$ based on our noisy information. We can see in Fig. 6.4(a) that using a conservative overestimate of the parameter, namely setting $\hat{\gamma} > \gamma$, has good performance. While $\hat{\gamma} < \gamma$ can still have good performance near γ, setting too low of a parameter leads to a faster decay in performance than setting too high of a parameter.

If one only has noisy estimates of the job sizes, using these estimates directly provides still good performance. In Fig. 6.4(b) we examine the sensitivity of γ-Boost to noise in the labels. Using the noisy size as input to the boost policy maintains good performance, with TIR above 20%.

6.5 Boost in the partial-information setting

In this section, we consider the case when job sizes are unknown, and one may only have coarse information, such as the class, of incoming jobs or rough thresholds on job sizes. We find that, as with the full-information setting, γ-Boost demonstrates good practical performance, attaining the asymptotic TIR suggested by theory.

In Fig. 6.5 we evaluate the performance of the optimal boost policy in one such limited-information setting. We consider the case where jobs can be *type-1* or *type-2*, where each type has a different

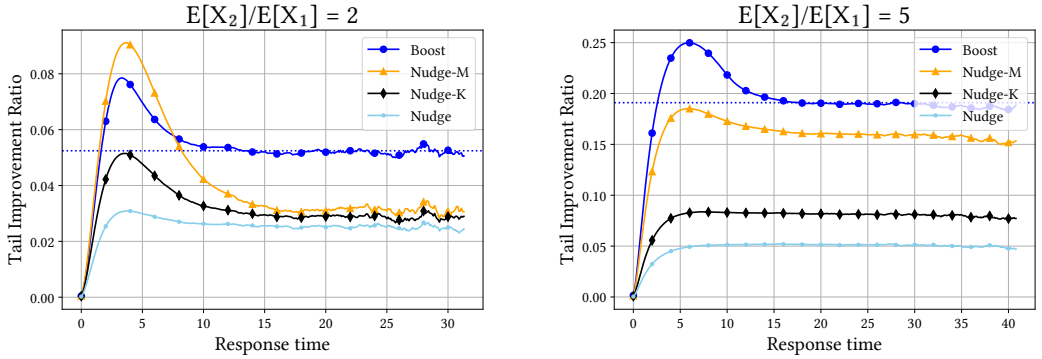


Fig. 6.5. Comparison of γ-Boost's performance to Nudge-K's and Nudge-M's performance in the two-class partial-information setting. The mean of the job size distribution is 1, and, following the settings in Van Houdt [45, Fig. 1], we set the load to be $\rho = 0.75$ and consider the case where both type-1 and type-2 jobs come from Exponential distributions and the probability of drawing from either branch is 1/2. On the left, we consider the case where the ratio of the means type-2/type-1 is 2; on the right it is 5. The more separable the means are, the better the performance of γ-Boost becomes. In both cases, the asymptotic TIR of the optimal boost, represented by the dotted line, is better than that of both Nudge-K and Nudge-M. In both cases we set the optimal value of K for the Nudge policies ($K = 3$ in the first setting and $K = 5$ in the second setting). Simulations run with 50 million arrivals.

distribution, and consider Nudge-K and Nudge-M for K in the setup from Van Houdt [45, Fig. 1]. We also extend this to the case where the distributions of the two job types are further separated. We find that performance increases the more distinguishable the distributions of the two job types are. We also find that under these settings, γ-Boost has better performance than both Nudge-K and Nudge-M.

7 CONCLUSION

In this work, we introduce the *Boost* family of scheduling policies, which provide a simple new approach to balancing the tradeoff between prioritizing short jobs and prioritizing jobs that have been waiting a long time. We prove that a policy in this new family, called *γ-Boost*, is a *strongly tail-optimal scheduling policy* for the M/G/1 with light-tailed job size distributions, resolving a long-standing open problem in queueing theory. Our simulations show that in addition to achieving a theoretical milestone, γ-Boost has excellent practical performance.

Our results on Boost reveal many promising future directions, some of which we outline below.

7.1 Settings beyond the full-information M/G/1

The most direct question prompted by our results is: what policy is strongly tail-optimal in the partial-information setting? We believe that γ-Boost is the desired policy if one further restricts attention to nonpreemptive policies. But if one allows preemption, it is less clear what to do. For minimizing mean response time in the M/G/1, it is known that the *Gittins policy* [16] is optimal for a wide variety of preemptive partial-information settings [35, 37]. Moreover, versions of the Gittins policy exist for the batch relaxation [33] with discounting. Perhaps a version of Boost based on the Gittins policy is needed for the preemptive partial-information setting.

Boost is also a natural candidate for systems beyond the M/G/1. One important direction would be multiserver systems. Recent results have shown that simple priority policies for optimizing

mean response time in the M/G/1 generalize well to a wide variety of multiserver systems [17–20, 24, 34, 35]. Can we leverage similar techniques to analyze Boost in the M/G/k or dispatching settings? Similarly, one could ask about non-Poisson arrival processes. We suspect that γ-Boost is also strongly-tail optimal in models like the G/G/1 with light-tailed job sizes, as long as we still have $\mathbf{P}[T_{\text{FCFS}} > t] \sim C_{\text{FCFS}} \exp(-\gamma t)$.

7.2 Metrics beyond the tail constant

One could reexamine the metric or metrics being optimized. We now know how to minimize mean response time, namely using SRPT, and the tail constant, namely using γ-Boost. Can we characterize the Pareto frontier of achievable means and tail constants, and can we design policies to achieve the entire frontier? Because the tail constant is a purely asymptotic notion, we conjecture that, at least theoretically, it is possible to design a policy with mean response time arbitrarily close to SRPT's and tail constant arbitrarily close to γ-Boost's. But whether such a policy would have good tail performance in practice remains to be seen.

Other important metrics arise in systems with multiple priority classes, where we want to promise better performance to better priority classes. Can we design policies to optimize a *weighted* tail constant, meaning a convex combination of each class's tail constant? It seems likely that reducing to a weighted batch problem would yield a version of γ-Boost that accounts for the weights.

But what if we also want to balance a tradeoff between the *decay rates* of the different classes' response times? This seems like a more challenging problem, but we expect it to be important for settings where the top priority classes must have very low response times. The results of Stolyar and Ramanan [44] show how to balance decay rates optimally using what is essentially an accumulating priority policy [11, 12, 43], albeit in a slightly weaker sense than our Definition 2.2. Combining Boost with accumulating priority thus seems like a promising direction for further exploration.

7.3 Boost in practice

We are excited about the potential of deploying Boost in real-world computer systems, due to its promising simulation performance and simplicity of implementation. The simplicity angle is especially important in high-performance environments such as network switches. Boost fits into the recently proposed Priority-In, First-Out (PIFO) scheduling abstraction [42]. One question is whether Boost is amenable to being approximated using methods like SP-PIFO [3], which would allow it to be deployed more easily on common network hardware.

Another question is how to determine γ in practice, given that the job size distribution may be unknown, or may change over time. We suspect that the best way to obtain γ in practice is to directly measure the empirical decay rate of the work distribution. This seems feasible to do in many computer systems, and it seems simpler than trying to estimate the full job size distribution. Our simulations show that Boost is robust to some misspecification of γ, so such an empirical estimate would likely suffice.

Of course, computer systems are just one of many domains where one might consider deploying Boost. Different domains have different prioritization needs, which could be met by different boost functions. It is worth highlighting that Boost is weakly tail-optimal for any boost function satisfying the relatively mild condition of (2.4). Even jobs that receive zero boost experience weakly optimal response time tail. This means that for systems with light-tailed job sizes, Boost offers a flexible framework for priority scheduling without sacrificing tail performance.

ACKNOWLEDGMENTS

We thank Isaac Grosof, Benny Van Houdt, and David Shmoys for helpful discussions. This work was supported by the National Science Foundation under grant nos. CMMI-2307008, DMS-2023528, and DMS-2022448.

REFERENCES

[1] Joseph Abate, Gagan L. Choudhury, and Ward Whitt. 1994. Waiting-Time Tail Probabilities in Queues with Long-Tail Service-Time Distributions. *Queueing Systems* 16, 3-4 (Sept. 1994), 311–338. https://doi.org/10.1007/BF01158960

[2] Joseph Abate and Ward Whitt. 1997. Asymptotics for $M/G/1$ Low-Priority Waiting-Time Tail Probabilities. *Queueing Systems* 25, 1 (June 1997), 173–233. https://doi.org/10.1023/A:1019104402024

[3] Albert Gran Alcoz, Alexander Dietmüller, and Laurent Vanbever. 2020. SP-PIFO: Approximating Push-in First-out Behaviors Using Strict-Priority Queues. In *17th USENIX Symposium on Networked Systems Design and Implementation (NSDI 2020)*. USENIX Association, Santa Clara, CA, 59–76. https://www.usenix.org/conference/nsdi20/presentation/alcoz

[4] François Baccelli and Pierre Brémaud. 2003. *Elements of Queueing Theory: Palm Martingale Calculus and Stochastic Recurrences* (2 ed.). Number 26 in Stochastic Modelling and Applied Probability. Springer, Berlin, Germany. https://doi.org/10.1007/978-3-662-11657-9

[5] 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, UK.

[6] Sem C. Borst, Onno J. Boxma, Rudesindo Núñez-Queija, and Bert Zwart. 2003. The Impact of the Service Discipline on Delay Asymptotics. *Performance Evaluation* 54, 2 (Oct. 2003), 175–206. https://doi.org/10.1016/S0166-5316(03)00071-3

[7] Onno J. Boxma and Bert Zwart. 2007. Tails in Scheduling. *ACM SIGMETRICS Performance Evaluation Review* 34, 4 (March 2007), 13–20. https://doi.org/10.1145/1243401.1243406

[8] Pierre Brémaud. 2020. *Probability Theory and Stochastic Processes.* Springer, Cham, Switzerland. https://doi.org/10.1007/978-3-030-40183-2

[9] Nils Charlet and Benny Van Houdt. 2024. Tail Optimality and Performance Analysis of the Nudge-M Scheduling Algorithm. arXiv:2403.06588 [cs, math] http://arxiv.org/abs/2403.06588

[10] Yan Chen and Jing Dong. 2021. Scheduling with Service-Time Information: The Power of Two Priority Classes. arXiv:2105.10499 [cs, math] http://arxiv.org/abs/2105.10499

[11] Val Andrei Fajardo and Steve Drekic. 2015. Controlling the Workload of $M/G/1$ Queues via the q-Policy. *European Journal of Operational Research* 243, 2 (June 2015), 607–617. https://doi.org/10.1016/j.ejor.2014.12.036

[12] Val Andrei Fajardo and Steve Drekic. 2017. Waiting Time Distributions in the Preemptive Accumulating Priority Queue. *Methodology and Computing in Applied Probability* 19, 1 (March 2017), 255–284. https://doi.org/10.1007/s11009-015-9476-1

[13] Eric J. Friedman and Shane G. Henderson. 2003. Fairness and Efficiency in Web Server Protocols. *ACM SIGMETRICS Performance Evaluation Review* 31, 1 (June 2003), 229–237. https://doi.org/10.1145/885651.781056

[14] Eric J. Friedman and Gavin Hurley. 2003. *Protective Scheduling.* Technical Report 1373. Cornell University, Ithaca, NY. 11 pages. https://hdl.handle.net/1813/9250

[15] Bezalel Gavish and Paul J. Schweitzer. 1977. The Markovian Queue with Bounded Waiting Time. *Management Science* 23, 12 (Aug. 1977), 1349–1357. https://doi.org/10.1287/mnsc.23.12.1349

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

[17] Isaac Grosof. 2023. *Optimal Scheduling in Multiserver Queues.* Ph. D. Dissertation. Carnegie Mellon University, Pittsburgh, PA. https://isaacg1.github.io/assets/isaac-thesis.pdf

[18] Isaac Grosof, Ziv Scully, and Mor Harchol-Balter. 2018. SRPT for Multiserver Systems. *Performance Evaluation* 127–128 (Nov. 2018), 154–175. https://doi.org/10.1016/j.peva.2018.10.001

[19] Isaac Grosof, Ziv Scully, and Mor Harchol-Balter. 2019. Load Balancing Guardrails: Keeping Your Heavy Traffic on the Road to Low Response Times. *Proceedings of the ACM on Measurement and Analysis of Computing Systems* 3, 2, Article 42 (June 2019), 31 pages. https://doi.org/10.1145/3341617.3326157

[20] Isaac Grosof, Ziv Scully, Mor Harchol-Balter, and Alan Scheller-Wolf. 2022. Optimal Scheduling in the Multiserver-Job Model under Heavy Traffic. *Proceedings of the ACM on Measurement and Analysis of Computing Systems* 6, 3, Article 51 (Dec. 2022), 32 pages. https://doi.org/10.1145/3570612

[21] Isaac Grosof, Kunhe Yang, Ziv Scully, and Mor Harchol-Balter. 2021. Nudge: Stochastically Improving upon FCFS. *Proceedings of the ACM on Measurement and Analysis of Computing Systems* 5, 2, Article 21 (June 2021), 29 pages. https://doi.org/10.1145/3460088

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

[23] 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. https://doi.org/10.1145/762483.762486

[24] Yige Hong and Ziv Scully. 2024. Performance of the Gittins Policy in the G/G/1 and G/G/k, with and without Setup Times. *Performance Evaluation* 163, Article 102377 (Jan. 2024), 26 pages. https://doi.org/10.1016/j.peva.2023.102377

[25] J. F. C. Kingman. 1993. *Poisson Processes.* Number 3 in Oxford Studies in Probability. Oxford University Press, Oxford.

[26] Aryeh Kontorovich and Iosif Pinelis. 2019. Exact Lower Bounds for the Agnostic Probably-Approximately-Correct (PAC) Machine Learning Model. *The Annals of Statistics* 47, 5 (Oct. 2019), 2822–2854. https://doi.org/10.1214/18-AOS1766

[27] Jan Karel Lenstra and David B. Shmoys. 2020. Elements of Scheduling. arXiv:2001.06005 [cs] http://arxiv.org/abs/2001.06005

[28] Andrea Marin, Sabina Rossi, and Carlo Zen. 2020. Size-Based Scheduling for TCP Flows: Implementation and Performance Evaluation. *Computer Networks* 183, Article 107574 (Dec. 2020), 15 pages. https://doi.org/10.1016/j.comnet.2020.107574

[29] Jayakrishnan Nair, Adam Wierman, and Bert Zwart. 2010. Tail-Robust Scheduling via Limited Processor Sharing. *Performance Evaluation* 67, 11 (Nov. 2010), 978–995. https://doi.org/10.1016/j.peva.2010.08.012

[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. https://doi.org/10.1023/A:1020905810996

[31] Misja Nuyens, Adam Wierman, and Bert Zwart. 2008. Preventing Large Sojourn Times Using SMART Scheduling. *Operations Research* 56, 1 (Feb. 2008), 88–101. https://doi.org/10.1287/opre.1070.0504

[32] Misja Nuyens and Bert Zwart. 2006. A Large-Deviations Analysis of the $GI/GI/1$ SRPT Queue. *Queueing Systems* 54, 2 (Oct. 2006), 85–97. https://doi.org/10.1007/s11134-006-8767-1

[33] Michael Pinedo. 2016. *Scheduling: Theory, Algorithms, and Systems* (5 ed.). Springer, Cham, Switzerland.

[34] Ziv Scully. 2022. *A New Toolbox for Scheduling Theory.* Ph. D. Dissertation. Carnegie Mellon University, Pittsburgh, PA. https://ziv.codes/pdf/scully-thesis.pdf

[35] 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 (Nov. 2020), 29 pages. https://doi.org/10.1145/3428328

[36] 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. https://doi.org/10.1109/ALLERTON.2018.8635963

[37] 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).* IFIP, Philadelphia, PA, 248–255. https://doi.org/10.23919/WiOpt52861.2021.9589051

[38] 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. https://doi.org/10.1145/3179419

[39] Ziv Scully and Lucas van Kreveld. 2024. When Does the Gittins Policy Have Asymptotically Optimal Response Time Tail in the M/G/1? *Operations Research* 72, 2 (Feb. 2024). https://doi.org/10.1287/opre.2022.0038

[40] 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. https://doi.org/10.1145/3392148

[41] J. George Shanthikumar and Ushio Sumita. 1987. Convex Ordering of Sojourn Times in Single-Server Queues: Extremal Properties of FIFO and LIFO Service Disciplines. *Journal of Applied Probability* 24, 3 (Sept. 1987), 737–748. https://doi.org/10.2307/3214103

[42] Anirudh Sivaraman, Suvinay Subramanian, Mohammad Alizadeh, Sharad Chole, Shang-Tse Chuang, Anurag Agrawal, Hari Balakrishnan, Tom Edsall, Sachin Katti, and Nick McKeown. 2016. Programmable Packet Scheduling at Line Rate. In *Proceedings of the 2016 ACM SIGCOMM Conference (SIGCOMM 2016).* ACM, Florianopolis, Brazil, 44–57. https://doi.org/10.1145/2934872.2934899

[43] David A. Stanford, Peter Taylor, and Ilze Ziedins. 2014. Waiting Time Distributions in the Accumulating Priority Queue. *Queueing Systems* 77, 3 (July 2014), 297–330. https://doi.org/10.1007/s11134-013-9382-6

[44] Alexander L. Stolyar and Kavita Ramanan. 2001. Largest Weighted Delay First Scheduling: Large Deviations and Optimality. *The Annals of Applied Probability* 11, 1 (Feb. 2001), 1–48. https://doi.org/10.1214/aoap/998926986

[45] Benny Van Houdt. 2022. On the Stochastic and Asymptotic Improvement of First-Come First-Served and Nudge Scheduling. *Proceedings of the ACM on Measurement and Analysis of Computing Systems* 6, 3 (Dec. 2022), 1–22. https://doi.org/10.1145/3570610

[46] Adam Wierman and Bert Zwart. 2012. Is Tail-Optimal Scheduling Possible? *Operations Research* 60, 5 (Oct. 2012), 1249–1257. https://doi.org/10.1287/opre.1120.1086
[47] Ronald W. Wolff. 1982. Poisson Arrivals See Time Averages. *Operations Research* 30, 2 (1982), 223–231. http://www.jstor.org/stable/170165
[48] Bert Zwart and Onno J. Boxma. 2000. Sojourn Time Asymptotics in the M/G/1 Processor Sharing Queue. *Queueing Systems* 35, 1/4 (2000), 141–166. https://doi.org/10.1023/A:1019142010994

A STRONG TAIL OPTIMALITY FOR HEAVY-TAILED JOB SIZE DISTRIBUTIONS

In this appendix, we show that many scheduling policies that are known to be weakly tail-optimal in an M/G/1 with heavy-tailed job size distribution are, in fact, strongly tail-optimal. This confirms conjectures by Boxma and Zwart [7] and Wierman and Zwart [46].

The specific class of heavy-tailed distributions we consider are *regularly varying* distributions [5]. These are the distributions S such that there exists a constant $-\alpha$ such that for all $k > 0$,

$$\lim_{t\to\infty} \frac{\mathbf{P}[S > kt]}{\mathbf{P}[S > t]} = k^{-\alpha}. \tag{A.1}$$

The constant $-\alpha$ is called the *index* of regular variation of S. For the rest of this appendix, we consider an M/G/1 with regularly varying job size distribution S with index $-\alpha$.

There are a number of results showing that under various scheduling policies π,

$$\lim_{t\to\infty} \frac{\mathbf{P}[T_\pi > t]}{\mathbf{P}[S > (1-\rho)t]} = 1, \tag{A.2}$$

where $\rho = \lambda\mathbf{E}[S]$ is the system load. Policies π which satisfy (A.2) include classic policies like Processor Sharing, Shortest Remaining Processing Time, and Least Attained Service [7, 30, 31, 48]; as well as more complex policies like Shortest Expected Remaining Processing Time, Randomized Multi-Level Feedback, and the Gittins policy [39, 40].

Motivated by (A.1) and (A.2), let

$$C_\pi = \lim_{t\to\infty} \frac{\mathbf{P}[T_\pi > t]}{\mathbf{P}[S > (1-\rho)t]}.$$

Because a job's response time is at least its size, by (A.1), we have $C_\pi \geq (1-\rho)^\alpha$ for all policies. Any policy π satisfying (A.2) has $C_\pi = 1$ and is thus weakly tail-optimal. The question is whether any policy can achieve $C_\pi < 1$. Below, we use a result of Wierman and Zwart [46] to answer negatively.

Theorem A.1. *Consider an M/G/1 whose job size distribution has regularly varying tail. Any scheduling policy whose response time distribution satisfies* (A.2) *is strongly tail-optimal, and thus* $\inf_\pi C_\pi = 1$.

Proof. Fix a scheduling policy π. The key is a result of Wierman and Zwart [46] which gives a necessary condition to have $C_\pi < \infty$. We show that the condition also implies $C_\pi \geq 1$, as desired.

We first state a version of the necessary condition. Consider a tagged job arriving at time 0, and let $R_\pi(t)$ be the total time during $[0, t]$ for which π serves jobs that arrive after the tagged job. If $C_\pi < \infty$, then for all $\delta > 0$ [46, Proposition 1],

$$\lim_{t\to\infty} \mathbf{P}\big[R_\pi(t) > (\rho-\delta)t \;\big|\; S > (1-\rho+\delta)t\big] = 1. \tag{A.3}$$

Notice that $R_\pi(t) > (\rho-\delta)$ and $S > (1-\rho+\delta)t$ together imply $R_\pi(t) + S > t$. When this occurs, the tagged job does not receive enough service to complete by time t, so its response time satisfies

$T_\pi > t$. Therefore, for any $\delta > 0$,

$$\begin{aligned}
1 &= \lim_{t\to\infty} \mathbf{P}\left[R_\pi(t) > (\rho-\delta)t \;\middle|\; S > (1-\rho+\delta)t\right] \\
&\le \lim_{t\to\infty} \mathbf{P}\left[T_\pi > t \;\middle|\; S > (1-\rho+\delta)t\right] \\
&\le \lim_{t\to\infty} \frac{\mathbf{P}[T_\pi > t]}{\mathbf{P}[S > (1-\rho+\delta)t]} \\
&= \left(\frac{1-\rho+\delta}{1-\rho}\right)^{\alpha} \lim_{t\to\infty} \frac{\mathbf{P}[T_\pi > t]}{\mathbf{P}[S > (1-\rho)t]},
\end{aligned}$$

where the last line follows from (A.1). Taking the $\delta \to 0$ limit then implies all policies have $C_\pi \ge 1$. The fact that $\inf_\pi C_\pi = 1$ follows from the fact that multiple policies π achieve $C_\pi = 1$ [7, 30, 31, 39, 40, 48]

We further conjecture that the regularly varying requirement on S can be relaxed to requiring S be *intermediate regularly varying*, namely

$$\limsup_{\delta\to 0} \limsup_{t\to\infty} \frac{\mathbf{P}[S > t]}{\mathbf{P}[S > (1-\delta)t]} = 1.$$

This property suffices for the computation in our proof above, and it suffices for many of the prior works showing $C_\pi = 1$ for various policies π [30, 39, 40]. The only step of the proof that requires (non-intermediate) regular variation is (A.3), the result of Wierman and Zwart [46, Proposition 1]. If their result could be generalized to give the same necessary condition for $C_\pi < \infty$ even when S is intermediate regularly varying, it would imply $\inf_\pi C_\pi = 1$ in that setting, too.

B REDUCTION TO THE BATCH SCHEDULING PROBLEM FOR UNKNOWN SIZES

In this section, we extend the batch reduction to the case of unknown sizes with only size-label information for jobs. Specifically, we expand the definitions from Section 4 and show how the unknown-size case differs from the known size case.

Definition B.1. A *batch instance* $\mathcal{I} = \big((a_1, l_1), \ldots, (a_n, l_n)\big)$ is a finite batch of arrival times and labels.

Unlike the batch instance in Section 4, which consists of fully deterministic arrival time, job size pairs, we additionally need to define a *job size model*, which describes the distribution of job sizes given instance information.

Definition B.2. A *job size model* $\mathcal{S}$ is a function that maps batch instances $\mathcal{I} = \big((a_1, l_1), \ldots, (a_n, l_n)\big)$ to a joint distribution $(S_1, \ldots, S_n)$ on job sizes.

We consider two job size models. The first model is $\mathcal{S}_{\text{ind}}$, which is the model where the joint distribution of job sizes for an instance $\mathcal{I}$ is simply the product of the conditional distributions $(S \mid l_i)$. That is,

$$\big(\mathcal{S}_{\text{ind}}(\mathcal{I})\big)_i \sim (S \mid l_i), \qquad \big(\mathcal{S}_{\text{ind}}(\mathcal{I})\big)_i \text{ independent of } \big(\mathcal{S}_{\text{ind}}(\mathcal{I})\big)_j \text{ if } i < j.$$

The second job size model is the busy period model $\mathcal{S}_{\text{busy}}$. $\mathcal{S}_{\text{busy}}$ maps from batches to joint distributions of job sizes in the following way. Let BP denote the busy period distribution, where the triples $(A_i, L_i, S_i) \sim BP$. Then $\mathcal{S}_{\text{busy}}$ is a mapping such that if $\mathcal{B}$ is a random variable (A_i, L_i), then $(\mathcal{B}, \mathcal{S}_{\text{busy}}(\mathcal{B})) \sim BP$. That is, the joint distribution of sizes conditional on the batch instance coming from a busy period is also distributed such that it makes the triple distributed with the busy period distribution.

We can now define the cost function given a batch and job size model.

Definition B.3. The θ-cost of a policy π for an instance $\mathcal{I}$ under a given job size model $\mathcal{S}$ is given by

$$K_\pi(\theta, \mathcal{I}, \mathcal{S}) = \sum_{i=1}^{n} \mathbf{E}[\exp(\theta(D_{\pi,i} - a_i))],$$

where the expectation is over the joint job size distribution $\mathcal{S}(\mathcal{I})$ and any randomness in the policy.

Under the job size model $\mathcal{S}_{\text{ind}}$, we have the following optimality result:

Theorem B.4. *For any batch instance $\mathcal{I} = \big((a_i, l_i)\big)_{i=1}^n$, the θ-Cheat policy minimizes $K_\pi(\theta, \mathcal{I}, \mathcal{S}_{\text{ind}})$ in the nonpreemptive batch scheduling problem. Specifically, for any nonpreemptive policy π under this job size model,*

$$K_{\theta\text{-Cheat}}(\theta, \mathcal{I}, \mathcal{S}_{\text{ind}}) \le \min_\pi K_\pi(\theta, \mathcal{I}, \mathcal{S}_{\text{ind}}).$$

Proof. We observe that θ-Cheat serves jobs in $\mathcal{I}$ in the same order as the Weighted Discounted Shortest Expected Processing Time (WDSEPT) rule, with the weights equal to $\exp(-\theta a_i)$ and negative discount rate $-\theta$ for each job i. This is because boosted arrival time $a_i - b_\theta(l_i)$ is a monotonic function, namely the negative log, of WDSEPT's priority index:

$$a_i - b_\theta(l_i) = -\frac{1}{\theta} \log\left(\exp(-\theta a_i) \frac{\mathbf{E}[\exp(\theta S) \mid L = l_i]}{\mathbf{E}[\exp(\theta S) \mid L = l_i] - 1}\right).$$

The proof is an interchange argument identical to that of Pinedo [33, Theorem 10.1.3], with the signs for the discount rate and objective flipped. Specifically, with negative discounting, we define discounted completion time as $\exp(\theta D_i) - 1$ instead of $1 - \exp(\theta D_i)$ to keep the sign positive.

In Theorem 4.3, we concluded that θ-Cheat's optimality for all batch instances could be translated to a bound in the M/G/1. Why does this not work in Theorem B.4? The issue is that our optimality argument in the M/G/1 relies on sampling instances from busy periods. In particular, letting $\mathcal{B}$ be the random batch instance resulting from a busy period, by reasoning to that in the proof Theorem 4.3, we have

$$\mathbf{E}[\exp(\theta T_\pi)] = \frac{\mathbf{E}[K_\pi(\theta, \mathcal{B}, \mathcal{S}_{\text{busy}})]}{|\mathcal{B}|},$$

where the expectations on the right-hand side are over the distribution of $\mathcal{B}$. The key difference is that the job size model is $\mathcal{S}_{\text{busy}}$, not $\mathcal{S}_{\text{ind}}$, so Theorem B.4 does not apply.

Why is there not a similar issue in the full-information case? Because when a job's size is its label, the instance $\mathcal{I}$ already contains all the job size information. In particular, $\mathcal{S}_{\text{ind}}(\mathcal{I}) = \mathcal{S}_{\text{busy}}(\mathcal{I})$ for all instances $\mathcal{I}$, with both models simply yielding a deterministic vector of sizes extracted from the labels in the instance $\mathcal{I}$.

C OPTIMIZING THE BOOST FUNCTION IN THE PARTIAL-INFORMATION SETTING

In this appendix, we show that in the partial-information setting, γ-Boost has a lower tail constant than Boost with any other boost function:

$$C_{\gamma\text{-Boost}} \le C_{\text{Boost}}.$$

To keep the argument simple, we assume a finite set of labels $\mathbb{L} = \{1, \dots, n\}$. The result can be generalized to arbitrary sets of labels using a calculus of variations argument.

For brevity, we adopt the notation

$$p_i = \mathbf{P}[L = i], \qquad b_i = b(i), \qquad s_i = \mathbf{E}[\exp(\gamma S) \mid L = i].$$

Choosing a boost function amounts to choosing a vector of boosts $(b_1, \ldots, b_n)$. To make the dependence of C_{Boost} on the b_i explicit, we let

$$C(b_1, \ldots, b_n) = C_{\text{Boost}}.$$

By Definition 2.3, we can write $C_{\gamma\text{-Boost}} = C(b_1^*, \ldots, b_n^*)$, where

$$b_i^* = \frac{1}{\gamma} \log \frac{s_i}{s_i - 1}.$$

Optimality of γ-Boost among Boost policies amounts to showing that $(b_1^*, \ldots, b_n^*)$ is a minimizer of $C(\cdot)$. To show this, it suffices to show the following two claims:

- $C(\cdot)$ is convex.
- $\nabla C(b_1^*, \ldots, b_n^*) = 0$.

We start with convexity of $C(\cdot)$. Writing $p_i = \mathbf{P}[L = i]$, we can rewrite the tail constant from Theorem 3.1 as

$$\begin{aligned} C(b_1, \ldots, b_n) &= C_W \left(\sum_{i=1}^n p_i s_i \exp(-\gamma b_i) \right) \exp\left(\sum_{i=1}^n \lambda p_i b_i (s_i - 1) \right) \\ &= C_W \sum_{i=1}^n p_i s_i \exp\left(-\gamma b_i + \sum_{j=1}^n \lambda p_j b_j (s_j - 1) \right). \end{aligned}$$

We observe that $C(\cdot)$ a sum of compositions of linear functions with exp, so it is convex. In fact, $C(\cdot)$ is log-convex, because it is a product of a mixture of log-convex functions, which is log-convex [26, proof of Lemma A.3], and another log-convex function.

To show the gradient at $(b_1^*, \ldots, b_n^*)$ is zero, we take the derivative of $C(b_1, \ldots, b_n)$ with respect to b_k, obtaining

$$\frac{\partial}{\partial b_k} C(b_1, \ldots, b_n) = C_W \sum_{i=1}^n p_i s_i \big(\lambda p_k (s_k - 1) - \gamma \mathbb{1}(i = k) \big) \exp\left(-\gamma b_i + \sum_{j=1}^n \lambda p_j b_j (s_j - 1) \right).$$

This derivative is zero if and only if

$$\lambda p_k (s_k - 1) \sum_{i=1}^n p_i s_i \exp(-\gamma b_i) = \gamma p_k s_k \exp(-\gamma b_k).$$

Plugging in $b_i = b_i^*$ and noticing $s_i \exp(-\gamma b_i^*) = s_i - 1$, it suffices to show

$$\lambda \sum_{i=1}^n p_i (s_i - 1) = \gamma.$$

But the right-hand is $\lambda(\mathbf{E}[\exp(\gamma S)] - 1)$, so this is simply the definition of γ from (2.2).

D COMPARISON OF NUDGE-M AND BOOST TAIL CONSTANTS

In this appendix, we show that the γ-Boost policy for unknown job sizes with two labels is state-of-the-art among policies for this setting. Specifically, we compare to the known state-of-the-art, Nudge-M [9], and show that for any choice for the parameter K, there is a Boost policy which achieves a better tail constant. In particular, this means that the optimal Boost policy in this setting, namely γ-Boost (see Appendix C), has better tail constant than the optimal Nudge-M policy.

Recall that Nudge-M partitions the jobs into type-1 and type-2 jobs, allowing any type-1 job to pass any type-2 job that has arrived as one of the last M jobs. This can be thought of as the setting where each job is given label 1 or label 2, and a scheduling decision is made based on this label.

From here onward we will refer to type-1 jobs as jobs with label 1 and type-2 jobs as jobs with label 2.

For brevity, we adopt the notation

$$\begin{aligned} p_1 &= \mathbf{P}[L = 1], & p_2 &= \mathbf{P}[L = 2], \\ s_1 &= \mathbf{E}[e^{\gamma S} \mid L = 1], & s_2 &= \mathbf{E}[e^{\gamma S} \mid L = 2], & s &= \mathbf{E}[e^{\gamma S}], \\ b_1 &= b(1), & b_2 &= b(2). \end{aligned}$$

Note that with just two labels, $p_2 = 1 - p_1$. We assume both p_1 and p_2 are nonzero.

The tail constant of Nudge-M for any choice of parameter K is given by rewriting the expression in Charlet and Van Houdt [9, Theorem 1] as

$$\frac{C_{\text{Nudge-M}}}{C_{\text{FCFS}}} = \frac{p_1 s_1}{s} \frac{(p_1 s_1 + p_2)^K}{s^K} + \frac{p_2 s_2}{s} (p_1 s_1 + p_2)^K. \tag{D.1}$$

Recall that the tail constant of Boost is given by

$$\frac{C_{\text{Boost}}}{C_{\text{FCFS}}} = \frac{1}{s} \mathbf{E}[e^{\gamma(S-B)}] \mathbf{E}[e^{\gamma V}].$$

Under the two-label case above, the term $\mathbf{E}[e^{\gamma(S-B)}]$ can be written as

$$\begin{aligned} \mathbf{E}[e^{\gamma(S-B)}] &= p_1 \mathbf{E}[e^{\gamma S} \mid L = 1] e^{-\gamma b_1} + p_2 \mathbf{E}[e^{\gamma S} \mid L = 2] e^{-\gamma b_2} \\ &= p_1 s_1 e^{-\gamma b_1} + p_2 s_2 e^{-\gamma b_2}, \end{aligned}$$

where b_1 and b_2 are, respectively, the boosts given to jobs of label 1 and label 2. The crossing work can be similarly written, using Lemma 3.5, as the sum of an arrival stream of label-1 jobs and of label-2 jobs, i.e., as

$$\begin{aligned} \mathbf{E}[e^{\gamma V}] &= \exp\big(\lambda p_1 b_1 \mathbf{E}[e^{\gamma S} - 1 \mid L = 1] + \lambda p_2 b_2 \mathbf{E}[e^{\gamma S} - 1 \mid L = 2]\big) \\ &= \exp\big(\lambda p_1 (b_1 - b_2)(s_1 - 1) + \lambda b_2 \mathbf{E}[e^{\gamma S} - 1]\big) \\ &= \exp\big(\lambda p_1 (b_1 - b_2)(s_1 - 1) + \gamma b_2\big). \end{aligned}$$

The tail constant for Boost can thus be written as

$$\begin{aligned} \frac{C_{\text{Boost}}}{C_{\text{FCFS}}} &= \frac{1}{s} \big(p_1 s_1 e^{-\gamma b_1} + p_2 s_2 e^{-\gamma b_2}\big) \exp\big(\lambda p_1 (b_1 - b_2)(s_1 - 1) + \gamma b_2\big) \\ &= \left(\frac{p_1 s_1}{s} e^{-\gamma(b_1 - b_2)} + \frac{p_2 s_2}{s}\right) \exp\big(\lambda p_1 (b_1 - b_2)(s_1 - 1)\big). \end{aligned}$$

Consider an arbitrary K for Nudge-M. For such an K, consider $b_1 - b_2$ such that

$$b_1 - b_2 = \frac{K \log(s)}{\gamma}. \tag{D.2}$$

There are many boost functions that satisfy this, e.g. $b_1 = \frac{K \log(s)}{\gamma}$ and $b_2 = 0$.

For a boost function satisfying (D.2), we observe that $s^K = e^{\gamma(b_1 - b_2)}$. Using this observation, we can rewrite the expression in (D.1) as

$$\begin{aligned} \frac{C_{\text{Nudge-M}}}{C_{\text{FCFS}}} &= \frac{p_1 s_1}{s} \frac{(p_1 s_1 + p_2)^K}{e^{\gamma(b_1 - b_2)}} + \frac{p_2 s_2}{s} (p_1 s_1 + p_2)^K \\ &= \left(\frac{p_1 s_1}{s} e^{-\gamma(b_1 - b_2)} + \frac{p_2 s_2}{s}\right) (p_1 s_1 + p_2)^K \\ &= \left(\frac{p_1 s_1}{s} e^{-\gamma(b_1 - b_2)} + \frac{p_2 s_2}{s}\right) (1 + p_1 (s_1 - 1))^K. \end{aligned}$$

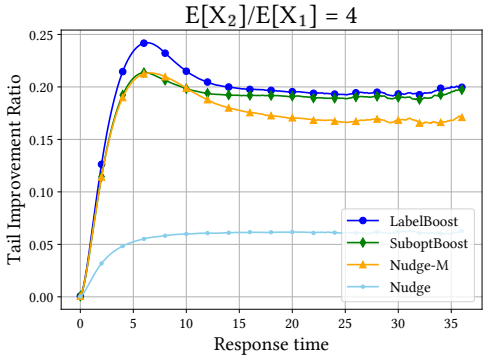


Fig. D.1. Empirical TIR of γ-Boost's performance compared to the Nudge-M policy in the two-label setting. Parameters are set to match those of Charlet and Van Houdt [9, Fig. 5]: label-1 and label-2 jobs are both exponential, with $\mathrm{E}[X_2]/\mathrm{E}[X_1] = 4$, the probability of a label-1 job $p_1 = 2/3$, arrival rate $\lambda = 0.7$. For Nudge-M, we use the optimal parameter $K = 5$. Observe that both γ-Boost, denoted as LabelBoost, and Boost with $b_1 - b_2 = \frac{K \log(s)}{\gamma}$, denoted as SuboptBoost, outperform Nudge-M.

Therefore, to compare the tail constant of Boost and that of Nudge-M, it suffices to compare the expressions $\exp\big(\lambda p_1(b_1 - b_2)(s_1 - 1)\big)$ and $(1 + p_1(s_1 - 1))^K$. We rewrite the first expression as

$$\begin{aligned}\exp\big(\lambda p_1(b_1 - b_2)(s_1 - 1)\big) &= \exp\left(\frac{\gamma(b_1 - b_2)p_1(s_1 - 1)}{s - 1}\right) \\ &= \exp(\gamma(b_1 - b_2))^{\frac{p_1(s_1-1)}{s-1}},\end{aligned}$$

and rewrite the second expression as

$$\begin{aligned}(1 + p_1(s_1 - 1))^K &= (1 + p_1(s_1 - 1))^{\frac{\log(e^{\gamma(b_1-b_2)})}{\log(s)}} \\ &= \exp\left(\frac{\log(1 + p_1(s_1 - 1))\log(e^{\gamma(b_1-b_2)})}{\log s}\right) \\ &= \exp(\gamma(b_1 - b_2))^{\frac{\log(1+p_1(s_1-1))}{\log(1+(s-1))}}.\end{aligned}$$

Finally, we observe that

$$\exp(\gamma(b_1 - b_2))^{\frac{p_1(s_1-1)}{s-1}} < \exp(\gamma(b_1 - b_2))^{\frac{\log(1+p_1(s_1-1))}{\log(1+(s-1))}},$$

since $s - 1 \geq p_1(s - 1)$ and $\frac{\log(1+x)}{x}$ is monotonically decreasing in x. This shows that for any K, Boost with $b_1 - b_2 = \frac{K \log(s)}{\gamma}$ performs better than Nudge-M, implying that the optimal Boost policy, namely γ-Boost (Appendix C), performs better than the optimal Nudge-M policy.

Received January 2023; revised April 2024; accepted April 2024