A Better Model for Job Redundancy: Decoupling Server Slowdown and Job Size

Kristen Gardner, Mor Harchol-Balter, Alan Scheller-Wolf, and Benny Van Houdt

Abstract—Recent computer systems research has proposed using redundant requests to reduce latency. The idea is to replicate a request so that it joins the queue at multiple servers. The request is considered complete as soon as any one of its copies completes. Redundancy allows us to overcome server-side variability – the fact that a server might be temporarily slow due to factors such as background load, network interrupts, and garbage collection – to reduce response time. In the past few years, queueing theorists have begun to study redundancy, first via approximations, and, more recently, via exact analysis. Unfortunately, for analytical tractability, most existing theoretical analysis has assumed an Independent Runtimes (IR) model, wherein the replicas of a job each experience independent runtimes (service times) at different servers. The IR model is unrealistic and has led to theoretical results which can be at odds with computer systems implementation results. This paper introduces a much more realistic model of redundancy. Our model decouples the inherent job size (X) from the server-side slowdown (S), where we track both S and X for each job. Analysis within the $S\&X$ model is, of course, much more difficult. Nevertheless, we design a dispatching policy, Redundant-to-Idle-Queue (RIQ), which is both analytically tractable within the $S\&X$ model and has provably excellent performance.

1 Introduction

As cloud computing and resource sharing become more prevalent, we are faced with greater degrees of server variability. Recent computer systems studies have shown that the same job can take up to 12× or 27× longer to run on one machine than another [3], [26]. This is due to varying background load, temporary garbage collection, networking interrupts, and other transient events. This server variability is exacerbated by multiplexing of applications and by our increased reliance on virtual machines (VMs); multiple VMs may share the same host resources, affecting each other in unpredictable ways.

In an effort to reduce overall latency, and particularly tail latency, the computer systems community has proposed using redundancy [3], [4], [7], [18], [19], [23]. Redundancy, also known as job replication, is the idea of dispatching the same job to multiple servers, where the job is considered "done" as soon as it *completes service on any one server*. Redundancy provides two key advantages: First, a job that is dispatched to d servers experiences the queue with the least work. This is true even if jobs are dispatched to queues immediately via a front-end load balancer that has no knowledge of the number of jobs in each queue, or the jobs' sizes. Second, each job experiences the minimum slowdown of the servers on which it runs. Both advantages are important when server-side variability is high.

As redundancy has become more popular in computer systems, a raft of theory papers have attempted to analyze the response time benefits of redundancy [6], [10]–[16], [20], [21], [23]. These results assume an *Independent Runtimes (IR) model*, where a job's copies have independent runtimes (service times) at different servers. The independent runtimes typically are assumed to be exponentially distributed or more highly variable. Here, the IR model suggests that (barring cancellation costs), "more redundancy is better."

While the IR model makes sense in certain settings, it can be problematic in others. Consider for example a job that is very large, meaning that it comprises a large volume of computation. That job should appear to be large on all servers, possibly slowed down more on some servers than others. This does not happen in the IR model: the job is assigned a different, independent, runtime on different servers. When the job runs on multiple servers it experiences the minimum runtime of all those servers' independent runtimes. Thus an inherently large job can become arbitrarily small under the independence assumption. There is no concept of an inherently large job which remains large at every server.

In this paper we propose a more realistic model for redundancy, called the $S\&X$ model (see Figure 1). The $S\&X$ model explicitly decouples the server slowdown (represented by the random variable S) from the inherent job size (represented by the random variable X). The $S\&X$ model marks a departure from traditional queueing theory, which uses a single "service time" variable to jointly represent the server speed and job size. A single random variable is insufficient in the context of redundancy: We need to decouple the variables so that a job with a large X component (large job size) will have a large X component on every server.

The $S\&X$ model sheds light, however, on some sad truths: redundancy is *not* always a win and can in fact be dangerous. Consider for example, the Redundancy-d policy,

- Kristen Gardner is with the Computer Science Department, Amherst College. E-mail: kgardner@amherst.edu
- Mor Harchol-Balter is with the Computer Science Department, Carnegie Mellon University. E-mail: harchol@cs.cmu.edu
- Alan Scheller-Wolf is with the Tepper School of Business, Carnegie Mellon University. E-mail: awolf@andrew.cmu.edu
- Benny Van Houdt is with the Mathematics and Computer Science Department, University of Antwerp. E-mail: benny.vanhoudt@uantwerpen.be
- This work is supported by the Siebel Scholars Foundation, by a Google Anita Borg Memorial Scholarship, by NSF-CMMI-1538204, NSF-CMMI-1334194, and NSF-XPS-1629444, by a Google Faculty Research Award, by a Facebook Faculty Research Award, and by the FWO grant G024514N. An earlier short version of this paper appeared in [9].

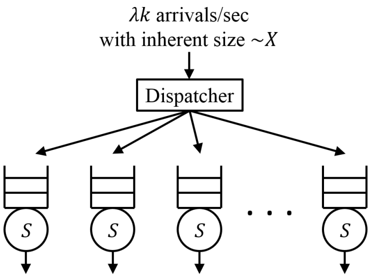


Fig. 1. The $S\&X$ model. The system has k servers and jobs arrive as a Poisson process with rate λk. Each job has an inherent size X. When a job runs on a server it experiences slowdown S. A job's running time on a single server is $R(1) = X \cdot S$. When a job runs on multiple servers, its inherent size X is the same on all these servers and it experiences a different, independently drawn instance of S on each server.

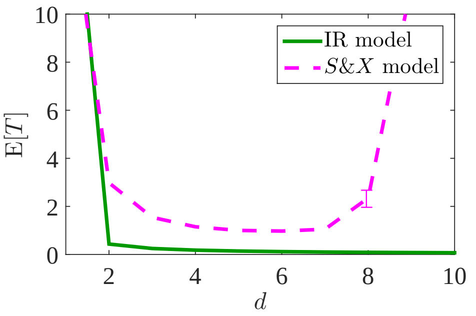


Fig. 2. Under Redundancy-d, each arriving job sends copies to d servers chosen uniformly at random. Here we show mean response time, $\mathbf{E}[T]$, as a function of d under Redundancy-d when the system has $k = 1000$ servers, the total arrival rate is λk where $\lambda = 0.7$, and inherent job sizes, X, follow a two-phase hyperexponential distribution with balanced means, $\mathbf{E}[X] = \frac{1}{4.7}$, and $C_X^2 = 10$. In the IR model (solid green line, from analysis in [10]), each job draws an i.i.d. instance of X on each server. As d increases, mean response time decreases. In the $S\&X$ model (dashed pink line, simulated; 95% confidence intervals are within the line unless shown), a job draws a single instance of X which is the same on all servers, and an i.i.d. instance of S on each server. Here S is an empirically measured distribution described in Section 3. While in the $S\&X$ model mean response time initially decreases as a function of d, as d becomes high the system eventually becomes unstable.

which replicates every arriving job to d queues [10]. Under the IR model assumed in [10], mean response time only decreases as we increase d. By contrast, in the $S\&X$ model, mean response time under Redundancy-d can improve significantly as we add a small amount of redundancy, but the system (typically) eventually becomes unstable because of the increased load of replication, sending mean response time to infinity (see Figure 2). Unfortunately, in general we cannot determine the values of d that will lead to good performance or to instability, because providing a performance analysis of policies like Redundancy-d in the $S\&X$ model is an open problem that is likely very difficult (analyzing Redundancy-d even within the IR model requires a very complex state space, since one needs to track all the copies of every job in every queue, see [10]).

The difficulty in tuning and potential instability of Redundancy-d in the $S\&X$ model motivate us to look for new redundancy policies which are less sensitive to d, are robust in that they provably will not go into overload, and are analytically tractable within the $S\&X$ model. To this end, we introduce a new redundancy policy, called Redundant-to-Idle-Queue (RIQ). This policy is similar to Redundancy-d in that every arrival queries d servers. However replicas are only made to those servers which are idle. If no server is idle, then the job is sent to a random one of the d servers (with no additional replicas). We provide an analytical approximation for the transform of response time under RIQ as a function of d. Our analysis allows for any distribution of S, any distribution of X, and any cancellation time. Our analysis matches simulation very well, provided that d is small relative to the total number of servers, which is certainly typical in practice. Most importantly, our analysis shows us that RIQ is extremely robust. While Redundancy-d can outperform RIQ at low d, we derive an analytical upper bound on the response time of RIQ under any S and X for any d that shows the system does not go into overload as d gets high.

The RIQ policy demonstrates that it is possible to design provably robust and analytically understood redundancy policies within the $S\&X$ model. But there is still room for improvement. We consider several modifications to the baseline policies, motivated by strategies used in practice. One idea that further reduces response time is to use Join-the-Shortest-Queue dispatching for jobs that do not find idle servers. Another idea is to replicate only small jobs; unfortunately this hurts RIQ, which is already quite conservative. On the other hand, replicating only small jobs can prevent instability under Redundancy-d, while still allowing Redundancy-d to achieve good performance at low d. As a compromise between Redundancy-d and RIQ, we propose the THRESHOLD-n policy, under which each arriving job polls d servers and replicates itself to those servers with fewer than n jobs in the queue. Like under RIQ, the system is stable under THRESHOLD-n for all d. Like under Redundancy-d, mean response time under THRESHOLD-n can be quite low at the best choice of d.

The remainder of this paper is organized as follows. In Section 2 we review prior theoretical and empirical work on redundancy. Section 3 introduces the $S\&X$ model. In Section 4 we discuss Redundancy-d and the difficulties it poses, and in Section 5 we introduce and analyze Redundant-to-Idle-Queue (RIQ). Sections 6 and 7 evaluate the performance of RIQ at low d and high d respectively. In Section 8 we discuss improvements upon Redundancy-d and RIQ. Section 9 studies an alternative setting where a job's extra copies are cancelled upon the start, rather than upon completion, of the first copy. Finally, in Section 10 we conclude.

2 Prior Work: The Gap Between Theory and Systems

Recently there has been growing interest in the theoretical community in analyzing systems with redundancy, with the goal of understanding how the number of copies per job affects response time. All of this theoretical work makes crucial simplifying assumptions for analytical tractability, most commonly assuming the IR model. In a few cases, other simplifications are adopted instead, including that the system has no queueing (i.e., it is an M/G/∞) or that all jobs replicate to all servers. As we will see below, these assumptions lead to results that are qualitatively different from those produced by empirical systems studies.

In the Redundancy-d system, every arriving job is replicated to d servers chosen at random, where d is a small constant compared to the number of servers [10]. The full distribution of response time as a function of d has been analyzed in closed form in the IR model [10]. Exact response time distributions also have been derived in the IR model in a system in which a job's class determines where it is replicated [11]. This result has been extended to networks of queues, still in the IR model [6]. In [22], jobs are composed of multiple tasks, where tasks may be replicated; again the analysis assumes the IR model. In all these works, when runtimes are exponential or more variable, mean response time decreases as the number of copies per job increases: it is optimal to replicate jobs at all servers. This is also in accordance with [15], [21].

In a redundancy model called the "(n, k) system," each job sends copies of itself to all n servers and waits for $k \leq n$ copies to complete service. Bounds and approximations for mean response time in the (n, k) system are derived in [13], [14], [20], [24] in the IR model. While [13] does consider a model in which a job's runtime consists of a deterministic component that is the same on all servers and an exponential component that is independent across servers, this model is only analyzed in a system where there is no queueing (i.e., an $\mathrm{M/G/\infty}$). In a variation called the "(n, k, r) system," in which each job sends copies to $r \leq n$ of the servers and waits for $k \leq r$ of these copies to complete, increasing the value of r decreases mean response time whenever runtimes are at least as variable as an exponential distribution [14], [20].

The story told by empirical systems work is more cautious. While theoretical results suggest that response time decreases as the number of copies increases, practical studies have shown that creating too many copies can lead to unacceptably high response times and even instability [23]. This gap emerges because the strong assumptions required for the above theoretical analysis do not hold in practice. Specifically, a large job remains large when replicated; hence, too much redundancy can lead to overload. Systems researchers have observed that in realistic settings, more sophisticated dispatching and scheduling policies are needed to leverage the potential benefits of redundancy. One idea is to replicate only small jobs to limit the amount of load added to the system; this can lead to a 46% reduction in mean response time [3]. In MapReduce systems, many algorithms begin running replicated copies of jobs only after waiting for some delay to identify which jobs are experiencing significant slowdown [5], [27]. Recent work builds on this idea by combining delayed execution of replicas with scheduling policies that reserve a set of servers on which to run replicas (assuming jobs are non-preemptible) [4], [19].

Our goal in this paper is to bring the theoretical models of redundancy systems closer to the real systems that theoretical work endeavors to analyze. To this end, we propose a new model called the $S\&X$ model that removes the problematic assumptions in the IR model. We revisit policies used in practice in Section 8, where we discuss these policies in the context of the $S\&X$ model.

TABLE 1
The Dolly(1,12) empirical slowdown distribution [3]. The server slowdown ranges from 1 to 12, with mean 4.7.

S	1	2	3	4	5	6	7	8	9	10	11	12
Prob.	0.23	0.14	0.09	0.03	0.08	0.10	0.04	0.14	0.12	0.021	0.007	0.002

3 The $S\&X$ Model

We consider a setting with k homogeneous servers (see Figure 1). Each server has its own queue and works in first-come first-served order. Jobs arrive as a Poisson process with rate $k\lambda$, where λ is a constant, and are dispatched immediately upon arrival. We assume that jobs are non-preemptible.

We introduce the $S\&X$ *model*, which captures the effects of both job-dependent and server-dependent factors on a job's runtime. Here S is a random variable denoting the *slowdown* that a job experiences at a server and X is the job's *inherent size*. A job that runs on one server has *runtime* $R(1) = X \cdot S$ (here S is assumed to be at least 1; S represents the factor by which the inherent size, X, is "stretched")[1]. Unlike in the IR model, in the $S\&X$ model runtimes are *not* independent across servers because a job's X component is the same on all servers.

When a server begins working on a job it draws a new independent instance of S. Thus consecutive jobs may see different slowdowns on the same server. This reflects the fact that slowdowns change over time and are not fixed for a particular server.

If a job is dispatched to a single server j, it completes service after time equal to its queueing time, T_j^Q, plus its running time, $X \cdot S_j$, where S_j is the instance of S that the job experiences on server j (S_j is drawn independently across servers for a given job, and across jobs for a given server). If a job is dispatched to all servers in some set $\mathcal{S}$, its response time is

$$T = \min_{j \in \mathcal{S}} \{T_j^Q + X \cdot S_j\}.$$

The $S\&X$ model takes a big departure from traditional queueing theory where there is a *single* random variable for "service time," making it impossible to differentiate between the inherent job size and the server slowdown components.

When a job that is running on multiple servers completes service on one of these servers, all of its remaining copies must be cancelled. We model the time it takes to cancel a copy as taking time Z, where Z is a random variable.

We use ρ to denote the system load. Unlike in traditional queueing systems, in systems with redundancy load and stability depend not only on the arrival rate and mean runtime, but also on the particular dispatching and scheduling policies. Hence we defer a more detailed discussion of load and stability to Sections 4 and 5, which introduce specific dispatching policies: Redundancy-d and Redundant-to-Idle-Queue.

In much of this paper, we focus on the Dolly(1,12) distribution (see Table 1), which was measured empirically by

[1] Alternatively, slowdown can be additive, where $R(1) = X + S$. The analysis and bounds in Sections 5 and 7 easily extend to this setting.

analyzing traces collected from Facebook's Hadoop cluster and Microsoft Bing's Dryad cluster and has mean 4.7 [3].[2]

4 The Redundancy-d Policy

A natural dispatching policy for systems with redundancy is *Redundancy-d* [10].

Definition 1. *Under **Redundancy-**d, each arriving job creates d copies of itself and sends these copies to d servers chosen uniformly at random without replacement.*

Figure 2 compares mean response time, $\mathbf{E}[T]$, under Redundancy-d in the IR model (from analysis, see [10]) to that in the $S\&X$ model (simulated; in the $S\&X$ model, we set $S \sim \text{Dolly}(1, 12)$). In the IR model, as d increases $\mathbf{E}[T]$ decreases provided runtimes are at least as variable as an exponential distribution. This is very different from what happens under the $S\&X$ model: while at first $\mathbf{E}[T]$ decreases as a function of d, as d becomes higher $\mathbf{E}[T]$ starts to increase and ultimately the system becomes unstable.

While Figure 2 shows that the system can become unstable under Redundancy-d in the $S\&X$ model if d is too high, it is difficult to prove this analytically. Typically a system is stable as long as the system load, ρ, is less than 1. We can think of ρ as being the arrival rate, λ, multiplied by the expected server capacity used per job. But in the $S\&X$ model, deriving the expected server capacity used per job is not straightforward because it requires knowing the average number of servers on which a job runs; this is not d because some copies may be cancelled while still in the queue. Furthermore, copies can enter service at different times on different servers, so knowing the number of servers on which a job runs is not enough to determine the duration of time for which the job occupies each server.

Figure 2 highlights the importance of making the right modeling assumptions. Unlike in the IR model, in the $S\&X$ model Redundancy-d is *not* robust to the choice of d; choosing the wrong value of d can lead to unacceptably poor performance or even instability. Unfortunately, the analysis of Redundancy-d in the $S\&X$ model remains an open problem, meaning that it is very difficult to know how to choose a good d.

The lack of robustness, difficulty in tuning, and potentially poor performance of Redundancy-d motivate the need for a better dispatching policy for the $S\&X$ model. In designing such a policy, it is important to avoid the factors that cause poor performance for Redundancy-d. In particular, Redundancy-d creates copies of jobs even when the system has no extra capacity with which to run these copies. Consequently, Redundancy-d can add too much load to the system, causing queue lengths to build up. An important consideration when designing dispatching policies for the $S\&X$ model therefore is to ensure that we do not add excessive load.

5 Redundant-to-Idle-Queue

Here we introduce the Redundant-to-Idle-Queue (RIQ) dispatching policy (Section 5.1).[3] We derive an approximation for the Laplace transform of response time under RIQ (Section 5.2), and evaluate the accuracy of our approximation by comparing our analytical results to simulation (Section 5.3).

5.1 The Redundant-to-Idle-Queue Dispatching Policy

Definition 2. *Under **Redundant-to-Idle-Queue** (RIQ), each arriving job queries d servers chosen uniformly at random without replacement. If all d queried servers are busy, the job joins the queue at one of the d servers chosen at random. If $0 < i \le d$ queried servers are idle, the job enters service at all i idle servers, and its runtime is $R(i) = X \cdot \min\{S_1, \ldots, S_i\}$.*

Under RIQ, the system load is $\rho = \lambda \cdot \mathbf{E}[I \cdot R(I)]$, where I is a random variable denoting the number of servers on which a job runs. Here ρ is the average arrival rate multiplied by the average service capacity used per job. Forming an expression for load under RIQ is easier than for Redundancy-d because under RIQ any job that runs on multiple servers occupies all of its servers for the same duration. Nonetheless, it is not immediately obvious how to compute $\mathbf{E}[I \cdot R(I)]$; we defer our derivation of ρ to the end of Section 5.2.

The analysis of RIQ relies on the Asymptotically Independent Idleness assumption, defined as follows:

Assumption 1. ***[Asymptotically Independent Idleness]*** *The servers are idle d-wise independently; that is,*

$$\Pr\{\text{server } s_d \text{ idle} \mid \text{servers } s_1, \ldots, s_{d-1} \text{ idle}\} = \Pr\{\text{server } s_d \text{ idle}\}$$

for all sets of d distinct servers $s_1, \ldots, s_d$.

In Section 5.3 we show that our analysis matches simulation when $d \ll k$, indicating that Assumption 1 is reasonable.

5.2 Analysis: Laplace Transform of Response Time

Our derivation of the transform of response time, $\widetilde{T}(s)$, under RIQ begins by conditioning on whether an arrival to the system finds any idle servers:

$$\begin{aligned} \widetilde{T}(s) &= \Pr\left\{\begin{matrix}\text{job finds}\\ \text{idle servers}\end{matrix}\right\} \cdot \widetilde{T}\left(s \,\middle|\, \begin{matrix}\text{job finds}\\ \text{idle servers}\end{matrix}\right) \\ &\quad + \Pr\left\{\begin{matrix}\text{job finds no}\\ \text{idle servers}\end{matrix}\right\} \cdot \widetilde{T}\left(s \,\middle|\, \begin{matrix}\text{job finds no}\\ \text{idle servers}\end{matrix}\right) \\ &= (1-\rho^d)\widetilde{T}\left(s \,|\text{job finds idle servers}\right) \\ &\quad + \rho^d \widetilde{T}\left(s \,|\text{job finds no idle servers}\right), \qquad (1) \end{aligned}$$

where the second line is due to Assumption 1.

We find $\widetilde{T}(s|\text{job finds idle servers})$ by further conditioning on the number of idle servers a job finds:

$$\begin{aligned} \widetilde{T}\left(s \,\middle|\, \begin{matrix}\text{job finds}\\ \text{idle servers}\end{matrix}\right) &= \sum_{i=1}^{d} \Pr\left\{\begin{matrix}\text{job finds } i\\ \text{idle servers}\end{matrix} \,\middle|\, \begin{matrix}\text{job finds}\\ \text{idle servers}\end{matrix}\right\} \cdot \widetilde{R(i)}(s) \\ &= \sum_{i=1}^{d} \frac{(1-\rho)^i \rho^{d-i} \binom{d}{i}}{1-\rho^d} \widetilde{R(i)}(s). \qquad (2) \end{aligned}$$

[2] The full distribution does not appear in [3]; we obtained this from personal communication with the authors.

[3] RIQ is motivated by the highly practical Join-Idle-Queue (JIQ) policy, under which each arrival is dispatched to a single idle queue, if one exists [17].

To find $\widetilde{T}(s|\text{job finds no idle servers})$, we observe that a job that finds no idle servers joins the queue at a single server, which has the following properties:

1) When the server is idle, arrivals form a Poisson process with rate λd. This is because the total arrival rate is λk, each arrival polls the server with probability $\frac{d}{k}$, and every job that polls the server while it is idle runs on it.
2) When the server is busy, arrivals form a Poisson process with rate $\lambda' = \lambda\rho^{d-1}$. This is because the total arrival rate is λk, each arrival polls the server with probability $\frac{d}{k}$, and runs on it if the job's $d-1$ other servers are also busy (ρ^{d-1}) and the job chooses our particular server ($\frac{1}{d}$).
3) All jobs that find the server idle have runtime R_0, where:
$$R_0 = \begin{cases} R(1) & \text{w.p. } \rho^{d-1} \\ R(i) + Z & \text{w.p. } (1-\rho)^{i-1}\rho^{d-i}\binom{d-1}{i-1},\ 1 < i \leq d. \end{cases} \tag{3}$$
Here we use the probability that the arrival found $i-1$ *other* idle servers among its $d-1$ other servers. Note that in practice, if a job runs on i servers then only $i-1$ servers need to incur a cancellation cost Z. We simplify the analysis by assuming that all i servers incur the cost.
4) All jobs that find the server busy have runtime $R(1)$.

We call the system described above an M*/G/1/efs. Here M* denotes that the arrival rate depends on whether the system is idle, and "efs" indicates that the system has an exceptional first service (i.e., the first job served during a busy period has a different service time distribution than all other jobs).

We now have:
$$\widetilde{T}\left(s \,\middle|\, \begin{matrix}\text{job finds no}\\ \text{idle servers}\end{matrix}\right) = \widetilde{R(1)}(s) \cdot \widetilde{T_Q}(s \mid \text{queueing})^{\mathrm{M}^*/\mathrm{G}/1/\mathrm{efs}}, \tag{4}$$
where $\widetilde{T_Q}(s|\text{queueing})$, the Laplace transform of time in queue given that a job waits in the queue, is
$$\widetilde{T_Q}(s \mid \text{queueing})^{\mathrm{M}^*/\mathrm{G}/1/\mathrm{efs}} = \frac{\widetilde{T}(s)^{\mathrm{M}^*/\mathrm{G}/1/\mathrm{efs}} - (1 - P_Q^{\mathrm{M}^*/\mathrm{G}/1/\mathrm{efs}}) \cdot \widetilde{R_0}(s)}{P_Q^{\mathrm{M}^*/\mathrm{G}/1/\mathrm{efs}} \cdot \widetilde{R(1)}(s)}, \tag{5}$$
and
$$P_Q^{\mathrm{M}^*/\mathrm{G}/1/\mathrm{efs}} = \frac{\mathbf{E}[N_B]}{\mathbf{E}[N_B] + 1} \tag{6}$$
is the probability of queueing in an M*/G/1/efs and
$$\mathbf{E}[N_B] = \lambda' \cdot \frac{\mathbf{E}[R_0]}{1 - \lambda'\mathbf{E}[R(1)]} \tag{7}$$
is the expected number of arrivals during an M*/G/1/efs busy period, and $\widetilde{T}(s)^{\mathrm{M}^*/\mathrm{G}/1/\mathrm{efs}}$ is given in Lemma 1.

Lemma 1. *The Laplace transform of response time in an M*/G/1/efs where the first job experiences service time R_0, all remaining jobs experience service time $R(1)$, the arrival rate while the server is idle is λd, and the arrival rate while the server is busy is λ' is the same as that in an M/G/1/efs where the first job experiences service time R_0, all remaining jobs experience service time $R(1)$, and the arrival rate is λ'.*

Proof. The M/G/1/efs response time analysis relies on PASTA, so it does not directly apply to the M*/G/1/efs, which has a state-dependent arrival rate. However, the arrival rate while the system is idle only affects how close together busy periods occur. It does not affect the response time of any of the jobs during the busy period. Thus to derive response time, we can view the system as an M/G/1/efs. □

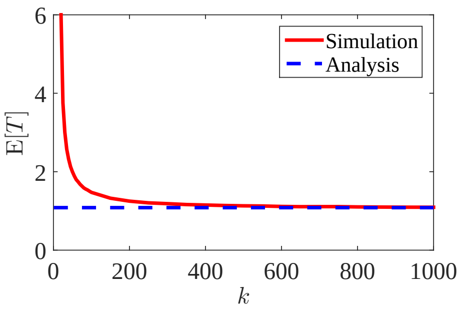


Fig. 3. $\mathbf{E}[T]$ under RIQ from analysis (dashed blue line) and simulation (solid red line, 99% confidence intervals are within the line) when $\lambda = 0.7$, $d = 20$, $X \sim H2$ with $\mathbf{E}[X] = \frac{1}{4.7}$ and $C_X^2 = 10$, $S \sim \mathrm{Dolly}(1, 12)$, and $Z = 0$. When $k = 1000$, the analysis is within 1% of simulation.

The response time in an M/G/1/efs is well understood (see [25]). For our system, we have
$$\widetilde{T}(s)^{\mathrm{M}^*/\mathrm{G}/1/\mathrm{efs}} = \frac{1 - \lambda'\mathbf{E}[R(1)]}{1 - \lambda'\mathbf{E}[R(1)] + \lambda'\mathbf{E}[R_0]} \cdot \frac{(\lambda' - s)\widetilde{R_0}(s) - \lambda'\widetilde{R(1)}(s)}{\lambda' - s - \lambda'\widetilde{R(1)}(s)}. \tag{8}$$

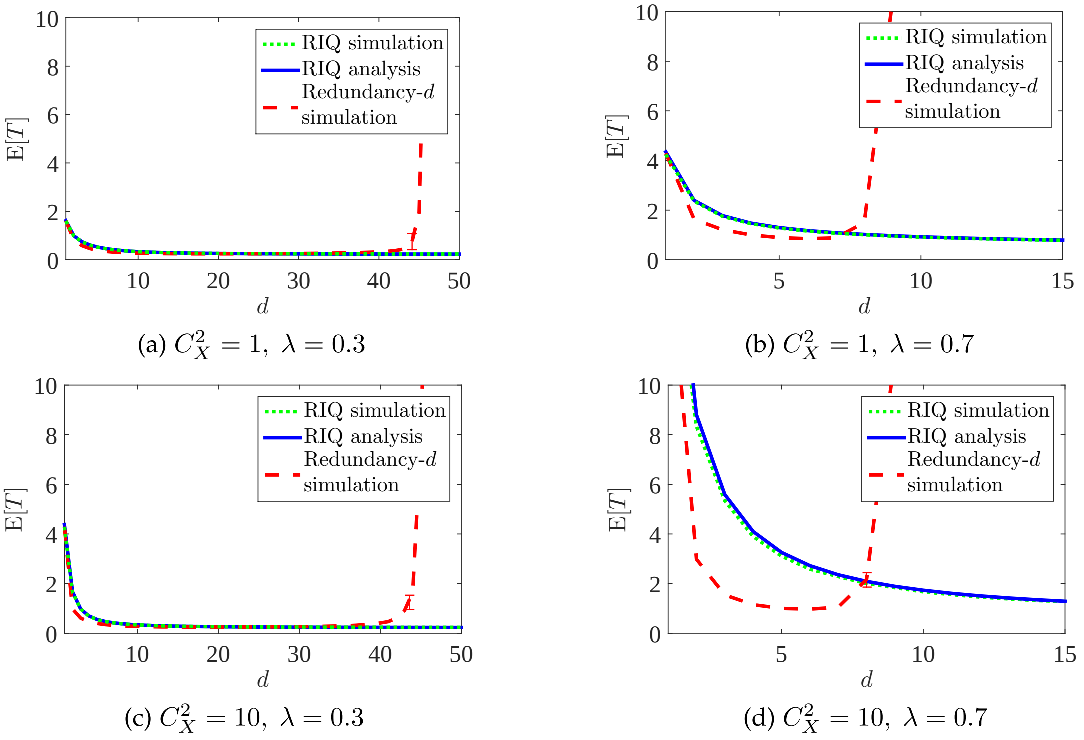
Combining equations (1)-(8) gives a closed-form expression for mean response time under RIQ in terms of ρ.

Finally, we derive ρ, the probability that a server is busy. In Section 3 we defined $\rho = \lambda \cdot \mathbf{E}[I \cdot R(I)]$, where I is the number of servers on which a job runs. Alternatively, we can compute ρ using renewal-reward theory, defining a cycle as the time from when a server becomes idle until it is next idle:
$$\rho = \frac{\mathbf{E}[B]}{\mathbf{E}[B] + \frac{1}{\lambda d}}, \tag{9}$$
where $\mathbf{E}[B] = \frac{\mathbf{E}[R_0]}{1 - \lambda'\mathbf{E}[R(1)]}$ is the expected busy period duration and $\frac{1}{\lambda d}$ is the mean interarrival time to an idle server. Note that $\mathbf{E}[R_0]$ is defined in terms of ρ, hence (9) defines ρ in terms of itself. When $d = 2$, the equation is of degree 2 and can be solved exactly; for higher d we solve for ρ numerically.

The system is stable as long as $\rho < 1$. However, without a closed-form expression for ρ, it is difficult to understand this stability condition intuitively. In Section 7 we derive an upper bound on mean response time under RIQ which gives us an alternative condition: the system is stable if $\lambda \cdot \mathbf{E}[R(1)] < 1$.

5.3 Quality of Approximation

Our approximation matches simulation quite well provided d is sufficiently small relative to k. Figure 3 compares mean response time from our analysis to that obtained via simulation when $d = 20$, $\lambda = 0.7$, X follows a two-phase hyperexponential distribution with balanced means



(a) $C_X^2 = 1,\ \lambda = 0.3$

(b) $C_X^2 = 1,\ \lambda = 0.7$

(c) $C_X^2 = 10,\ \lambda = 0.3$

(d) $C_X^2 = 10,\ \lambda = 0.7$

Fig. 4. $\mathbf{E}[T]$ vs. d under Redundancy-d (from simulation) and RIQ (from both simulation and analysis) for $X \sim H2$ with $\mathbf{E}[X] = \frac{1}{4.7}$ and $C_X^2 = 1$ (top row) and $C_X^2 = 10$ (bottom row), where $\mathbf{E}[R(1)] = \mathbf{E}[X] \cdot \mathbf{E}[S] = 1$, $S \sim$ Dolly(1,12), and the cancellation time is $Z = 0$, under low ($\lambda = 0.3$, left) and high ($\lambda = 0.7$, right) arrival rate. The simulations have $k = 1000$ servers; 95% confidence intervals are within the line where not shown. At low λ, RIQ and Redundancy-d perform similarly, but at high λ Redundancy-d becomes unstable when d is large, whereas RIQ continues to achieve low $\mathbf{E}[T]$.

and $\mathbf{E}[X] = \frac{1}{4.7}$ and $C_X^2 = 10$, and S follows the Dolly(1,12) server slowdown distribution. Our analysis is more accurate at high k; when $k = 1000$ the analysis is within 1% of simulation. When d and λ are lower the analysis converges to simulation more quickly.

6 Results: $d \ll k$

In this section we evaluate the performance of RIQ using our analysis from Section 5. Throughout the section, $\mathbf{E}[R(1)] = 1$, $k = 1000$ servers, S follows the Dolly(1,12) distribution (Table 1, $\mathbf{E}[S] = 4.7$), and X follows a two-phase hyperexponential distribution ($H2$) with balanced means:

$$X \sim \begin{cases} \text{Exp}(\mu_1) & \text{w.p. } p \\ \text{Exp}(\mu_2) & \text{w.p. } 1-p \end{cases},$$

where $\frac{p}{\mu_1} = \frac{1-p}{\mu_2}$.

We consider only the $d \ll k$ regime, in which our analysis provides a close approximation for performance under RIQ. In Section 7 we study what happens when d is close to k.

In Figure 4 we compare RIQ (our analysis in Section 5 matches simulation) to Redundancy-d (simulated) at both low and high λ when the inherent job size X has different squared coefficients of variation, C_X^2, and the cancellation time is $Z = 0$. When d is low, Redundancy-d outperforms RIQ because Redundancy-d allows all jobs to benefit from creating multiple copies, not just jobs finding multiple idle servers, and this outweighs any pain caused by the extra load added by these copies. However, as d gets higher, Redundancy-d becomes unstable, whereas mean response time under RIQ continues to decrease.

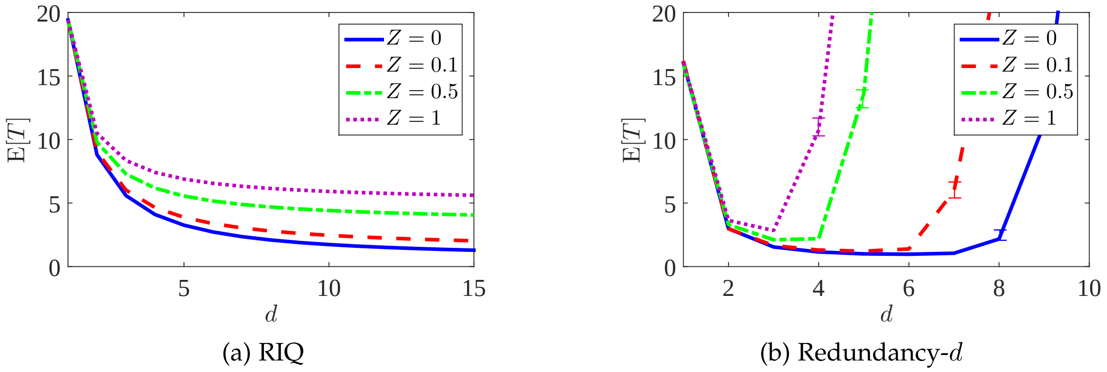
Under both RIQ and Redundancy-d, mean response time increases as the cancellation time Z increases (see Figure 5). Under Redundancy-d, as Z increases the system becomes unstable at lower values of d. Under RIQ, the system remains stable, indicating that RIQ is *more robust* than Redundancy-d.

Results are more dramatic at the tail of response time, which in computer systems is often a more important metric than the mean. We numerically invert the transform derived analytically in Section 5 to obtain the c.d.f. of response time using the procedure described in [1][4]. As d increases, the 95th percentile of response time, T_{95}, drops much more steeply than $\mathbf{E}[T]$ (see Figure 6), indicating that RIQ successfully overcomes the negative effects of the variable server slowdowns. In fact, all percentiles of response time are much lower at $d = 5$ than at $d = 1$ (see Figure 7).

The trends are similar when the server slowdown follows distributions other than the Dolly$(1, 12)$ distribution. In Figure 8 we show mean response time as a function of d under Redundancy-d and RIQ when S follows a Bimodal distribution, as was measured in a Google cluster [7]:

$$S \sim \begin{cases} 1 & \text{w.p. } 0.99 \\ 14 & \text{w.p. } 0.01 \end{cases}.$$

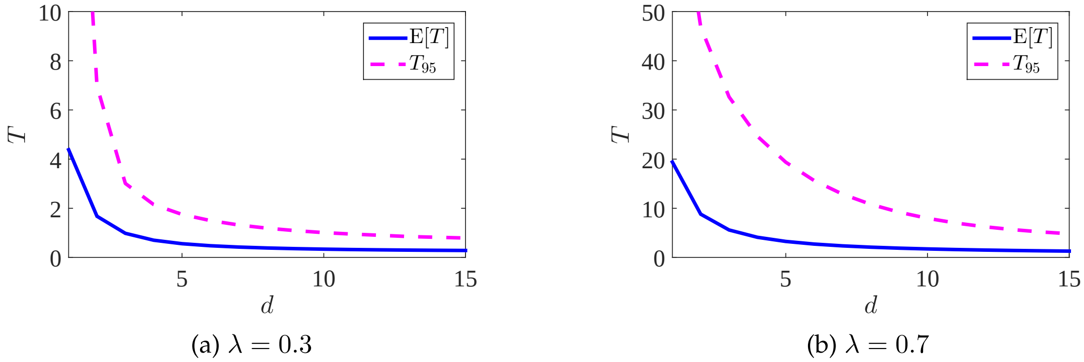
[4] We thank Jan-Pieter Dorsman for help with the inversion.



(a) RIQ

(b) Redundancy-d

Fig. 5. Mean response time as a function of d under (a) RIQ (from analysis) and (b) Redundancy-d (simulated with $k = 1000$ servers, 95% confidence intervals are within the line except where shown), $\lambda = 0.7$, job sizes are $X \sim H2$ with $\mathbf{E}[X] = \frac{1}{4.7}$ and $C_X^2 = 10$, and server slowdowns are $S \sim \text{Dolly}(1, 12)$ for different deterministic cancellation times Z. As Z increases, mean response time increases under both RIQ and Redundancy-d. Under Redundancy-d, this leads to the system becoming unstable at lower values of d, whereas RIQ remains stable for all values of Z.



(a) $\lambda = 0.3$

(b) $\lambda = 0.7$

Fig. 6. Mean (solid blue line) and 95th percentile (dashed pink line) of response time, T, (via analysis) under RIQ when $X \sim H2$ with $\mathbf{E}[X] = \frac{1}{4.7}$ and $C_X^2 = 10$, $S \sim \text{Dolly}(1, 12)$, $Z = 0$, and (a) $\lambda = 0.3$ and (b) $\lambda = 0.7$. As d increases, both the mean and the 95th percentile of response time decrease; the improvement is more pronounced in the tail.

With Bimodal slowdown, Redundancy-d is unstable at lower values of d than with the more uniform Dolly$(1, 12)$ distribution, but $\mathbf{E}[T]$ under both RIQ and Redundancy-d follows the same qualitative shape regardless of the distribution of S.

7 Results: High d

One might think that mean response time should be lowest when $d = k$, since when $d \ll k$ we saw that $\mathbf{E}[T]$ decreases as d increases. However, this intuitively does not make sense. As d increases, a job achieves decreasing marginal runtime and queueing time benefit from querying one more server. Eventually, increasing d only increases load without providing any benefit. As d gets high, one might think that the servers are always busy, driving queue lengths to infinity. Indeed, we see in Figure 9 that ρ (the probability that a server is busy, measured via simulation) approaches 1 as $d \to k$. But surprisingly, the system is never unstable, even when $d = k$. This is because RIQ limits the amount of added load. Effectively, when a server goes idle we can think of it as being given some extra work, where this extra work is some job's replicated copy. This causes the server to be always busy (sending ρ to 1) but it affects the queue length like a vacation time, which cannot cause instability.

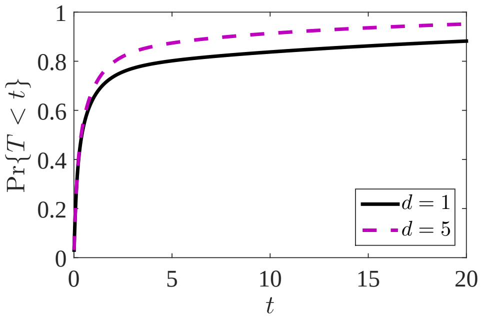
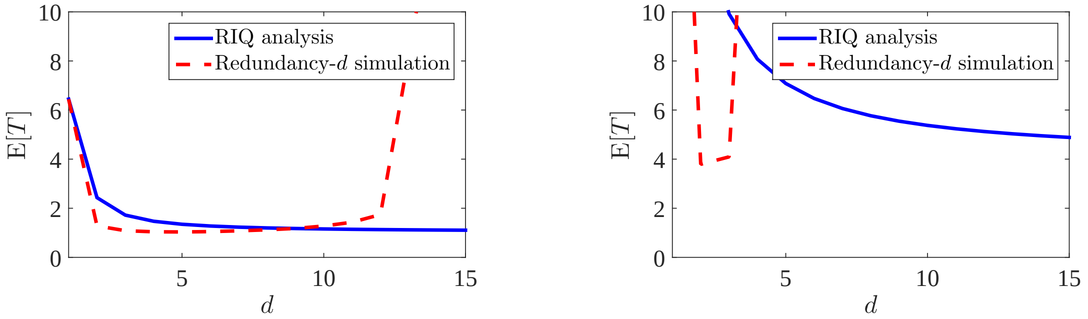


Fig. 7. Distribution of response time under RIQ when $d = 1$ (solid black line) and $d = 5$ (dashed purple line) when $S \sim \text{Dolly}(1, 12)$, $X \sim H2$ with $\mathbf{E}[X] = \frac{1}{4.7}$ and $C_X^2 = 10$, and $\lambda = 0.7$.

We formalize this intuition in Theorem 1, which gives an upper bound on $\mathbf{E}[T]$ under RIQ, for all d and for all k. Our bound uses an M/G/1/vacation system, which was introduced in [8] and is described in Appendix A.

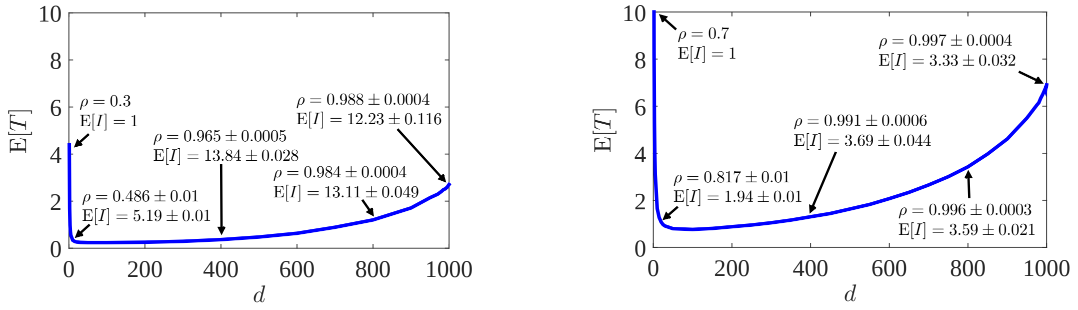
Theorem 1. *Assume that $\lambda \cdot \mathbf{E}[R(1)] < 1$. Then the response time under RIQ is upper bounded by the response time in an M/G/1/vacation queue with arrival rate λ, runtime $G = R(1) =$*



(a) $\lambda = 0.3$

(b) $\lambda = 0.7$

Fig. 8. Both RIQ and Redundancy-d exhibit similar trends to their performance when $S \sim \text{Dolly}(1, 12)$ under alternative S distributions. Here we show results for $S \sim \text{Bimodal}(1, 14; 0.99)$ ($\mathbf{E}[S] = 1.13$) when $X \sim H2$ with $\mathbf{E}[X] = \frac{1}{1.13}$ and $C_X^2 = 10$, and $Z = 0$, for (a) $\lambda = 0.3$ and (b) $\lambda = 0.7$. RIQ results are from analysis; Redundancy-d is simulated (95% confidence intervals are within the line).



(a) $\lambda = 0.3$

(b) $\lambda = 0.7$

Fig. 9. Mean response time under RIQ (simulated; 95% confidence intervals are within the line) as a function of d when $k = 1000$, $X \sim H2$ with $\mathbf{E}[X] = \frac{1}{4.7}$ and $C_X^2 = 10$, $S \sim \text{Dolly}(1, 12)$, $Z = 0$, and (a) $\lambda = 0.3$, (b) $\lambda = 0.7$. While mean response time increases as d gets close to k, even when $d = k$ mean response time is lower than when $d = 1$. The lines are annotated with the values of ρ and $\mathbf{E}[I]$ (the expected number of copies per job) at several different values of d. 99% confidence intervals are within the line for $\mathbf{E}[T]$ and are explicitly shown for ρ and $\mathbf{E}[I]$.

$S \cdot X$, and vacation time $R(1) + Z$:

$$\begin{aligned}\mathbf{E}[T]^{\text{RIQ}} &\le \mathbf{E}[T]^{\text{M/G/1/vacation}} \\ &= \mathbf{E}[T]^{\text{M/G/1}} + \mathbf{E}[(R(1)+Z)_e].\end{aligned}$$

where $(R(1)+Z)_e$ denotes the excess of $R(1)+Z$.

Proof. We will begin by expressing mean response time under RIQ by conditioning on whether a tagged arrival to the system finds any idle servers. We have

$$\begin{aligned}\mathbf{E}[T]^{\text{RIQ}} = \mathbf{P}\left\{\begin{matrix}\text{tagged job finds}\\ \text{idle servers}\end{matrix}\right\} \cdot \mathbf{E}\left\{T \,\middle|\, \begin{matrix}\text{tagged job finds}\\ \text{idle servers}\end{matrix}\right\} \\ + \mathbf{P}\left\{\begin{matrix}\text{tagged job finds}\\ \text{no idle servers}\end{matrix}\right\} \cdot \mathbf{E}\left\{T \,\middle|\, \begin{matrix}\text{tagged job finds}\\ \text{no idle servers}\end{matrix}\right\}.\end{aligned}$$

When the tagged job arrives to the system, if it finds i idle servers it enters service on all of these servers. For all i, we have

$$\begin{aligned}\mathbf{E}\left[T \,\middle|\, \begin{matrix}\text{tagged job finds}\\ i \text{ idle servers}\end{matrix}\right] &= \mathbf{E}[R(i)] \\ &\le \mathbf{E}[R(1)] \\ &\le \mathbf{E}[R(1)] + \mathbf{E}\left[T^Q \,\middle|\, \begin{matrix}\text{tagged job finds}\\ \text{no idle servers}\end{matrix}\right] \\ &= \mathbf{E}\left[T \,\middle|\, \begin{matrix}\text{tagged job finds}\\ \text{no idle servers}\end{matrix}\right].\end{aligned}$$

Hence we have the following upper bound:

$$\mathbf{E}[T]^{\text{RIQ}} \le \mathbf{E}[T \mid \text{tagged job finds no idle servers}]. \qquad (10)$$

Given that a tagged arrival found no idle servers, it joins the queue at a randomly chosen server with the following properties:

1) When the server is busy, jobs arrive with rate $k\lambda \cdot \frac{d}{k} \cdot \mathbf{P}\{\text{job finds other } d-1 \text{ servers busy}\} \cdot \frac{1}{d} \le \lambda$.
2) All jobs that arrive to the server while it is busy experience runtime $R(1)$.
3) A job that arrives to the server while it is idle, and finds $i-1$ other idle servers, experiences runtime $R(i)$ and possibly a cancellation time Z, hence keeps the server busy for time $R(i) + Z \le_{st} R(1) + Z$.

Since (10) forces all jobs to have a non-zero queueing time in our upper bound, the first job in the busy period (the job that arrives to the server while it is idle) can be viewed as a "dummy" job that does not contribute to response time. Every time a server goes idle, we can imagine that the server is forced to work on a "dummy" job with size $R(1) + Z$, so the server is always busy when the next real job arrives. This exactly describes an M/G/1/vacation system with arrival rate λ, runtime $R(1)$, and vacation time $R(1) + Z$. □

Theorem 2. *Under RIQ, the system is stable for all d if $\lambda \mathbf{E}[R(1)] < 1$.*

Proof. The stability region for the M/G/1/vacation is the same as that for the M/G/1, namely $\lambda \cdot \mathbf{E}[R(1)] < 1$. If the M/G/1/vacation is stable, then RIQ is stable since the M/G/1/vacation gives an upper bound on RIQ. □

8 Improving Redundancy Policies

In this section we study several policy variations designed to improve performance relative to the baseline RIQ and Redundancy-d policies. In some cases, the analysis presented in Section 5 extends easily to the variations on RIQ. In other cases, we develop new response time analysis. When analysis is not feasible, we study performance via simulation.

8.1 JSQ: Better Dispatching for Jobs that Find No Idle Servers

One weakness of RIQ is that when a job finds no idle servers it is dispatched to a single queue chosen at random, which is known to yield poor performance. A superior dispatching policy is Join-the-Shortest-Queue (JSQ), under which each arriving job joins the queue containing the fewest jobs. A practical variant of JSQ is JSQ-d, where each arriving job polls d queues chosen at random and joins the queue containing the fewest jobs among the d polled queues. JSQ-d is motivated by the fact that polling all k queues can be expensive. The RIQ+JSQ policy combines RIQ with JSQ-d dispatching.

Definition 3. *Under **RIQ+JSQ**, each arriving job polls d servers chosen uniformly at random without replacement. If $1 \le i \le d$ of the servers are idle, the job enters service at all i idle servers. If all d servers are busy, the job joins the queue with the fewest jobs among the d polled queues.*

Observation 1. *RIQ+JSQ should have the same stability region as RIQ, since the only difference between the two policies is that under RIQ+JSQ we achieve better load balancing of the jobs that do not find idle servers. Hence under RIQ+JSQ, the system should be stable as long as $\lambda \cdot \mathbf{E}[X \cdot S] < 1$.*

Response Time Analysis

Our analysis of mean response time under RIQ+JSQ follows the approach used in [2]. We write a system of equations that describes the evolution of the RIQ+JSQ system, then use numerical methods to approximate the steady-state behavior. It is easy to modify the analysis presented below to include a cancellation time.

Let $Y_{\ell,i}(t, t+r)$ be the fraction of servers with at least ℓ jobs in the queue (including the job in service, if there is one) at time t such that the job in service at time t is still in service at time $t+r$ and is running on exactly i servers. Let $Y_\ell(t, t+r) = \sum_{i=1}^{d} Y_{\ell,i}(t, t+r)$.

For $\ell, i > 1$, there are two ways in which a server can contribute to $Y_{\ell,i}(t, t+r)$:

1) The queue length was at least ℓ at time 0 and the same job remains in service on i servers during time interval $(0, t+r)$.
2) At some time $u < t$ an arrival caused the queue length to go from $\ell - 1$ to ℓ and the same job is in service on i servers during the time interval $(u, t+r)$.

When $i = 1$ there is a third case: a job completed service at time $u \le t$ and left behind at least ℓ jobs, and the next job stays in service during time $(u, t+r)$. Hence for $\ell > 1$,

$$\begin{aligned} Y_{\ell,i}(t, t+r) &= Y_{\ell,i}(0, t+r) \\ &\quad + \lambda \int_{u=0}^{t} (Y_{\ell-1}(u,u)^d - Y_\ell(u,u)^d) \\ &\qquad \cdot \frac{Y_{\ell-1,i}(u, t+r) - Y_{\ell,i}(u, t+r)}{Y_{\ell-1}(u,u) - Y_\ell(u,u)} du \\ &\quad + I_{i=1} \int_{u=0}^{t} (-\partial_r Y_{\ell+1}(u,u)) \bar{G}(t+r-u) du, \end{aligned} \tag{11}$$

where $\bar{G}(x)$ is the probability that a job's runtime exceeds x and

$$-\partial_r Y_{\ell+1}(u,u) = \lim_{r \to 0} \frac{Y_{\ell+1}(u,u) - Y_{\ell+1}(u, u+r)}{r}$$

is the service completion rate of jobs in a queue with length at least $\ell + 1$ at time u.

When $\ell = 1$ the second term, which corresponds to a new arrival, changes because now the arrival joins an idle queue and therefore runs on $1 \le i \le d$ servers:

$$\begin{aligned} Y_{1,i}(t, t+r) &= Y_{1,i}(0, t+r) \\ &\quad + i\lambda \int_{u=0}^{t} \binom{d}{i} Y_1(u,u)^{d-i} \\ &\qquad \cdot (1 - Y_1(u,u))^i \bar{G}^{(i)}(t+r-u) du \\ &\quad + I_{i=1} \int_{u=0}^{t} (-\partial_r Y_2(u,u)) \bar{G}(t+r-u) du, \end{aligned} \tag{12}$$

where $\bar{G}^{(i)}(x)$ is the probability that a job's runtime exceeds x given that the job is running on exactly i servers.

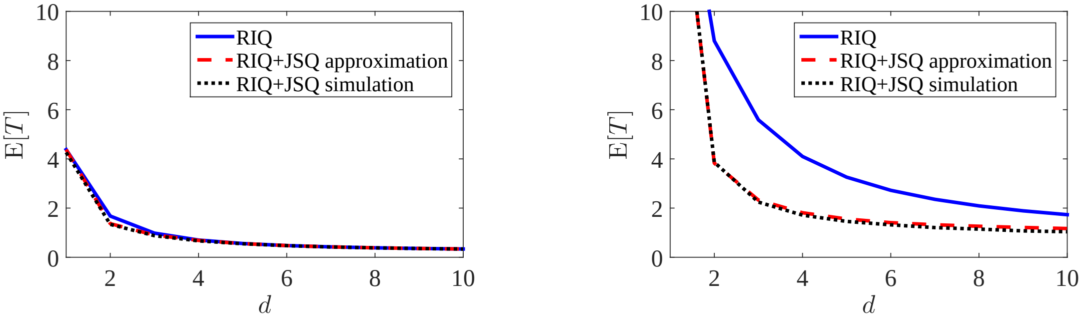
Summing over i in (11), we find

$$\begin{aligned} Y_\ell(t, t+r) &= Y_\ell(0, t+r) \\ &\quad + \lambda \int_{u=0}^{t} (Y_{\ell-1}(u,u)^d - Y_\ell(u,u)^d) \\ &\qquad \cdot \frac{Y_{\ell-1}(u, t+r) - Y_\ell(u, t+r)}{Y_{\ell-1}(u,u) - Y_\ell(u,u)} du \\ &\quad + \int_{u=0}^{t} (-\partial_r Y_{\ell+1}(u,u)) \bar{G}(t+r-u) du, \end{aligned} \tag{13}$$

which corresponds to equation (8) in [2] when $d = 2$. Summing over i in (12) yields

$$\begin{aligned} Y_1(t, t+r) &= Y_1(0, t+r) \\ &\quad + \lambda \int_{u=0}^{t} \sum_{i=1}^{d} \binom{d}{i} Y_1(u,u)^{d-i} \\ &\qquad \cdot (1 - Y_1(u,u))^i i \bar{G}^{(i)}(t+r-u) du \\ &\quad + I_{i=1} \int_{u=0}^{t} (-\partial_r Y_2(u,u)) \bar{G}(t+r-u) du. \end{aligned} \tag{14}$$

We now use equations (13) and (14) to numerically estimate π_ℓ, the steady-state probability that a server has at least ℓ jobs (assuming the number of servers k is large). Note that $\pi_1 = \rho$ is the probability that a server is busy. We introduce a mesh of width δ and approximate $Y_\ell(t, r)$ for $t \in [0, \delta, 2\delta, \ldots, t_0]$ and $r \in [0, \delta, 2\delta, \ldots, r_0]$. Here r_0 is chosen such that $\bar{G}(r_0)$ is negligible, and t_0 is chosen dynamically so that $\sum_{\ell,r} |Y_\ell(t+\delta, t+\delta+r) - Y_\ell(t, t+r)|$



(a) $\lambda = 0.3$

(b) $\lambda = 0.7$

Fig. 10. Comparing baseline RIQ and RIQ+JSQ from our approximation (dashed red line) and simulation (dotted black line; 95% confidence intervals are within the line). Here $S \sim \text{Dolly}(1, 12)$, $X \sim H2$ with $\mathbf{E}[X] = \frac{1}{4.7}$ and $C_X^2 = 10$, $Z = 0$, and (a) $\lambda = 0.3$ and (b) $\lambda = 0.7$.

is sufficiently small, e.g., 10^{-10}. We also truncate the queue lengths to at most n_0 jobs.

We start with an empty system at time 0, so $Y_\ell(0, r) = 0$ for all ℓ and r, and compute $Y_\ell(t+\delta, t+r+\delta)$ based on $Y_\ell(t, t+r)$ as follows:

$$\begin{aligned}
Y_1(t+\delta, t+r+\delta) &= Y_1(t, t+r+\delta) \\
&\quad + \lambda \int_{u=t}^{t+\delta} \sum_{i=1}^{d} \binom{d}{i} \cdot Y_1(u,u)^{d-i} \\
&\qquad \cdot (1 - Y_1(u,u))^i i \bar{G}(t+r+\delta-u) du \\
&\quad + \int_{u=t}^{t+\delta} (-\partial_r Y_2(u,u)) \bar{G}(t+r+\delta-u) du \\
&\approx Y_1(t, t+r+\delta) - (Y_2(t, t+\delta) - Y_2(t,t))\bar{G}(r + \frac{\delta}{2}) \\
&\quad + \lambda\delta \sum_{i=1}^{d} \binom{d}{i} Y_1(t,t)^{d-i}(1 - Y_1(t,t))^i i \bar{G}^{(i)}(r + \frac{\delta}{2})
\end{aligned}$$

$$\begin{aligned}
Y_\ell(t+\delta, t+r+\delta) &\approx Y_\ell(t, t+r+\delta) \\
&\quad - (Y_{\ell+1}(t, t+\delta) - Y_{\ell+1}(t,t))\bar{G}(r + \frac{\delta}{2}) \\
&\quad + \lambda\delta \frac{Y_{\ell-1}(t,t)^d - Y_\ell(t,t)^d}{Y_{\ell-1}(t,t) - Y_\ell(t,t)} (Y_{\ell-1}(t, t+r) - Y_\ell(t, t+r)).
\end{aligned}$$

The numerical scheme now uses these two approximations in the following manner.[5] First, we define

$$\begin{aligned}
\bar{Y}_\ell(t+\delta, t+r+\delta) &= \hat{Y}_\ell(t, t+r+\delta) \\
&\quad - (\hat{Y}_{\ell+1}(t, t+\delta) - \hat{Y}_{\ell+1}(t,t))\bar{G}(r + \frac{\delta}{2}) \\
&\quad + \lambda\delta \frac{\hat{Y}_{\ell-1}(t,t)^d - \hat{Y}_\ell(t,t)^d}{\hat{Y}_{\ell-1}(t,t)^d - \hat{Y}_\ell(t,t)} (\hat{Y}_{\ell-1}(t, t+r) - \hat{Y}_\ell(t, t+r)),
\end{aligned}$$

and subsequently we compute $\hat{Y}_\ell(t+\delta, t+r+\delta)$ as

$$\begin{aligned}
\hat{Y}_\ell(t+\delta, t+r+\delta) &= \hat{Y}_\ell(t, t+r+\delta) \\
&\quad - (\bar{Y}_{\ell+1}(t+\delta, t+2\delta) - \bar{Y}_{\ell+1}(t+\delta, t+\delta))\bar{G}(r + \frac{\delta}{2}) \\
&\quad + \lambda\delta \frac{\bar{Y}_{\ell-1}(t+\delta, t+\delta)^d - \bar{Y}_\ell(t+\delta, t+\delta)^d}{\bar{Y}_{\ell-1}(t+\delta, t+\delta) - \bar{Y}_\ell(t+\delta, t+\delta)} \\
&\qquad \cdot (\bar{Y}_{\ell-1}(t+\delta, t+r+\delta) - \bar{Y}_\ell(t+\delta, t+r+\delta)).
\end{aligned}$$

A similar approach is used to compute $\hat{Y}_1(t+\delta, t+\delta+r)$.

We now use $\hat{Y}_\ell(t,t)$ to approximate π_ℓ. The expected number of jobs in the system, $\mathbf{E}[N]$, is

$$\mathbf{E}[N] = \sum_{\ell=1}^{\infty} \pi_{ell} - \lambda \sum_{i=2}^{d} \binom{d}{i} \pi_1^{d-i}(1-\pi_1)^i (i-1) \mathbf{E}\left[G^{(i)}\right],$$

where $\binom{d}{i}\pi_1^{d-i}(1-\pi_1)^i$ is the probability that a job runs on i servers. Note that the second term is included to avoid overcounting jobs that run on $i > 1$ servers. Finally, we find $\mathbf{E}[T]$ by applying Little's Law.

Validation of Approximation

We compare our computed π_ℓ values with simulation; when $k = 1000$, $d = 10$, $C_X^2 = 10$, $\lambda = 0.7$, and $S \sim \text{Dolly}(1, 12)$, and for $\delta = 0.2$ and $n_0 = 10$, the approximation is within 5% of simulation for π_1, π_2, and π_3 (queue lengths exceed 3 less than 0.5% of the time). The approximation is more accurate when λ, d, C_X^2, and δ are lower and when k is higher.

Performance

We use our new analysis to compare RIQ+JSQ with baseline RIQ, assuming d is small (for both policies, this is the regime in which our analysis holds). When λ is low, RIQ+JSQ provides very little benefit over RIQ because most jobs find idle servers (see Figure 10). When λ is high, RIQ+JSQ beats RIQ because many jobs do not find idle servers and benefit from being dispatched via JSQ rather than randomly.

Note that it does not make sense to combine JSQ with Redundancy-d; this would require either polling $> d$ servers per job, or limiting the number of replicas to be $< d$.

8.2 SMALL: Replicate Only Jobs with Small X

A major concern when replicating jobs is that running multiple copies of the same job adds load to the system. This is particularly true of jobs with a large inherent size X. When a large job runs on many servers, it clogs up these servers for a long time even if it experiences a small slowdown S on some server. This makes it harder for jobs with small X components to find any idle servers on which to run.

[5] We thank Xingjie Li for helpful suggestions to stabilize the numerical scheme.

One way to prevent the small jobs from being blocked on many servers by large jobs is to allow only jobs with a small X component to create replicas. That is, we define a constant threshold x and only replicate jobs with inherent size $X \leq x$.

Definition 4. *Under **RIQ+SMALL**, if an arriving job has inherent size $X \leq x$ it polls d servers chosen uniformly at random without replacement. If $1 \leq i \leq d$ of the polled servers are idle, the job replicates itself on all i idle servers. If none are idle, the job joins the queue at one of the d servers chosen uniformly at random. If the job has inherent size $X > x$, it is dispatched to a single server chosen uniformly at random.*

Definition 5. *Under **Redundancy-d+SMALL**, if a job has inherent size $X \leq x$ it replicates itself to d servers chosen uniformly at random, whereas if a job has inherent size $X > x$ it is dispatched to a single server chosen uniformly at random.*

Theorem 3. *Under RIQ+SMALL, the system is stable if $\lambda \cdot \mathbf{E}[X \cdot S] < 1$.*

Proof. The approach is the same as the approach used to prove stability under the original RIQ policy; details can be found in Appendix B. □

Response Time Analysis: RIQ+SMALL

The analysis of response time under RIQ+SMALL follows the same approach as that under the baseline RIQ policy. We begin by conditioning on a job's inherent size:

$$\widetilde{T}(s) = \widetilde{T}(s \mid X \leq x) \cdot \mathbf{P}\{X \leq x\} + \widetilde{T}(s \mid X > x) \cdot \mathbf{P}\{X > x\}.$$

To find $\widetilde{T}(s \mid X > x)$, observe that any job with inherent size larger than x is dispatched randomly to a single M*/G/1/efs. As before, the exceptional first service comes from the fact that the first job in the busy period may run on anywhere between 1 and d servers. However, now the job's response time is *not* simply the response time in an M*/G/1/efs because jobs with size $X > x$ are less likely to be the first job in a busy period than jobs with size $X < x$. So we condition on whether the job finds the server idle:

$$\begin{aligned}\widetilde{T}(s \mid X > x) &= \widetilde{T}\left(s \,\middle|\, \begin{matrix} X > x \text{ \& job} \\ \text{finds idle servers} \end{matrix}\right) \cdot \mathbf{P}\left\{\begin{matrix}\text{job finds} \\ \text{idle servers}\end{matrix}\right\} \\ &\quad + \widetilde{T}\left(s \,\middle|\, \begin{matrix} X > x \text{ \& job finds} \\ \text{no idle servers} \end{matrix}\right) \cdot \mathbf{P}\left\{\begin{matrix}\text{job finds no} \\ \text{idle servers}\end{matrix}\right\} \\ &= \widetilde{R(1)}(s \mid X > x) \cdot (1-\rho) \\ &\quad + \widetilde{T_Q}(s \mid \text{queueing})^{\mathrm{M^*/G/1/efs}} \cdot \widetilde{R(1)}(s \mid X > x) \cdot \rho.\end{aligned}$$

The arrival and service rates in this M*/G/1/efs differ from those in the original RIQ analysis, and are discussed below.

To find $\widetilde{T}(s \mid X \leq x)$, we follow the original RIQ analysis:

$$\begin{aligned}\widetilde{T}(s \mid X \leq x) &= \widetilde{T}\left(s \,\middle|\, \begin{matrix} X \leq x \text{ \& job} \\ \text{finds idle servers} \end{matrix}\right) \cdot \mathbf{P}\left\{\begin{matrix}\text{job finds} \\ \text{idle servers}\end{matrix}\right\} \\ &\quad + \widetilde{T}\left(s \,\middle|\, \begin{matrix} X \leq x \text{ \& job finds} \\ \text{no idle servers} \end{matrix}\right) \cdot \mathbf{P}\left\{\begin{matrix}\text{job finds no} \\ \text{idle servers}\end{matrix}\right\},\end{aligned}$$

where $\mathbf{P}\{\text{job finds idle servers}\} = 1 - \rho^d$ and ρ is computed numerically using the same approach as in the original RIQ analysis and the M*/G/1/efs parameterization given below.

Similarly to the original analysis, we have:

$$\begin{aligned}\widetilde{T}\left(s \,\middle|\, \begin{matrix} X \leq x \text{ \& job} \\ \text{finds idle servers} \end{matrix}\right) &= \sum_{i=1}^{d} \mathbf{P}\left\{\begin{matrix}\text{job finds } i \\ \text{idle servers}\end{matrix} \,\middle|\, \begin{matrix}\text{job finds} \\ \text{idle servers}\end{matrix}\right\} \\ &\qquad \cdot \widetilde{R(i)}(s \mid X \leq x) \\ &= \sum_{i=1}^{d} \frac{(1-\rho)^i \rho^{d-i} \binom{d}{i}}{1-\rho^d} \cdot \widetilde{R(i)}(s \mid X \leq x),\end{aligned}$$

where $\widetilde{R(i)}(s \mid X \leq x) = X \cdot \min\widetilde{\{S_1, \ldots, S_i\}}(s \mid X \leq x)$.

If a job with $X < x$ finds no idle servers, it simply joins the queue at a single server, which behaves like an M*/G/1/efs:

$$\begin{aligned}\widetilde{T}\left(s \,\middle|\, \begin{matrix} X \leq x \text{ \& job finds} \\ \text{no idle servers} \end{matrix}\right) &= \widetilde{T}(s \mid X \leq x)^{\mathrm{M^*/G/1/efs}} \\ &= \widetilde{R(1)}(s \mid X \leq x) \cdot \widetilde{T_Q}(s)^{\mathrm{M^*/G/1/efs}}.\end{aligned}$$

In this system, our M*/G/1/efs has the following parameters:

1) When the server is idle, arrivals form a Poisson process with rate
$$\begin{aligned}&\mathbf{P}\{X > x\} \cdot \lambda k \cdot \frac{1}{k} + \mathbf{P}\{X \leq x\} \cdot \lambda k \cdot \frac{d}{k} \\ &\qquad = \lambda(\mathbf{P}\{X > x\} + \mathbf{P}\{X \leq x\} \cdot d).\end{aligned}$$

2) When the server is busy, arrivals form a Poisson process with rate
$$\begin{aligned}&\mathbf{P}\{X > x\} \cdot \lambda k \cdot \frac{1}{k} + \mathbf{P}\{X \leq x\} \cdot \lambda k \cdot \frac{d}{k} \cdot \rho^{d-1} \cdot \frac{1}{d} \\ &\qquad = \lambda(\mathbf{P}\{X > x\} + \mathbf{P}\{X \leq x\} \cdot \rho^{d-1}).\end{aligned}$$

3) Jobs that find the server idle experience runtime
$$R_0 = \begin{cases} R(1) \mid X > x & \text{w.p. } \frac{\mathbf{P}\{X > x\}}{\mathbf{P}\{X > x\} + d \cdot \mathbf{P}\{X \leq x\}} \\ R(i) + Z \mid X \leq x & \text{w.p. } \frac{d \cdot \mathbf{P}\{X \leq x\}}{\mathbf{P}\{X > x\} + d \cdot \mathbf{P}\{X \leq x\}} \\ & \quad \cdot (1-\rho)^{i-1} \rho^{d-i} \binom{d-1}{i-1}, \\ & \quad 1 \leq i \leq d \end{cases}$$

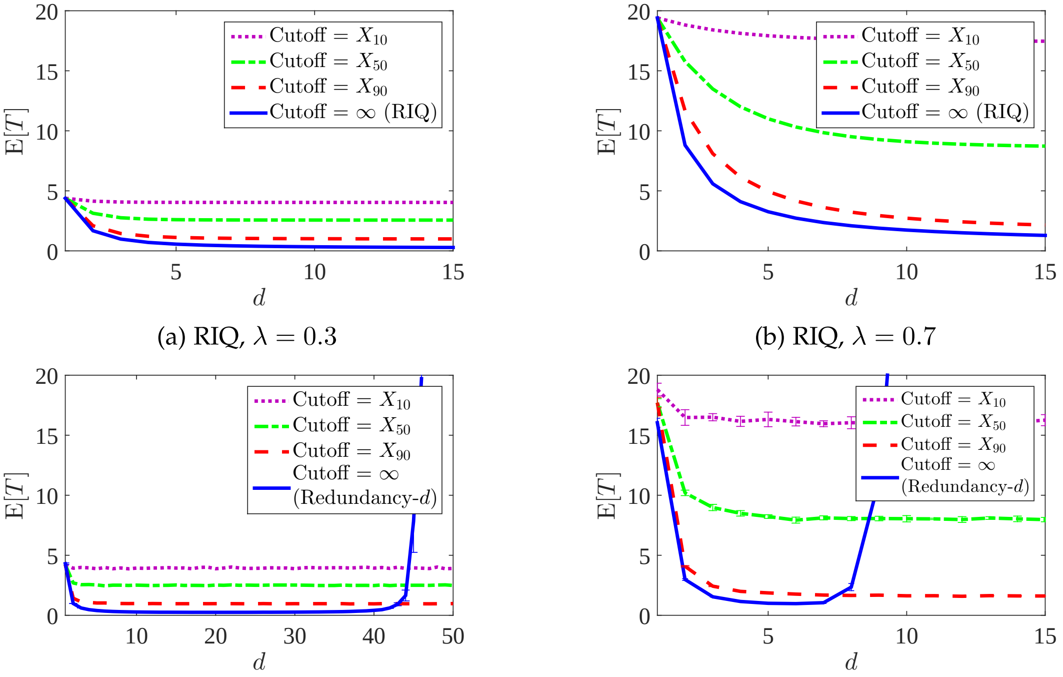
4) Jobs that find the server busy experience runtime
$$R_b = \begin{cases} R(1) \mid X > x & \text{w.p. } \frac{\mathbf{P}\{X > x\}}{\mathbf{P}\{X > x\} + \mathbf{P}\{X \leq x\} \cdot \rho^{d-1}} \\ R(1) \mid X \leq x & \text{w.p. } \frac{\mathbf{P}\{X \leq x\} \cdot \rho^{d-1}}{\mathbf{P}\{X > x\} + \mathbf{P}\{X \leq x\} \cdot \rho^{d-1}} \end{cases}$$

Here the probabilities are reweighted because the probability that a job of size $> x$ enters the *server* is not equal to the probability that a job of size $> x$ enters the *system*.

Performance

Figure 11 compares baseline RIQ and Redundancy-d to their SMALL variants with inherent size cutoffs at the 10th, 50th, and 90th percentiles of X. The RIQ+SMALL results are analytical; Redundancy-d+SMALL is simulated.

Surprisingly, replicating only the small jobs does not reduce mean response time under RIQ. In fact, as the size cutoff increases (i.e., more jobs are allowed to replicate) $\mathbf{E}[T]$ decreases, and is lowest under baseline RIQ. This is because RIQ is designed to replicate only when the system has spare capacity. Further limiting the number of jobs that replicate leads to a minor increase in the number of idle



(a) RIQ, $\lambda = 0.3$

(b) RIQ, $\lambda = 0.7$

(c) Redundancy-d, $\lambda = 0.3$

(d) Redundancy-d, $\lambda = 0.7$

Fig. 11. Comparing $\mathbf{E}[T]$ under baseline RIQ (top, from analysis) and Redundancy-d (bottom, simulated; 95% confidence intervals are within the line where not shown) to that under the SMALL variant, which replicates only jobs with inherent size below a certain cutoff X_i, where X_i represents the ith percentile of X. Here $S \sim \text{Dolly}(1, 12)$, $X \sim H2$ with $\mathbf{E}[X] = \frac{1}{4.7}$ and $C_X^2 = 10$, $Z = 0$, and $\lambda = 0.3$ (left) and $\lambda = 0.7$ (right).

servers, thereby helping small jobs that can still replicate. But this benefit is outweighed by the harm experienced by the large jobs that no longer get to see the minimum of multiple server slowdowns.

Similarly, when the system is stable under baseline Redundancy-d, lowering the size cutoff (i.e., replicating fewer jobs) leads to an increase in mean response time. But importantly, the system is stable at higher values of d under Redundancy-d+SMALL than under baseline Redundancy-d. While Redundancy-d+SMALL eventually becomes unstable (e.g., when $\lambda = 0.3$ and $C_X^2 = 10$, instability occurs around $d = 680$), at practical (low) values of d the system is both stable and relatively insensitive to the particular choice of d.

8.3 THRESHOLD: Replicate to Short Queues Only

A major advantage of RIQ is that the system cannot become unstable as d gets large. But RIQ is not perfect: with the right choice of d, Redundancy-d can achieve lower response time than RIQ. RIQ allows very few jobs to replicate, sacrificing potential response time gains to guarantee stability. On the other hand, Redundancy-d allows all jobs to replicate, offering the potential for large response time improvements, but risking instability. Ideally, our policy would lie between RIQ and Redundancy-d: we can afford to replicate more than under RIQ, but not as much as under Redundancy-d. To accomplish this, we introduce the THRESHOLD-n policy, a compromise between RIQ and Redundancy-d.

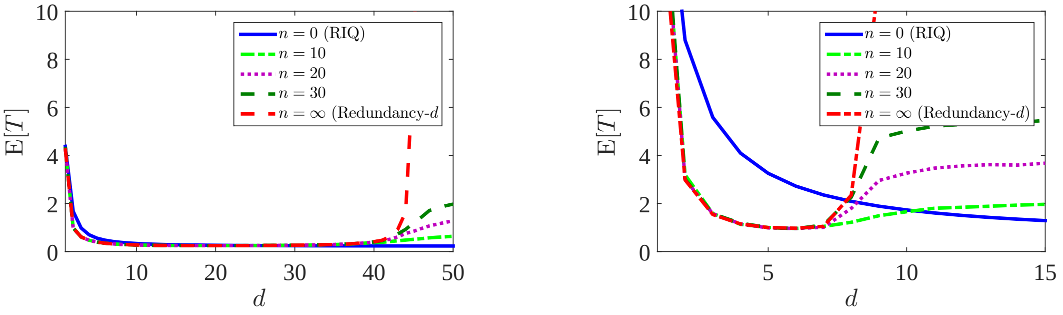
Definition 6. *Under* ***THRESHOLD-n****, each arriving job polls d servers and joins the queue at those servers with queue length $\leq n$. If all queue lengths are $> n$, the job joins a single queue at random from among the d polled servers.*

Note that RIQ and Redundancy-d are the extrema of the THRESHOLD-n policies, where RIQ≡THRESHOLD-0 and Redundancy-d≡THRESHOLD-∞.

Unfortunately, THRESHOLD-n is not analytically tractable. The key feature enabling us to analyze RIQ (and the JSQ and SMALL variants) is that under these policies, all copies of a job start service simultaneously, so we do not need to track the ages of copies in service. This is not true for THRESHOLD-n, hence we study THRESHOLD-n via simulation.

THRESHOLD-n achieves the best features of both RIQ and Redundancy-d (see Figure 12). Like Redundancy-d, THRESHOLD-n allows for enough redundancy to achieve substantial response time improvements. Like for RIQ, we can derive an upper bound on $\mathbf{E}[T]$ that shows that the system does not become unstable as a function of d (see Theorem 4). For example, when $\lambda = 0.7$ (Figure 12(b)), THRESHOLD-10 performs nearly identically to Redundancy-d for $d \leq 7$; here both policies outperform RIQ. At $\lambda = 8$, Redundancy-d approaches instability, but under THRESHOLD-10 response time plateaus to a value only slightly higher than that under RIQ. The plateau occurs because all of the queues tend to remain at exactly n, so all jobs have similar queueing times.

While we cannot analyze performance under THRESHOLD-n, we can prove that for all $n < \infty$ and for all d, the system is stable under THRESHOLD-n provided the system is stable when $d = 1$. This means that we can make n as high as needed to achieve the low response times offered by Redundancy-d at low d, without worrying about potential instability if we choose d too high.



(a) $\lambda = 0.3$

(b) $\lambda = 0.7$

Fig. 12. Mean response time under THRESHOLD-n for different values of n ($n = 0$ is from our RIQ analysis; other values of n are simulated, 95% confidence intervals are within the line). Here $X \sim H2$ with $\mathbf{E}[X] = \frac{1}{4.7}$ and $C_X^2 = 10$, $S \sim \text{Dolly}(1, 12)$, $Z = 0$, and (a) $\lambda = 0.3$ and (b) $\lambda = 0.7$.

Theorem 4. *The system is stable under THRESHOLD-n, for all n, if $\lambda \cdot \mathbf{E}[X \cdot S] < 1$.*

Proof. We follow a similar approach to the proof of stability under the original RIQ policy: we derive an upper bound on mean response time under THRESHOLD-n using an M/G/1/setup system. Details are in Appendix C. □

9 Cancel-on-Complete versus Cancel-on-Start

Thus far we have assumed a *Cancel-on-Complete* model, in which a job's extra copies are cancelled as soon as the first copy completes service. For completeness, we now turn to an alternative *Cancel-on-Start* model, in which a job's extra copies are cancelled as soon as the first copy *enters* service. Unlike under Cancel-on-Complete, under Cancel-on-Start no extra load is added to the system because only one copy of each job actually runs. Hence with Cancel-on-Complete there is no disadvantage to making d as high as possible. While in practice communication overhead may impose an upper limit on d, in theory it is best to set $d = k$, hence for the remainder of this section we assume $d = k$.

Cancel-on-Start maximizes the queueing benefits of redundancy: each job gets to experience the shortest possible queueing time across all queues. Importantly, this is *not* the same as the Least-Work-Left dispatching policy, under which each arriving job joins the queue with the least work, where "work" is the sum of the inherent sizes of all jobs in the queue, plus the remaining size of the job currently in service if there is one. Least-Work-Left cannot take into account the server slowdown each job ultimately experiences, because server slowdowns are not known in advance. Cancel-on-Start achieves what Least-Work-Left would achieve if both X and S were known in advance for each job (which is equivalent to a central queue); Cancel-on-Start does this without knowing X or S. In the special case in which $R(1) = X \cdot S$ is a mixture of exponentials, response time can be analyzed numerically using Matrix-Analytic methods. Figure 13(b) shows that when λ is high, the queueing benefit allows Cancel-on-Start to outperform both Redundancy-d and RIQ at all values of d.

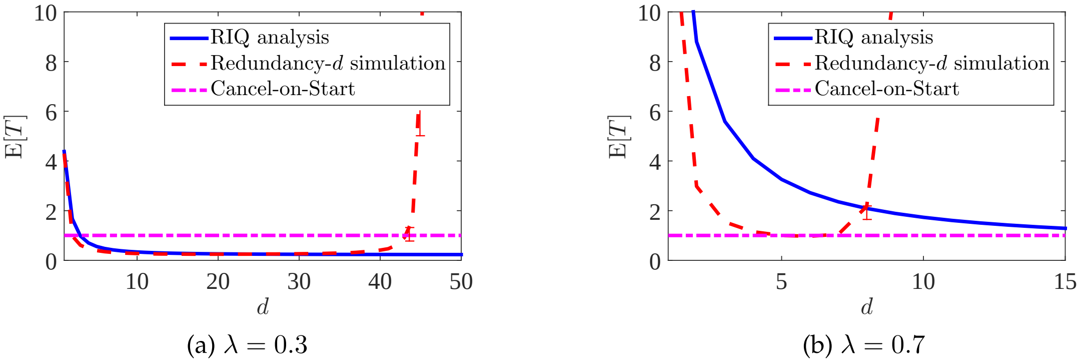
However, Cancel-on-Start does not allow jobs to experience the minimum slowdown across multiple servers. When λ is low, the system has extra capacity, which is wasted under Cancel-on-Start but used to reduce runtimes under Cancel-on-Complete. Hence at low λ Redundancy-d and RIQ both outperform Cancel-on-Start (see Figure 13(a)).

10 Conclusion

In this paper we introduce the $S\&X$ model, a new, more realistic model for computer systems with redundancy, where a job's inherent size X is decoupled from the server slowdown S. The model is very general, allowing for any inherent job size distribution X, any server slowdown distribution S, and any cancellation time Z. The $S\&X$ model is motivated by a common weakness of existing theoretical work on redundancy: the Independent Runtimes (IR) model, where a job's replicas experience independent runtimes across servers. The IR model leads to the conclusion that more redundancy always helps, which empirical systems work has shown is untrue.

In our new $S\&X$ model, dispatching policies previously studied in the theory literature, such as Redundancy-d, can perform much worse than predicted by prior theoretical analysis in the IR model. This motivates us to develop a new dispatching policy designed to perform well in the $S\&X$ model. We introduce the Redundant-to-Idle-Queue policy (RIQ), under which each arriving job creates redundant copies only when the job finds idle servers. We derive a highly accurate approximation for response time under RIQ. We also derive an upper bound on mean response time under RIQ that shows that RIQ does not cause instability even as the redundancy degree d becomes large. Our results demonstrate that RIQ is extremely *robust* to the system parameters, including the inherent job size distribution, the server slowdown distribution, and d.

The RIQ policy is one example of a redundancy-based policy that performs well in a realistic model. We analyze several variants to the baseline RIQ policy, such as RIQ+JSQ, under which jobs that find no idle servers use Join-the-Shortest-Queue dispatching, and RIQ+SMALL, under which only jobs with a small inherent size are allowed to create redundant copies. Our results indicate that because RIQ already significantly limits the number of jobs that are allowed to replicate, policy variations that further reduce the amount of redundancy do not help. Instead, we propose the THRESHOLD-n policy, which is a more liberal version



(a) $\lambda = 0.3$

(b) $\lambda = 0.7$

Fig. 13. Comparing Redundancy-d and RIQ under Cancel-on-Complete to Cancel-on-Start with $d = k = 1000$ at (a) $\lambda = 0.3$ and (b) $\lambda = 0.7$. Here $X \sim H2$ with $\mathbf{E}[X] = \frac{1}{4.7}$ and $C_X^2 = 10$, $S \sim \text{Dolly}(1, 12)$, and $Z = 0$.

of RIQ that allows jobs to replicate at any server with fewer than n jobs in the queue rather than just at idle servers. THRESHOLD-n combines the best features of RIQ and Redundancy-d: it achieves excellent performance by allowing many jobs to replicate, but avoids instability by ultimately limiting the amount of replication.

The $S\&X$ model represents an important first step in bridging the gap between theoretical models of redundancy and the practical characteristics of real systems that use redundancy. However, our model does not incorporate every aspect of such systems. For example, in practice server slowdowns may be time-dependent or correlated between consecutively processed jobs on the same server. We leave incorporating such features into a theoretical model of redundancy open for future work.

References

[1] Joseph Abate and Ward Whitt. Numerical inversion of laplace transforms of probability distributions. *ORSA Journal on computing*, 7(1):36–43, 1995.

[2] Reza Aghajani, Xingjie Li, and Kavita Ramanan. Mean-field dynamics of load-balancing networks with general service distributions. *arXiv preprint arXiv:1512.05056*, 2015.

[3] Ganesh Ananthanarayanan, Ali Ghodsi, Scott Shenker, and Ion Stoica. Effective straggler mitigation: Attack of the clones. In *NSDI 2013*.

[4] Ganesh Ananthanarayanan, Michael Chien-Chun Hung, Xiaoqi Ren, Ion Stoica, Adam Wierman, and Minlan Yu. Grass: Trimming stragglers in approximation analytics. In *Proceedings of the 11th USENIX Conference on Networked Systems Design and Implementation*, pages 289–302. USENIX Association, 2014.

[5] Ganesh Ananthanarayanan, Srikanth Kandula, Albert G Greenberg, Ion Stoica, Yi Lu, Bikas Saha, and Edward Harris. Reining in the outliers in Map-Reduce clusters using Mantri. In *OSDI 2010*.

[6] Thomas Bonald and Céline Comte. Networks of multi-server queues with parallel processing. *arXiv preprint arXiv:1604.06763*, April 2016.

[7] Jeffrey Dean and Luis Andre Barroso. The tail at scale. *Communications of the ACM*, 56(2):74–80, February 2013.

[8] SW Fuhrmann and Robert B Cooper. Stochastic decompositions in the M/G/1 queue with generalized vacations. *Operations research*, 33(5):1117–1129, 1985.

[9] Kristen Gardner, Mor Harchol-Balter, and Alan Scheller-Wolf. A better model for job redundancy: Decoupling server slowdown and job size. In *MASCOTS*, September 2016.

[10] Kristen Gardner, Mor Harchol-Balter, Alan Scheller-Wolf, Mark Velednitsky, and Samuel Zbarsky. Redundancy-d: The power of d choices for redundancy. *Operations Research*, 2016, To appear.

[11] Kristen Gardner, Samuel Zbarsky, Sherwin Doroudi, Mor Harchol-Balter, Esa Hyytiä, and Alan Scheller-Wolf. Reducing latency via redundant requests: Exact analysis. In *SIGMETRICS*, June 2015.

[12] Longbo Huang, Sameer Pawar, Hao Zhang, and Kannan Ramchandran. Codes can reduce queueing delay in data centers. In *Information Theory Proceedings (ISIT), 2012 IEEE International Symposium on*, pages 2766–2770. IEEE, 2012.

[13] Gauri Joshi, Yanpei Liu, and Emina Soljanin. Coding for fast content download. In *Allerton Conference'12*, pages 326–333, 2012.

[14] Gauri Joshi, Yanpei Liu, and Emina Soljanin. On the delay-storage trade-off in content download from coded distributed storage systems. *IEEE J. Sel. Area. Comm.*, 32(5):989–997, May 2014.

[15] Ger Koole and Rhonda Righter. Resource allocation in grid computing. *Journal of Scheduling*, 11:163–173, 2009.

[16] Akshay Kumar, Ravi Tandon, and T Charles Clancy. On the latency of heterogeneous MDS queue. In *Global Communications Conference (GLOBECOM), 2014 IEEE*, pages 2375–2380. IEEE, 2014.

[17] Yi Lu, Qiaomin Xie, Gabriel Kliot, Alan Geller, James R Larus, and Albert Greenberg. Join-idle-queue: A novel load balancing algorithm for dynamically scalable web services. *Performance Evaluation*, 68(11):1056–1071, 2011.

[18] Kay Ousterhout, Patrick Wendell, Matei Zaharia, and Ion Stoica. Sparrow: distributed, low latency scheduling. In *Proceedings of the 24th ACM Symposium on Operating Systems Principles*, pages 69–84, 2013.

[19] Xiaoqi Ren, Ganesh Ananthanarayanan, Adam Wierman, and Minlan Yu. Hopper: Decentralized speculation-aware cluster scheduling at scale. In *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication*, pages 379–392. ACM, 2015.

[20] Nihar B. Shah, Kangwook Lee, and Kannan Ramchandran. The MDS queue: Analysing latency performance of codes and redundant requests. Technical Report arXiv:1211.5405, November 2012.

[21] Nihar B. Shah, Kangwook Lee, and Kannan Ramchandran. When do redundant requests reduce latency? Technical Report arXiv:1311.2851, June 2013.

[22] Yin Sun, Can Emre Koksal, and Ness B. Shroff. On delay-optimal scheduling in queueing systems with replications. *CoRR*, abs/1603.07322, 2016.

[23] Ashish Vulimiri, P. Brighten Godfrey, Radhika Mittal, Justine Sherry, Sylvia Ratnasamy, and Scott Shenker. Low latency via redundancy. In *CoNEXT*, pages 283–294, December 2013.

[24] Da Wang, Gauri Joshi, and Gregory Wornell. Efficient task replication for fast response times in parallel computation. Technical Report arXiv:1404.1328, April 2014.

[25] Peter D Welch. On a generalized M/G/1 queuing process in which the first customer of each busy period receives exceptional service. *Operations Research*, 12(5):736–752, 1964.

[26] Yunjing Xu, Michael Bailey, Brian Noble, and Farnam Jahanian. Small is better: Avoiding latency traps in virtualized data centers. In *Proceedings of the 4th annual Symposium on Cloud Computing*, page 7. ACM, 2013.

[27] Matei Zaharia, Andy Konwinski, Anthony D Joseph, Randy H Katz, and Ion Stoica. Improving MapReduce performance in heterogeneous environments. In *OSDI*, volume 8, page 7, 2008.

Appendix A Definitions needed for Theorem 1

In this section we define the M/G/1/vacation system, which we use to upper bound mean response time under RIQ in our system. At a high level, in an M/G/1/vacation system, the server becomes unavailable ("goes on vacation") for a period of time every time it becomes idle. When the server "comes back" from its vacation, it begins serving any jobs that have arrived during the vacation time, or takes another vacation if there are no such jobs. The M/G/1/vacation system has been studied extensively in the literature, see, e.g., [8] for an overview.

Definition 7 (Paraphrased from [8])**.** *An M/G/1/vacation system is a single-server queueing system satisfying the following properties:*

1) *Jobs arrive to the system as a Poisson process with rate λ.*
2) *Each job has runtime R; runtimes are i.i.d. across jobs and are independent of the arrival process and the sequence of server vacations.*
3) *The system is stable. That is, $\rho = \lambda \mathbf{E}[R] < 1$.*
4) *The scheduling policy (i.e., the order in which jobs are served) does not depend on the job runtimes.*
5) *Services are non-preemptible.*
6) *When the queue becomes empty, the server immediately becomes unavailable for time V. If there are jobs waiting in the queue when the server returns from vacation, it immediately begins serving these jobs and continues to do so until it next becomes idle. If instead the queue is empty when the server returns from vacation, it immediately begins another vacation of duration V.*

The Laplace transform of response time in the M/G/1/vacation system, $\widetilde{T}^{\mathrm{M/G/1/vacation}}(s)$, has been shown to satisfy a decomposition property [8]:

$$\widetilde{T}^{\mathrm{M/G/1/vacation}}(s) = \widetilde{T}^{\mathrm{M/G/1}}(s) \cdot \widetilde{V_e}(s),$$

where V_e is the excess of the vacation time V, defined below.

Definition 8. *Consider a renewal process with inter-event times V. Suppose that we look at the process at a random time. The excess of V is defined to be the remaining time until the next event occurs. The Laplace transform of the excess of V is*

$$\widetilde{V_e}(s) = \frac{1 - \widetilde{V}(s)}{s\mathbf{E}[V]}$$

and the mean excess of V is

$$\mathbf{E}[V_e] = \frac{\mathbf{E}[V^2]}{2\mathbf{E}[V]}.$$

Appendix B Proof of Theorem 3

Theorem 5. *Under RIQ+SMALL, for all d the system is stable if $\lambda \cdot \mathbf{E}[X \cdot S] < 1$.*

Proof. As for the baseline RIQ policy, we will upper bound mean response time under RIQ+SMALL by the mean response time in an M/G/1/vacation system. We first express mean response time under RIQ+SMALL by conditioning on a job's inherent size. We have

$$\begin{aligned}\mathbf{E}[T]^{\mathrm{RIQ+SMALL}} &= \mathbf{P}\{X \le x\} \cdot \mathbf{E}[T|X \le x] \\ &\quad + \mathbf{P}\{X > x\} \cdot \mathbf{E}[T|X > x].\end{aligned}$$

When a small job (with size $X \le x$) arrives to the system, if it finds i idle servers it enters service on all of these servers. For all i, we have

$$\begin{aligned}&\mathbf{E}[T|\text{job finds } i \text{ idle servers} \;\&\; X \le x] = \mathbf{E}[R(i)|X \le x] \\ &\quad \le \mathbf{E}[R(1)|X \le x] \\ &\quad \le \mathbf{E}[R(1)|X \le x] + \mathbf{E}\left[T^Q|\text{job finds no idle servers}\right] \\ &\quad = \mathbf{E}[T|\text{job finds no idle servers} \;\&\; X \le x]. \qquad (15)\end{aligned}$$

When a small job arrives to the system and does not find any idle servers, it joins a single queue chosen at random and experiences response time

$$\mathbf{E}[T|\text{job finds no idle servers} \;\&\; X \le x]. \qquad (16)$$

When a large job (with size $X > x$) arrives to the system, the job joins a single queue chosen uniformly at random. The job experiences response time

$$\begin{aligned}&\mathbf{E}[T|X > x] \\ &\quad = \mathbf{E}\left[T \,\middle|\, \begin{matrix} X > x \;\&\; \text{job} \\ \text{finds idle servers}\end{matrix}\right] \cdot \mathbf{P}\left\{\begin{matrix} X > x \;\&\; \text{job} \\ \text{finds idle servers}\end{matrix}\right\} \\ &\qquad + \mathbf{E}\left[T \,\middle|\, \begin{matrix} X > x \;\&\; \text{job finds} \\ \text{no idle servers}\end{matrix}\right] \cdot \mathbf{P}\left\{\begin{matrix} X > x \;\&\; \text{job finds} \\ \text{no idle servers}\end{matrix}\right\} \\ &\quad \le \mathbf{E}[T|\text{job finds no idle servers} \;\&\; X > x]. \qquad (17)\end{aligned}$$

For both small and large jobs that find no idle servers, the job joins the queue at a randomly chosen server with the following properties:

1) When the server is busy, small jobs arrive with rate

$$\begin{aligned}&k\lambda \mathbf{P}\{X \le x\} \cdot \frac{d}{k} \cdot \mathbf{P}\left\{\begin{matrix}\text{job finds } d-1 \\ \text{other servers busy}\end{matrix}\right\} \cdot \frac{1}{d} \\ &\qquad = \lambda \mathbf{P}\{X \le x\} \mathbf{P}\left\{\begin{matrix}\text{job finds } d-1 \\ \text{other servers busy}\end{matrix}\right\}\end{aligned}$$

and large jobs arrive with rate

$$k\lambda \mathbf{P}\{X > x\} \cdot \frac{1}{k} = \lambda \mathbf{P}\{X > x\}.$$

2) Small jobs that arrive to the server while it is busy experience runtime $R(1|X \le x)$, and large jobs that arrive to the server while it is busy experience runtime $R(1|X > x)$.
3) A small job that arrives to the server while it is idle, and finds $i-1$ other idle servers, experiences runtime $R(i|X \le x) \le_{st} R(1|X \le x) + Z$. A large job that arrives to the server while it is idle experiences runtime $R(1|X > x)$.

Since (15), (16), and (17) force all jobs to have a non-zero queueing time in our upper bound, the first job in the busy period (the job that arrives to the server while it is idle) can be viewed as a "dummy" job that does not contribute to response time. Every time a server goes idle, we can imagine that the server is forced to work on a "dummy" job, so the server is always busy when the next real job arrives. This dummy job has size

$$\begin{aligned} R_0 &= \begin{cases} R(1|X > x) & \text{w.p. } \frac{\mathbf{P}\{X>x\}}{\mathbf{P}\{X>x\}+d\cdot\mathbf{P}\{X\leq x\}} \\ R(i) + Z|X \leq x & \text{w.p. } \frac{d\cdot\mathbf{P}\{X\leq x\}}{\mathbf{P}\{X>x\}+d\cdot\mathbf{P}\{X\leq x\}} \\ & \cdot\mathbf{P}\left\{\begin{matrix}\text{job finds } i-1 \\ \text{other servers idle}\end{matrix}\right\} \end{cases} \\ &\leq_{st} \begin{cases} R(1|X > x) & \text{w.p. } \frac{\mathbf{P}\{X>x\}}{\mathbf{P}\{X>x\}+d\cdot\mathbf{P}\{X\leq x\}} \\ R(1) + Z|X \leq x & \text{w.p. } \frac{d\cdot\mathbf{P}\{X\leq x\}}{\mathbf{P}\{X>x\}+d\cdot\mathbf{P}\{X\leq x\}} \end{cases} \\ &\leq_{st} R(1) + Z \end{aligned}$$

The system that runs a "dummy" job of size $R(1) + Z$ every time it becomes idle gives an upper bound on mean response time in our original system, and is exactly an M/G/1/vacation system with vacation time $R(1) + Z$ and arrival rates and runtimes as stated in items 1 and 2 above. We can further upper bound the M/G/1/vacation system by simply increasing the arrival rate of small jobs to $\lambda\mathbf{P}\{X \leq x\}$. We now have an M/G/1/vacation system with arrival rate λ, runtime $R(1)$, and vacation time $R(1) + Z$. This system is stable provided that $\lambda \cdot \mathbf{E}[R(1)] = \lambda \cdot \mathbf{E}[X \cdot S] < 1$. Since mean response time in the M/G/1/vacation system upper bounds mean response time in our original system under RIQ+SMALL, we also have that our system is stable under RIQ+SMALL, for all d, if $\lambda \cdot \mathbf{E}[X \cdot S] < 1$. □

Appendix C
Proof of Theorem 4

Theorem 6. *Under THRESHOLD-n, for all d and n the system is stable if $\lambda \cdot \mathbf{E}[X \cdot S] < 1$.*

Proof. We will upper bound mean response time under THRESHOLD-n by the mean response time in an M/G/1/setup system. We begin by expressing mean response time under THRESHOLD-n by conditioning on whether a tagged arrival to the system finds any idle servers. We have

$$\begin{aligned} &\mathbf{E}[T]^{\text{THRESH}-n} \\ &\quad = \mathbf{P}\left\{\begin{matrix}\text{job finds servers} \\ \text{with } \leq n \text{ jobs}\end{matrix}\right\} \cdot \mathbf{E}\left[T \,\middle|\, \begin{matrix}\text{job finds servers} \\ \text{with } \leq n \text{ jobs}\end{matrix}\right] \\ &\quad\quad + \mathbf{P}\left\{\begin{matrix}\text{job finds no servers} \\ \text{with } \leq n \text{ jobs}\end{matrix}\right\} \cdot \mathbf{E}\left[T \,\middle|\, \begin{matrix}\text{job finds no servers} \\ \text{with } \leq n \text{ jobs}\end{matrix}\right]. \end{aligned}$$

When a job arrives to the system, if it finds i servers with fewer than n jobs it joins the queue (or enters service if the queue is empty) at all of these servers. For all i, we have

$$\mathbf{E}\left[T \,\middle|\, \begin{matrix}\text{job finds } i \text{ servers} \\ \text{with } \leq n \text{ jobs}\end{matrix}\right] \leq \mathbf{E}\left[T \,\middle|\, \begin{matrix}\text{job finds no servers} \\ \text{with } \leq n \text{ jobs}\end{matrix}\right].$$

Hence we have the following upper bound:

$$\mathbf{E}[T]^{\text{THRESH}-n} \leq \mathbf{E}\left[T \,\middle|\, \begin{matrix}\text{job finds no servers} \\ \text{with } \leq n \text{ jobs}\end{matrix}\right]. \tag{18}$$

Given that a tagged arrival found no servers with $\leq n$ jobs, it joins the queue at a randomly chosen server with the following properties:

1) When the server is busy, jobs arrive with rate $k\lambda \cdot \frac{d}{k} \cdot \mathbf{P}\{\text{job finds other } d-1 \text{ servers with } > n \text{ jobs}\} \cdot \frac{1}{d} \leq \lambda$.
2) All jobs that arrive to the server while it is busy experience runtime $R(1)$.
3) A tagged job that arrives to the server while it has $\leq n$ jobs, and finds $i-1$ other servers with $\leq n$ jobs will complete when the first of its i copies completes service.

Since (18) forces all jobs to wait behind at least n jobs in our upper bound, we can view any job that arrives while there are $\leq n$ jobs at the server to be a "dummy" job that does not contribute to response time. Every time the queue length drops below n jobs, we can imagine that a "dummy" job with size $R(1) + Z$ is added to the queue, so the server always has greater than n jobs in the queue when the next "real" job arrives. Alternatively (but equivalently), we can imagine that a "real" arrival that finds no other "real" jobs at the server triggers a setup time of length $(R(1) + Z)_e + \sum_{j=1}^{n-1}(R(1) + Z)$ (i.e., the time remaining for the "dummy" job currently in service, plus $n-1$ additional "dummy" jobs in queue; $(R(1)+Z)_e$ is the excess of $R(1)+Z$). This exactly describes an M/G/1/setup system with arrival rate λ, runtime $R(1)$, and setup time $(R(1)+Z)_e+\sum_{j=1}^{n-1}(R(1)+Z)$. The M/G/1/setup is stable if $\lambda\cdot\mathbf{E}[R(1)] = \lambda\cdot\mathbf{E}[X \cdot S] < 1$. Since the mean response time in the M/G/1/setup upper bounds the mean response time in our original system under THRESHOLD-n, the system is stable under THRESHOLD-n if $\lambda \cdot \mathbf{E}[X \cdot S] < 1$. □