2016 IEEE 24th International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems

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

Kristen Gardner
Computer Science Department
Carnegie Mellon University
ksgardne@cs.cmu.edu

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

Alan Scheller-Wolf
Tepper School of Business
Carnegie Mellon University
awolf@andrew.cmu.edu

***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 copy of the request completes. Redundancy is beneficial because it allows us to overcome server-side variability – the fact that the server we choose might be temporarily slow due to factors such as background load, network interrupts, and garbage collection. When there is significant server-side variability, replicating requests can greatly reduce response times.**

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 allows us to decouple 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 policy, Redundant-to-Idle-Queue (RIQ) which is both analytically tractable within the $S\&X$ model and has provably excellent performance.

I. 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 $12\times$ longer on one machine than another [1], or even $27\times$ longer [23]. 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 (see for example [1], [2], [5], [15], [16], [20]). 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*.[1] Redundancy provides two key advantages: First, a job that is dispatched to d servers gets to experience the queue with the least work. This is true even if jobs are immediately dispatched to servers via a front-end load balancer (no central queue), where this front-end load balancer does not have any knowledge of the number of jobs in the queues, or their sizes. Second, under redundancy, each job experiences the minimum slowdown of the servers on which it runs. Both of these 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 (see [4], [7]–[13], [17], [18], [20]). All of these results assume an *Independent Runtimes (IR) model*, where the copies of a single job have independent runtimes (service times) at different servers. These independent runtimes are often exponentially distributed, but sometimes are assumed to have higher variability. In both these cases, the IR model leads to the conclusion that (barring cancellation costs), "more redundancy is always 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, in the sense that it comprises a large volume of computation. That job should appear to be a large job 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

*This work was supported by NSF-CMMI-1538204, NSF-CMMI-1334194, and NSF-CSR-1116282, by the Intel ISTC-CC, and by a Google Faculty Research Award 2015/16.

[1]There are many versions of redundancy. For instance, a "job" might be composed of many tasks. Some tasks might be replicated while others are not, the decision to replicate might be made after the initial dispatch, or copies may be cancelled when the first copy enters service. For the purposes of this paper, we stick to the model where a job is an atomic unit, the decision to replicate a job or not is made at the moment that the job is dispatched, and extra copies are cancelled as soon as the first copy completes service.

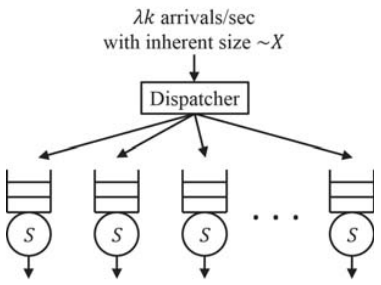


Fig. 1. The $S\&X$ model. The system has k servers and jobs arrive as a Poisson process with rate λk. Each job has an inherent size X. When a job runs on a server it experiences slowdown S. A job's running time on a single server is $R(1) = X \cdot S$. When a job runs on multiple servers, its inherent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

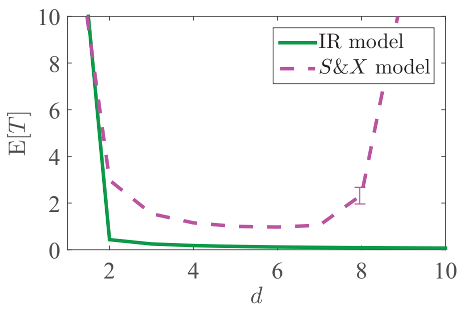


Fig. 2. Under the Redundancy-d policy, each arriving job sends copies to d servers chosen uniformly at random. Here we show mean response time, $\mathbf{E}[T]$, as a function of d under Redundancy-d when the system has $k = 1000$ servers, the total arrival rate is λk where $\lambda = 0.7$, and inherent job sizes are $X \sim H2$ with $C_X^2 = 10$. In the IR model (solid green line, from analysis in [8]), 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 III. 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.

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, a policy like Redundancy-d that replicates every arriving job to d queues [8]. Under the IR model assumed in [8], mean response time only decreases as we increase d. By contrast, in the $S\&X$ model, we show that the mean response time under Redundancy-d can improve as we increase d for a while, but the system (typically) eventually becomes unstable because of the increased load of replication, sending mean response time to infinity (see Figure 2). Unfortunately, we cannot in general determine

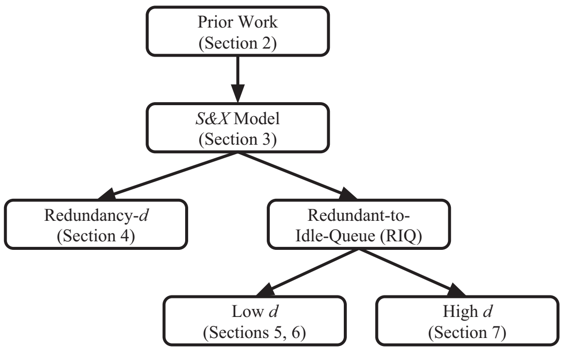


Fig. 3. Outline of the remainder of this paper.

which values of d will cause problems 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 [8]).

The deficiencies 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 RIQ's performance within the $S\&X$ model, including mean response time and the transform of response time, 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. 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. Thus RIQ represents a provably robust and analytically understood replication policy, which is also simple enough to be appealing to practitioners. The RIQ policy demonstrates that it is possible to design good redundancy policies within the $S\&X$ model.

The remainder of this paper is outlined in Figure 3.

II. Prior Work: The Gap Between Theory and Systems

In the past several years there has been a growing interest in the theoretical community in analyzing systems with redundancy, in which a job requires one or more copies to complete service, 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, introduced in [8], every arriving job is replicated to d servers chosen at random, where d is a small constant compared to the number of servers, k. The authors analyze the full distribution of response time as a function of d in the IR model [8]; as the number of copies per job increases, mean response time decreases. In [7], a classed-based replication system is considered, where a job's class determines where it is replicated. Here, exact response time distributions are derived in the IR model, assuming exponential runtimes. In [4], the authors extend the results in [7] to networks of queues, still in the IR model. In [19], a different model is considered where jobs are composed of tasks, where tasks may be replicated; again the analysis assumes the IR model. In all these works, when runtimes are exponential or more variable, it is optimal to replicate jobs at all servers. This is also in accordance with [12], [18].

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 [10], [11], [17], [21] in the IR model. While [10] 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 M/G/∞). 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 [11], [17].

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 [20]. 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, more redundancy can lead to overload rather than always improving performance. 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 [1]. 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 [3], [24]. More recently, [2], [16] build 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).

Our goal in this paper is to bring the theoretical models of redundancy systems closer to the real systems that the 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 the redundancy policies used in real computer systems in Section VIII, where we discuss these policies in the context of the $S\&X$ model.

III. The $S\&X$ Model

We consider a setting with k homogeneous servers (see Figure 1). Each server has its own queue and works on the jobs in its queue in first-come first-served order. Jobs arrive to the system 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*. When a job runs on one server, its *runtime* is $R(1) = X \cdot S$ (S is assumed to be ≥ 1 and can be thought of as the factor by which the inherent size, X, is "stretched")[2]. Unlike in the IR model, in the $S\&X$ model runtimes are *not* independent across servers, due to the fact that a job's X component is the same on all servers.

We assume that every time a server begins working on a job, it draws a new instance of S. Thus consecutive jobs running on the same server may see different slowdowns. This reflects the fact that the server slowdown changes over time (due to factors such as garbage collection, background load, network interference, etc.) and is not fixed for a particular server.

A job that joins the queue at a particular server j will complete service on that server after time equal to the queueing time at that server, T_j^Q, plus the job's running time on that server, $X \cdot S_j$, where S_j is the particular instance of S that the job experiences on server j (S_j is drawn independently across servers for a particular job, and across jobs for a particular server). If a job joins the queue at multiple servers, its response time is the minimum across all its servers:

$$T = \min_j \{T_j^Q + X \cdot S_j\}.$$

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.

In much of the remainder of this paper, we focus on one particular S distribution, called the Dolly(1,12) distribution (see Table I), which was measured empirically in [1] by analyzing traces collected from Facebook's Hadoop cluster and Microsoft Bing's Dryad cluster.[3] Slowdown values range from 1 to 12, with mean 4.7 and variance 9.5.

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

[3]The full distribution does not appear in [1]; we obtained this from personal communication with the authors.

TABLE I
THE DOLLY(1,12) EMPIRICAL SLOWDOWN DISTRIBUTION [1]. 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

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 IV and V, which introduce specific dispatching policies: Redundancy-d and Redundant-to-Idle-Queue.

IV. THE REDUNDANCY-d POLICY

A natural dispatching policy for systems with redundancy is *Redundancy-d* [8]. Under Redundancy-d, each job that arrives to the system creates d copies of itself and sends these copies to d different servers chosen uniformly at random without replacement. When a job's first copy completes service, the job is complete and all remaining copies are cancelled.

Response time under the Redundancy-d dispatching policy has been analyzed in the IR model [8]. Figure 2 compares mean response time under Redundancy-d in the IR model (from analysis, see [8]) 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 mean response time, $\mathbf{E}[T]$, decreases assuming 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 Redundancy-d can become unstable 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, jobs 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. The $S\&X$ model results tell a very different story from the IR model results. 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 is oblivious to the system state. It creates copies of jobs even when the system has no extra capacity with which to run these copies. Consequently, Redundancy-d is prone to adding 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.

V. REDUNDANT-TO-IDLE-QUEUE

In this section we introduce a new dispatching policy, Redundant-to-Idle-Queue (RIQ).[4] The RIQ policy is described in Section V-A and is analyzed in the remaining subsections. We begin by approximately analyzing mean response time (Section V-B) under RIQ. We then build on this analysis to derive an approximation for the transform of response time under RIQ; in a few special cases, we are able to approximate the full distribution of response time (Section V-C). In Section V-D we extend our analysis to allow for cancellation times. In Section V-E we evaluate the quality of our approximations by comparing our analytical results to simulation.

A. The Redundant-to-Idle-Queue Dispatching Policy

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 $0 < i \leq d$ 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 address this at the end of Section V-B.

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 V-E we show that while Assumption 1 does not hold in general, our analysis matches simulation when $d \ll k$.

B. Analysis: Mean Response Time

Our goal in this section is to derive mean response time, $\mathbf{E}[T]$, under RIQ. We use Assumption 1 throughout our analysis.

[4]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 [14].

We begin by conditioning on whether an arrival to the system finds any idle servers:

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

where the second line is due to Assumption 1. We defer our derivation of ρ to the end of the section.

We first find $\mathbf{E}\left[T \mid \text{job finds idle servers}\right]$ by conditioning on the number of idle servers a job finds, given that it finds at least one idle server:

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

To find $\mathbf{E}\left[T \mid \text{job finds no idle servers}\right]$, we observe that when a job finds no idle servers it joins the queue at a single server. That server 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 every job that polls the server while it is busy runs on it if all $d-1$ other servers that the job polls are also busy (probability ρ^{d-1}) and then the job chooses our particular server (probability $\frac{1}{d}$).
3) All jobs that find the server idle have runtime R_0, where

$$R_0 = R(i) \quad \text{w.p. } (1-\rho)^{i-1}\rho^{d-i}\binom{d-1}{i-1}, \quad 1 \le i \le d. \quad (3)$$

   Here we use the probability that the arrival found $i-1$ *other* idle servers among its $d-1$ other servers.
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.)

To find $\mathbf{E}\left[T \mid \text{job finds no idle servers}\right]$, we first observe:

$$\begin{aligned}\mathbf{E}\left[T \,\middle|\, \begin{matrix}\text{job finds no}\\ \text{idle servers}\end{matrix}\right] &= \mathbf{E}\left[R(1)\right] \\ &\quad + \mathbf{E}\left[T_Q \mid \text{queueing}\right]^{\text{M}^*/\text{G}/1/\text{efs}} \quad (4)\end{aligned}$$

$$\mathbf{E}\left[T_Q \mid \text{queueing}\right]^{\text{M}^*/\text{G}/1/\text{efs}} = \frac{\mathbf{E}\left[T_Q\right]^{\text{M}^*/\text{G}/1/\text{efs}}}{P_Q^{\text{M}^*/\text{G}/1/\text{efs}}}, \quad (5)$$

where

$$P_Q^{\text{M}^*/\text{G}/1/\text{efs}} = \frac{\mathbf{E}\left[N_B\right]}{\mathbf{E}\left[N_B\right]+1} \quad (6)$$

is the probability that an arrival has to wait in the queue in an M*/G/1/efs and

$$\mathbf{E}\left[N_B\right] = \lambda' \cdot \frac{\mathbf{E}\left[R_0\right]}{1-\lambda'\mathbf{E}\left[R(1)\right]} \quad (7)$$

is the expected number of arrivals during an M*/G/1/efs busy period, and $E[T_Q]^{\text{M}^*/\text{G}/1/\text{efs}}$ is given in Lemma 1.

Lemma 1. *The mean queueing time 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 while the system is busy is λ', is*

$$\begin{aligned}\mathbf{E}\left[T_Q\right]^{\text{M}^*/\text{G}/1/\text{efs}} &= \mathbf{E}\left[T_Q\right]^{\text{M}/\text{G}/1/\text{efs}} \\ &= \mathbf{E}\left[T\right]^{\text{M}/\text{G}/1/\text{efs}} - \mathbf{E}\left[S\right]^{\text{M}^*/\text{G}/1/\text{efs}}, \quad (8)\end{aligned}$$

where

$$\mathbf{E}\left[T\right]^{\text{M}/\text{G}/1/\text{efs}} = \frac{(1-\lambda'\mathbf{E}\left[R(1)\right])(2\mathbf{E}\left[R_0\right]+\lambda'\mathbf{E}\left[R_0^2\right]) + \lambda'^2\mathbf{E}\left[R_0\right]\mathbf{E}\left[R(1)^2\right]}{2(1-\lambda'\mathbf{E}\left[R(1)\right]+\lambda'\mathbf{E}\left[R_0\right])(1-\lambda'\mathbf{E}\left[R(1)\right])}$$

is derived in [22] and

$$\mathbf{E}\left[S\right]^{\text{M}^*/\text{G}/1/\text{efs}} = \frac{1}{\mathbf{E}\left[N_B\right]+1}\cdot\mathbf{E}\left[R_0\right] + \frac{\mathbf{E}\left[N_B\right]}{\mathbf{E}\left[N_B\right]+1}\cdot\mathbf{E}\left[R(1)\right],$$

where $\mathbf{E}\left[N_B\right]$ is given in (7).

Proof. The analysis used to find mean response time in an M/G/1/efs does not directly apply to the M*/G/1/efs because of our state-dependent arrival rate: the derivation of the transform of response time in an M/G/1/efs relies on PASTA, which does not apply in the M*/G/1/efs. However, from a *job's* perspective, the arrival rate while the system is idle does not affect response time. The arrival rate while the system is idle affects how close together busy periods occur, but does not affect any of the jobs during the busy period. Hence to derive response time, we can view the system as being an M/G/1/efs, the response time in which is derived in [22]. □

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

Our last step is to derive ρ, the probability that a server is busy. In Section III we defined $\rho = \lambda \cdot \mathbf{E}\left[I \cdot R(I)\right]$, where I is the number of servers on which a job runs. Using Assumption 1, we can write

$$\rho = \lambda\left(\sum_{i=1}^{d} i\cdot(1-\rho)^i\rho^{d-i}\binom{d}{i}\mathbf{E}\left[R(i)\right] + \rho^d\cdot\mathbf{E}\left[R(1)\right]\right). \quad (9)$$

Alternatively, we can compute ρ using renewal-reward theory, defining a cycle as the moment from when a server becomes idle until it is next idle. We have

$$\rho = \frac{\mathbf{E}\left[B\right]}{\mathbf{E}\left[B\right]+\frac{1}{\lambda d}}, \quad (10)$$

where $\mathbf{E}[B] = \frac{\mathbf{E}[R_0]}{1-\lambda'\mathbf{E}[R(1)]}$ is the expected duration of a busy period and $\frac{1}{\lambda d}$ is the mean interarrival time in the M*/G/1/efs when the server is idle.

Noting that $\mathbf{E}[R_0]$ is defined in terms of ρ, equation (10) gives us an expression for ρ in terms of itself. It is easy to verify algebraically that equations (9) and (10) are equivalent. When $d = 2$, the equations are of degree 2 and can be solved exactly; for higher d we can 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 VII we derive an upper bound on mean response time under RIQ which gives us an alternative condition: the system is stable if $\lambda \cdot \mathbf{E}[R(1)] < 1$.

C. Analysis: Transform of Response Time

Our approach in Section V-B also allows us to find the transform of response time. As before, we begin by conditioning on whether an arrival to the system finds any idle servers:

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

We first find $\widetilde{T}(s \mid \text{job finds idle servers})$ by conditioning on the number of idle servers a job finds, given that it finds at least one idle server:

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

Next we need to find $\widetilde{T}(s \mid \text{job finds no idle server})$. We do this using the M*/G/1/efs defined in Section V-B:

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

$$\widetilde{T_Q}(s \mid \text{queueing})^{\text{M}^*/\text{G}/1/\text{efs}} = \frac{\widetilde{T}(s)^{\text{M}^*/\text{G}/1/\text{efs}} - (1 - P_Q^{\text{M}^*/\text{G}/1/\text{efs}}) \cdot \widetilde{R_0}(s)}{P_Q^{\text{M}^*/\text{G}/1/\text{efs}} \cdot \widetilde{R(1)}(s)},$$

where $P_Q^{\text{M}^*/\text{G}/1/\text{efs}}$ is given in (6) and $\widetilde{T}(s)^{\text{M}^*/\text{G}/1/\text{efs}}$ is given in Lemma 2.

Lemma 2. *The 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 $R(1)$, and the arrival rate while the server is busy is λ' is*

$$\widetilde{T}(s)^{\text{M}^*/\text{G}/1/\text{efs}} = \widetilde{T}(s)^{\text{M}/\text{G}/1/\text{efs}} \overset{[22]}{=} \frac{1-\lambda'\mathbf{E}[R(1)]}{1-\lambda'\mathbf{E}[R(1)]+\lambda'\mathbf{E}[R_0]} \cdot \frac{(\lambda'-s)\widetilde{R_0}(s) - \lambda'\widetilde{R(1)}(s)}{\lambda' - s - \lambda'\widetilde{R(1)}(s)}.$$

Proof. The equivalence between response time in the M*/G/1/efs and in the M/G/1/efs follows the same reasoning as in the proof of Lemma 1. □

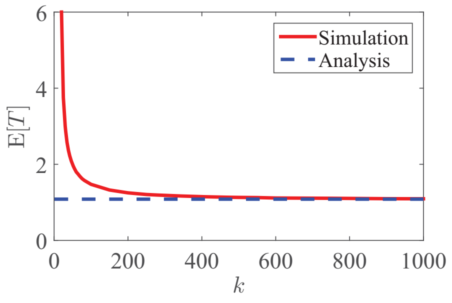


Fig. 4. Mean response time under RIQ from analysis (dashed blue line) and simulation (solid red line, 99% confidence intervals are within the line) when $\lambda = 0.7$, $d = 20$, $X \sim$ H2 with $\mathbf{E}[X] = \frac{1}{4.7}$ and $C_X^2 = 10$, and $S \sim$ Dolly(1, 12). When $k = 1000$, the analysis is within 1% of simulation.

D. Cancellation Costs

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 deterministic time Z. In practice, if a job is running on i servers then only $i-1$ of these servers need to incur a cancellation cost. To simplify the analysis slightly, we assume that all i servers incur the cancellation cost; this slightly increases our approximation for mean response time.

To include a cancellation time, we redefine R_0 from (3):

$$R_0 = \begin{cases} R(1) & \text{w.p. } \rho^{d-1} \\ R(i) + Z & \text{w.p. } (1-\rho)^{i-1}\rho^{d-1}\binom{d-1}{i-1}, \quad 1 < i \le d. \end{cases}$$

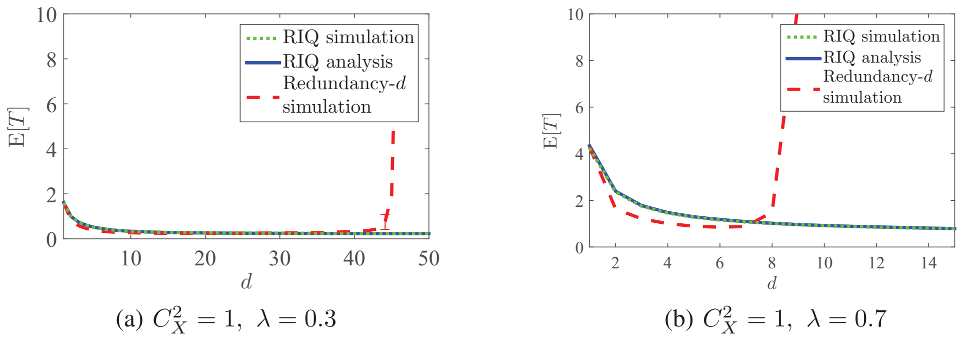
The rest of the analysis for both the mean and the transform of response time remains the same as above. It is easy to make Z a non-deterministic random variable if desired.

E. Quality of Approximation

Our analysis relies on the assumption that the servers are idle independently. This is not true in general. Nonetheless, by comparing our analytical approximation to simulation results, we observe that our approximation matches simulation quite well provided d is sufficiently small relative to k. Figure 4 compares mean response time obtained from our analysis to that obtained via simulation in a system where $d = 20$, $\lambda = 0.7$, inherent job sizes are highly variable, and S follows the Dolly(1,12) server slowdown distribution. As k increases, our analysis becomes more accurate; when $k = 1000$ the analysis is within 1% of simulation. While we only show results for one set of parameters, when d and λ are lower the simulation results converge to the analytical results even more quickly.

VI. Results: $d \ll k$

Our goal in this section is to evaluate the performance of RIQ using our analysis from Section V. Throughout the section we set $\mathbf{E}[R(1)] = 1$ and $k = 1000$ servers. Throughout this section, S follows the Dolly(1,12) distribution (Table I)



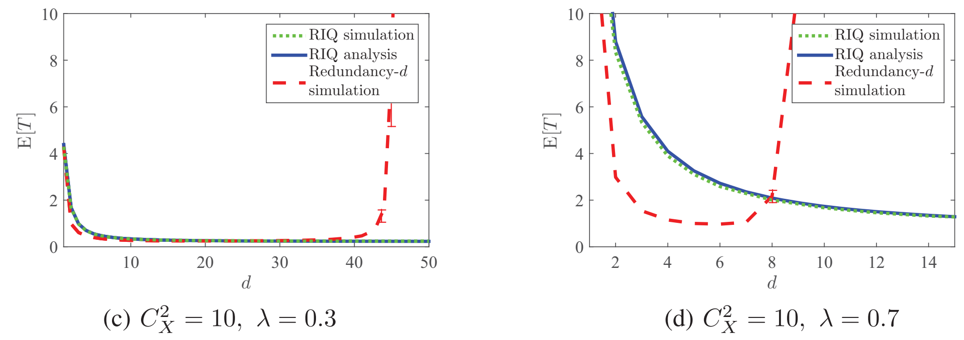


Fig. 5. $\mathbf{E}[T]$ vs. d under Redundancy-d (from simulation) and RIQ (from both simulation and analysis) for $X \sim H2$ with $\mathbf{E}[X] = \frac{1}{4.7}$ and $C_X^2 = 1$ (top row) and $C_X^2 = 10$ (bottom row), where $\mathbf{E}[R(1)] = \mathbf{E}[X] \cdot \mathbf{E}[S] = 1$ and $S \sim$ Dolly(1,12), 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 except where shown. At low arrival rate, RIQ and Redundancy-d perform similarly, but at high arrival rate Redundancy-d becomes unstable when d is large, whereas RIQ continues to achieve low $\mathbf{E}[T]$.

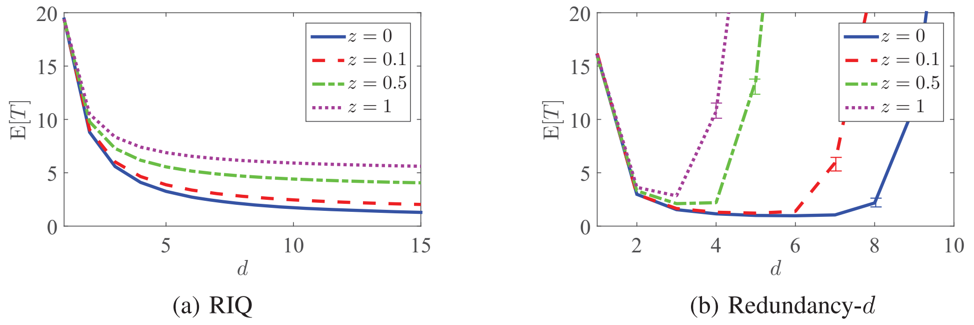


Fig. 6. Mean response time as a function of d under (a) RIQ (from analysis) and (b) Redundancy-d (simulated with $k = 1000$ servers, 95% confidence intervals are within the line except where shown), $\lambda = 0.7$, job sizes are $X \sim$ H2 with $\mathbf{E}[X] = \frac{1}{4.7}$ and $C_X^2 = 10$, and server slowdowns are $S \sim$ Dolly$(1, 12)$ for different constant 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.

with $\mathbf{E}[S] = 4.7$, and X follows a two-phase hyperexponential distribution with balanced means and with $\mathbf{E}[X] = \frac{1}{4.7}$:

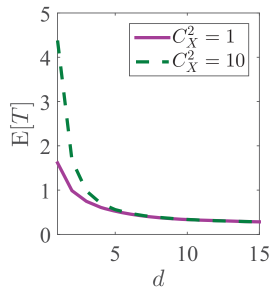
$$X \sim \begin{cases} \text{Exp}(\mu_1) & \text{w.p. } p \\ \text{Exp}(\mu_2) & \text{w.p. } 1-p \end{cases},$$

where

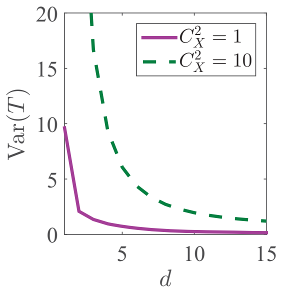
$$\frac{p}{\mu_1} = \frac{1-p}{\mu_2}.$$

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

In Figure 5 we compare RIQ (our analysis in Section V matches simulation) to Redundancy-d (simulated) at both low ($\lambda = 0.3$) and high ($\lambda = 0.7$) arrival rate where the inherent job size X has different squared coefficients of variation, C_X^2. At low load both policies perform similarly, which is unsurprising since under both policies many jobs find idle servers. Redundancy-d is slightly better since it allows *all* jobs to run multiple copies, not just jobs finding multiple idle servers. At high arrival rate, Redundancy-d becomes unstable



(a) $\mathbf{E}[T]$



(b) Var(T)

Fig. 7. (a) Mean, and (b) Variance of response time, T, (via analysis) under RIQ when $\lambda = 0.3$, $X \sim H2$ with $\mathbf{E}[X] = \frac{1}{4.7}$ and $C_X^2 = 1$ and 10, and $S \sim \text{Dolly}(1, 12)$. As d increases, both the mean and the variance of response time decrease; the improvement is more pronounced in the variance.

at high values of d, whereas mean response time under RIQ continues to decrease as a function of d.

Under both RIQ and Redundancy-d, mean response time increases as the cancellation time Z increases (see Figure 6). 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 higher moments of response time, which in computer systems are often more important metrics than the mean. Figure 7 shows the variance of response time, Var(T), under RIQ when $\lambda = 0.3$ and $C_X^2 = 1$ and 10 (computed analytically using the transform of T, see Section V-C). As d increases, Var(T) drops much more steeply than $\mathbf{E}[T]$, indicating that RIQ successfully overcomes the negative effects of the variable server slowdowns.

VII. Results: high d

One might think that mean response time should be lowest when $d = k$, since in the $d \ll k$ regime (Sections V and VI) we saw that $\mathbf{E}[T]$ decreases as d increases. However, this intuitively does not make sense. As a job runs on more servers, it achieves decreasing marginal runtime benefit from querying one more server (with respect to both minimizing server slowdown and increasing the likelihood of finding an idle server). Eventually, the job gets virtually no runtime benefit from increasing d, and instead only adds load to the system. Hence as d gets high, we would expect to see a turning point at which mean response time will begin to increase.

While our analysis from Section V does not apply when d becomes high, in this section we demonstrate that, unlike Redundancy-d, RIQ does not become unstable even as d gets large. We begin by discussing simulation results that show the worst-case behavior of RIQ (we omit Redundancy-d from our graphs in this section because we have already seen that Redundancy-d becomes unstable at relatively low d). We then derive an analytical upper bound on mean response time under RIQ for any k, d, S, X, and Z, which allows us to prove that RIQ does not become unstable as d gets large.

Figure 8 shows mean response time as a function of d when $k = 1000$, inherent job sizes are hyperexponential with $C_X^2 = 10$, and $S \sim \text{Dolly}(1, 12)$. Surprisingly, we see that the system is never unstable, even when $d = k$. In Section VII-A we explain this intuitively, and we formalize the stability conditions under RIQ in Section VII-B.

A. Intuition for why RIQ is Stable

One might think that as each job makes more copies, the added work causes the servers to be always busy, thereby driving queue lengths to infinity. Indeed, we see in Figure 8 that $\rho \to 1$ as $d \to k$. Nonetheless, this does not cause the system to become unstable because RIQ only allows jobs to replicate when there are idle servers. Effectively, whenever a server goes idle we can think of it as being given some extra work to do, where this extra work is some job's replicated copy. The extra work 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.

B. Formal Upper Bound on Mean Response Time under RIQ

Consider a system with k servers (unlike in the analysis in Section V, here we do not need to assume k is large). Our dispatching policy is RIQ, and d may take any value $0 < d \le k$. We allow S and X to follow any distribution with finite first and second moments. As defined in Section III, a job's runtime on a single server is $R(1) = S \cdot X$. The arrival rate λ can be anything such that $\lambda \cdot \mathbf{E}[R(1)] < 1$.

Theorem 1. *The response time under RIQ is upper bounded by the response time in an M/G/1/vacation queue [6] with arrival rate $\lambda < 1/\mathbf{E}[R(1)]$, where $G = R(1) = S \cdot X$, and where the vacation time is $R(1) + Z$:*

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

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

Proof. The only part of the analysis of mean response time under RIQ in Section V-B that is not exact is Assumption 1. We remove our dependence on Assumption 1 by upper bounding each component of the analysis that uses Assumption 1.

We begin by conditioning on whether an arrival finds any idle servers:

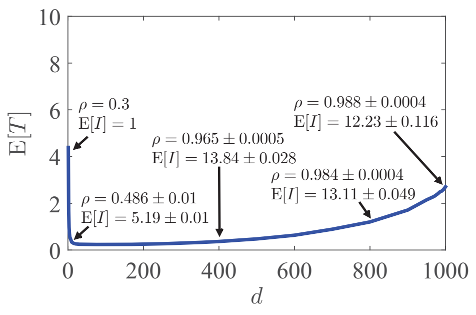
$$\begin{aligned}\mathbf{E}[T] = \mathbf{P}\left\{\begin{matrix}\text{job finds} \\ \text{idle servers}\end{matrix}\right\} \cdot \mathbf{E}\left\{T \,\middle|\, \begin{matrix}\text{job finds} \\ \text{idle servers}\end{matrix}\right\} \\ + \mathbf{P}\left\{\begin{matrix}\text{job finds no} \\ \text{idle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

In the analysis in Section V-B, we used Assumption 1 to approximate $\mathbf{P}\{\text{job finds idle servers}\}$. We can upper bound $\mathbf{E}[T]$ by instead assuming that no job ever finds an idle server:

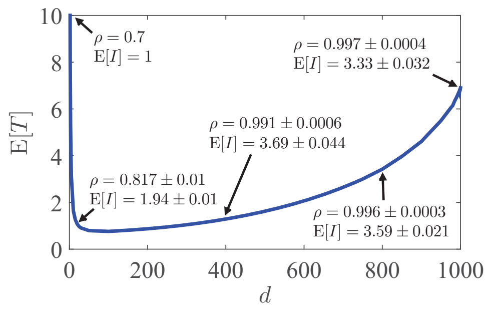
$$\begin{aligned}\mathbf{E}[T] &\le \mathbf{E}[T \mid \text{job finds no idle servers}] \\ &= \mathbf{E}[R(1)] + \mathbf{E}[T \mid \text{queueing}]^{\text{M}^*\text{/G/1/efs}}, \qquad (11)\end{aligned}$$

where the equality uses the same reasoning as in Section V-B.

In Section V-B, we further used Assumption 1 in two places when analyzing the M*/G/1/efs. First, we said that when the



(a) $\lambda = 0.3$



(b) $\lambda = 0.7$

Fig. 8. Mean response time under RIQ (simulated) as a function of d when $k = 1000$, $X \sim$ H2 with $\mathbf{E}[X] = \frac{1}{4.7}$ and $C_X^2 = 10$, $S \sim$ Dolly$(1, 12)$, 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 $\mathbf{E}[I]$.

server is busy arrivals occur with rate $\lambda\rho^{d-1}$. We will upper bound this by saying that arrivals occur with rate λ.

Second, we said that the first job served during a busy period experiences service time $R_0 + Z$, defined in (3). We will upper bound this by saying that the first job experiences runtime $R(1) + Z$. Since (11) forces all jobs to have a non-zero queueing time, we can think of the first job in the busy period as a "dummy" job that does not contribute to response time. Every time a server goes idle, we cause it to become busy working on a "dummy" job of size $R(1) + Z$, so the server is always busy no matter when the next real job arrives to the server. This is exactly an M/G/1/vacation queue with vacation time $R(1) + Z$. □

Figure 9 compares our upper bound to simulation results when $k = 1000$ and runtimes are deterministically equal to 1 and there is no cancellation cost. We consider deterministic S because, intuitively, this shows the worst possible case for redundancy: with no server slowdown variability, redundancy offers no possible runtime benefit. The only benefit of RIQ is therefore that it helps jobs find idle servers on which to run. While increasing d helps jobs find idle servers, it also means that jobs create many copies that *only* add load to the system. Hence we would expect deterministic server slowdowns to yield the worst performance as d increases. As we increase the variability of inherent job size X, the difference between actual mean response time and the upper bound increases. However, for all distributions of X, the difference between the exact mean response time and the upper bound at $d = 1$ (i.e., random dispatching; all jobs see an M/G/1 queue) is exactly the mean excess of $R(1)$. Hence RIQ can never perform more than an additive constant worse than random dispatching. We observe from simulation that RIQ actually does much better at high d than at $d = 1$ for nearly all parameter settings.

Theorem 2. *Under RIQ, the system is stable 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. □

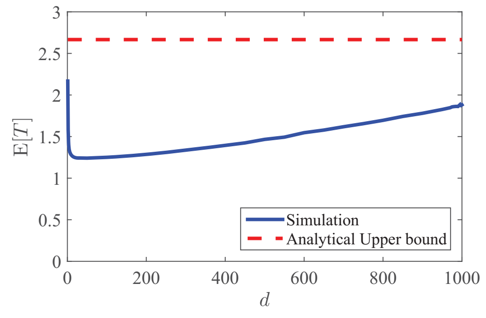


Fig. 9. Solid line shows mean response time under RIQ (simulated, 99% confidence intervals are within the line) when for all i, $R(i) = S \cdot X = 1$ is deterministic and $\lambda = 0.7$; dashed line shows the analytical upper bound. While the bound is not tight, it shows that mean response time under RIQ cannot become extremely high. In particular, the upper bound tells us that RIQ does not become unstable as d grows large.

VIII. Extensions and Variations on RIQ

There are many variations on the RIQ policy that can further improve performance. The analysis and upper bound presented in Sections 4 and 5 respectively extend easily to many of these variations, whereas others are not analytically tractable.

Several computer systems papers suggest waiting for a small amount of time after a job begins running; during this delay the system measures the slowdown factor S that the job is experiencing and begins running replicas only if S is large. It is straightforward to incorporate a delay into our analysis and upper bound for RIQ: adding a delay before replicas begin running is very similar to adding a cancellation cost after the first replica completes. Likewise, another idea used in practice is to replicate only jobs that have a small inherent size. When inherent job sizes are known upon arrival, the RIQ policy and its analysis and upper bound can be adapted easily to allow only jobs below a threshold inherent size to replicate.

RIQ is very conservative in choosing when jobs are allowed to replicate. When the arrival rate is low this conservatism is not necessary: Redundancy-d, which replicates all jobs no matter what, actually performs slightly better than RIQ. Instead of replicating jobs only when servers are idle, we can modify RIQ to allow jobs to replicate on any server whose queue length is below a certain threshold number of jobs n. Our analysis does not apply to this variation of RIQ; analyzing the threshold policy would require tracking the age of each replica. However, our upper bound applies if we consider the "vacation time" to be the sum of n instances of $R(1)$.

A final idea to improve the performance of RIQ is to use Join-the-Shortest-Queue (JSQ) dispatching instead of random dispatching when a job finds no idle servers. While this will surely improve RIQ's performance, JSQ is known to be extremely difficult to analyze.

IX. 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 the existing theoretical work on redundancy, most of which assumes an Independent Runtimes (IR) model where the 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 on which to run these copies. We derive a highly accurate approximation for both the mean and the transform of response time under RIQ. Furthermore, we derive an upper bound on mean response time under RIQ that shows that RIQ cannot become unstable 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 $S\&X$ 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] Ganesh Ananthanarayanan, Ali Ghodsi, Scott Shenker, and Ion Stoica. Effective straggler mitigation: Attack of the clones. In *NSDI 2013*.

[2] 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.

[3] 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*.

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

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

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

[7] 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.

[8] Kristen Gardner, Samuel Zbarsky, Mor Harchol-Balter, and Alan Scheller-Wolf. Analyzing response time in the Redundancy-d system. Poster, SIGMETRICS '16, Technical Report CMU-CS-15-141, 2015.

[9] 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.

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

[11] 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.

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

[13] 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.

[14] 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.

[15] 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.

[16] 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.

[17] 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.

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

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

[20] 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.

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

[22] 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.

[23] 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.

[24] 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.