$S\&X$: Decoupling Server Slowdown (S) and Job Size (X) in Modeling Job Redundancy

Kristen Gardner[1] **Mor Harchol-Balter**[1]
Alan Scheller-Wolf[2]

May 2016
CMU-CS-16-109

School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

[1]School of Computer Science, Carnegie Mellon University, Pittsburgh, PA, USA
[2]Tepper School of Business, Carnegie Mellon University, Pittsburgh, PA, USA

This work was supported by NSF-CMMI- 1538204, NSF-CMMI-1334194, and NSF-CSR-1116282, by the Intel Science and Technology Center for Cloud Computing, and by a Google Faculty Research Award 2015/16.

Keywords: Redundancy, Replication, Parallel servers, Server slowdown, Task Assignment, RIQ, Redundant-to-Idle-Queue, Redundancy-d

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 models with independent service times. This is unrealistic because a job with large size should actually remain large at all servers, rather than getting a new independent size at each server. The unrealistic independence assumption 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-on-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 12X longer on one machine than another [1], or even 27X longer [22]. 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), where multiple VMs 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, 5, 15, 19, 2, 16]). 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 though jobs are immediately dispatched to servers via the front-end load balancer (no central queue), where this front-end load balancer doesn't require 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 [12, 10, 19, 17, 18, 11, 8, 7, 9, 13, 4]. All of these papers rely on a key assumption: *A job's runtime at different servers is independent.* In many papers, the job's runtime at different servers is an independent exponential distribution. In some it is an independent distribution with higher variability than an exponential. Either way, this independence assumption leads to the conclusion that (barring cancellation costs) "more redundancy is always better."

But this independence assumption can be problematic. 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. Under the independence assumption, this does not happen: the job is assigned a different, independent, service time on different servers. When the job runs on multiple servers, it experiences the minimum service time of all those servers' independent service times. 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.[2]

In this paper we remedy the unrealistic independence assumption that has been used in analyzing redundancy by introducing a new model, called the $S\&X$ model. The $S\&X$ model explicitly decouples the server slowdown (represented by the random variable S) from the inherent job size

[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, and the decision to replicate might be made after the initial dispatch. For the purposes of this paper, however, we stick to the simple model where a job is an atomic unit, and the decision to replicate a job or not is made at the moment that the job is dispatched.

[2] In applications where the inherent work associated with a job is small and server slowdown dominates, the independence assumption is not problematic.

(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, a policy like Redundancy-d that replicates every arriving job to d queues [8]. Under the original analysis in [8], which assumes independent service times, mean response time only decreases as we increase d. By contrast, under 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 will eventually become unstable because of the increased load of replication, sending mean response time to infinity. Unfortunately, we are at a loss to figure out 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 independence 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 other redundancy policies. We seek policies which are robust in that they provably will not go into overload, which are less sensitive to d, and which are analytically tractable within the $S\&X$ model.

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 additonal replicas). We are able to approximately analyze RIQ within the $S\&X$ model, for any distribution of S, any distribution of X, and any cancellation cost. We derive both mean response time as a function of d and also the full response time distribution. We show that 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. In fact we derive an analytical upper bound on the response time of RIQ under any S and X for any d. Thus RIQ represents a provably robust and analytically understood replication policy, which is also simple enough to be appealing to practitioners.

The remainder of this paper is outlined in Figure 1.

2 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 that the same job's service times are independent across servers and that running times are exponentially distributed. When these assumptions are relaxed others 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

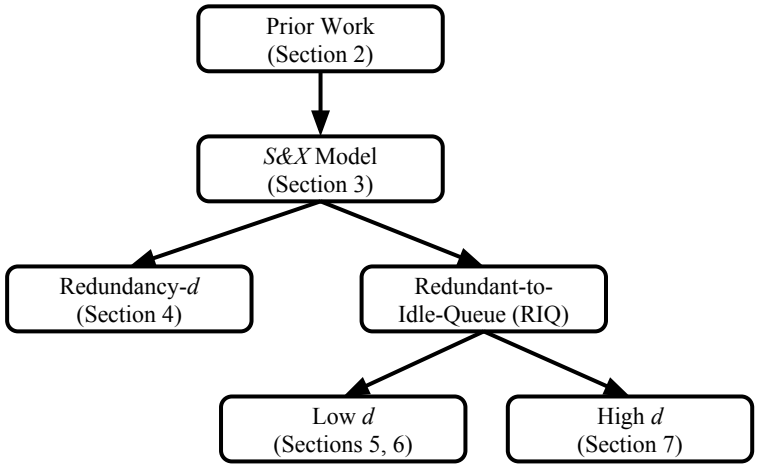


Figure 1: Outline of the remainder of this paper.

empirical systems studies.

In [12, 18] it was shown that when service times are i.i.d. across servers and follow a distribution with decreasing failure rate, it is optimal to replicate jobs at all servers. However these papers do not provide any analysis quantifying response time in redundancy systems. 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 [17, 10, 11, 20], assuming each job's service times are independent across servers. The results suggest that as n increases (i.e., each job makes more copies) mean response time decreases [11]. While [10] does consider a model in which a job's service time 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 [17, 11]. In the case where $k = 1$ and service times are exponentially distributed and independent across servers, the full distribution of response time is analyzed in [7, 8]. The analysis presented in [8] shows that as the number of copies per job increases, mean response time decreases.

The story told by empirical systems work is a bit more cautious than the theoretical results lauding the benefits of redundancy. Theoretical results suggest that the more copies created, the lower mean response time will be, but practical studies have shown that creating too many copies can lead to unacceptably high response times and even instability [19]. This gap emerges because the strong assumptions required for the theoretical analysis described above 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. Working in a more realistic setting, systems researchers have observed that they need to design more sophisticated dispatching and scheduling policies in order to leverage the potential benefits of redundancy. One idea is to replicate only

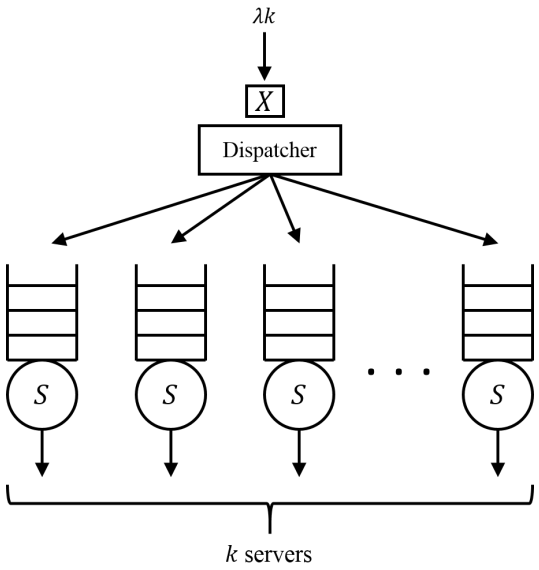


Figure 2: The $S\&X$ model. The system has k servers and jobs arrive as a Poisson process with rate λk. Each job has an inherent size X. A job that runs on server j is slowed down by a factor S_j. A job's running time on a single server is $R(1) = X \cdot S$; if the job enters service at $i > 1$ servers simultaneously it experiences running time $R(i) = X \cdot \min\{S_1, \ldots, S_i\}$.

small jobs to limit the amount of load added to the system; this leads to a 46% reduction in mean response time [1]. Many systems designed to reduce latency in MapReduce systems begin running replicated copies of jobs only after waiting for some delay to identify which jobs are experiencing significant slowdown [3, 23]. More recently, [16, 2] build on this idea by combining delayed execution of replicas with scheduling policies that reserve a set of servers on which to run these replicas (this assumes that 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 introduce a new model called the $S\&X$ model that removes some of the problematic assumptions made in earlier theoretical work on redundancy. We revisit the redundancy policies used in real computer systems in Section 8, where we discuss these policies in the context of the $S\&X$ model.

3 The $S\&X$ Model

We consider a system with k homogeneous servers (see Figure 2). 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.

In classical queueing theory, a job's service time is denoted by a single random variable Y. The meaning of Y can be interpreted in one of two mutually exclusive ways. First, Y could mean the job's inherent size. That is, no matter which server the job runs on, it will always

Table 1: The Dolly(1,12) empirical slowdown distribution [1]. The server slowdown ranges from 1 to 12.

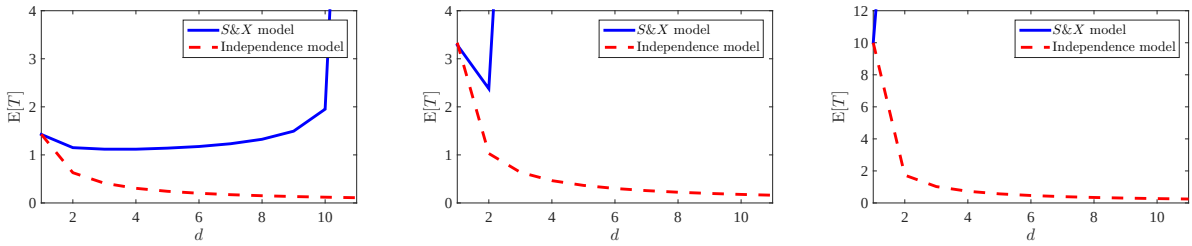
S	Probability
1	0.23
2	0.14
3	0.09
4	0.03
5	0.08
6	0.10
7	0.04
8	014
9	0.12
10	0.021
11	0.007
12	0.002

take the same amount of time to run. Alternatively, Y could mean the job's running time on a particular server. That is, Y is server-dependent; if the same job were to run on a different server, it would experience a completely different running time. The first interpretation implies that there is no variability in an individual server's speed: the implication is that factors such as network congestion, caching effects, and background load have absolutely no effect on a job's running time. The second interpretation implies that the server's speed is so unpredictable that it makes job-dependent factors such as the amount of computation required totally irrelevant in determining how long it will take for the job to run. Neither of these interpretations is sufficient to model the behavior of real systems.

We introduce the $S\&X$ *model*, which captures the effects of both job-dependent and server-dependent factors on a job's running time. 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 is run on one server, we say that its *runtime* is $R(1) = S \cdot X$. (S is assumed to be ≥ 1 and can be thought of as the amount by which the inherent size, X, is "stretched.") If a job of inherent size X begins running on i servers simultaneously, it sees an independent instance of S on each server. Its runtime is then $R(i) = X \cdot \min\{S_1, \ldots, S_i\}$. 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 server 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.

In much of the remainder of this paper, we focus on one particular S distribution, called the Dolly(1,12) distribution, which was measured empirically in [1] and is shown in Table 1.[3]

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



(a) $\lambda = 0.3$ (b) $\lambda = 0.7$ (c) $\lambda = 0.9$

Figure 3: Mean response time as a function of d under Redundancy-d when $k = 1000$ servers, job sizes are $X \sim \text{Exp}(1)$, and server slowdowns are $S = 1$. Under the Independence model (where a job's runtimes are i.i.d. across servers) (dashed red line), $\mathbf{E}[T]$ always decreases as a function of d. In the $S\&X$ model (solid blue line), the system becomes unstable when d gets high.

We use ρ to denote the system load. Unlike in a traditional queueing system, in the $S\&X$ model the system load and stability region 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, in which we introduce the Redundancy-d and Redundant-to-Idle-Queue dispatching policies respectively.

4 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. The job is complete as soon as its first copy completes service, at which time all remaining copies are cancelled.

Response time under the Redundancy-d policy has been analyzed in the "Independence model," where a job's runtimes are i.i.d. across servers [8]. Note that the Independence model differs from the $S\&X$ model, where a job's runtimes on different servers are the product of the job's inherent size X, which is the same on all servers, and the server slowdown S, which is i.i.d. across servers. Figure 3 compares mean response time under Redundancy-d in the Independence model to that in the $S\&X$ model in the simple setting where the server slowdown is $S = 1$ and inherent job size X is exponentially distributed (in the Independence model, this simply means that runtimes are exponentially distributed and i.i.d. across servers). In the Independence model, as d increases mean response time always decreases. 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. The value of d at which the system becomes unstable depends on λ; as λ gets higher instability occurs at lower and lower d. While Figure 3 shows results for exponentially distributed X and deterministic S, we observe the same trends in the $S\&X$ model under more realistic distributions where X has higher variability and S follows the Dolly(1,12) distribution from Table 1 (see Figure 4).

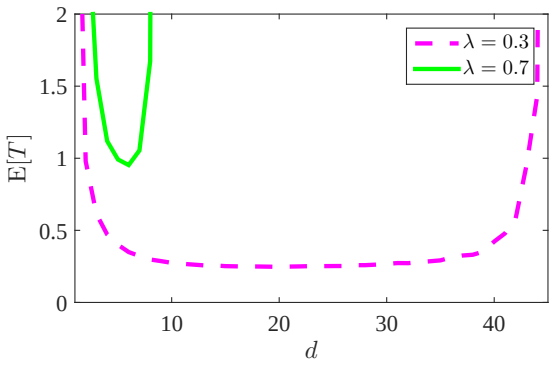


Figure 4: Redundancy-d in the $S\&X$ model when $k = 1000$ servers, job sizes are $X \sim \text{Hyperexponential}$ with $C_X^2 = 10$, and server slowdowns are $S \sim \text{Dolly}(1, 12)$ from Table 1. As d increases, $\mathbf{E}[T]$ decreases at first but eventually the system becomes unstable.

While Figures 3 and 4 show that Redundancy-d can become unstable in the $S\&X$ model if d is too high, it is difficult to prove this result 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 involves 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, since jobs can enter service at different times on different servers, even 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 3 highlights the importance of making the right modeling assumptions. In the unrealistic Independence model, the lessons learned about the benefits of redundancy are overly optimistic and misleading: we are led to believe that more redundancy is always better. The results under the $S\&X$ model tell a very different story. 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. By having all jobs create copies without regard to the system state, 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.

5 Redundant-to-Idle-Queue

In this section we introduce a new dispatching policy, Redundant-to-Idle-Queue (RIQ).[4] The RIQ policy, which we described in detail in Section 5.1, is motivated by the fact that in the $S\&X$ model it is important to avoid creating a large number of extra copies when the system does not have sufficient capacity with which to run these copies.

The rest of this section is devoted to analyzing the performance of RIQ. We begin by approximately analyzing mean response time (Section 5.2) 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 5.3). In Section 5.4 we extend our analysis to the case where cancelling the extra copies takes some amount of time. In Section 5.5 we evaluate the quality of our approximations by comparing our analytical results to simulation.

5.1 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 of the queried servers are busy, the job chooses one of the d servers at random and joins the queue at that server. If $0 < i \leq d$ of the d servers are idle, the job enters service at all i idle servers.

The RIQ policy is motivated by the fact that redundancy can add too much load to the system, causing unacceptable increases in response time and even instability. By only allowing jobs to create redundant copies when there are idle servers, we ensure that we are only adding load to the system when we have server capacity to spare.

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. Writing down an expression for load under RIQ is more straightforward 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 in Section 5.2.1.

Throughout the remainder of this section we rely on two assumptions:

1. **Asymptotically Independent Idleness assumption:** we assume that 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 $s_1, \ldots, s_d$.

2. **Small d assumption:** We assume that $d \ll k$.

[4] The name "RIQ" is influenced by the Join-Idle-Queue (JIQ) policy, under which each arrival is dispatched to a single idle queue, if one exists [14].

5.2 Analysis: Mean Response Time

Our goal in this section is to derive mean response time under RIQ. We begin by conditioning on whether an arrival to the system finds any idle servers:

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

where the second line is due to the asymptotically independent idleness assumption. We defer our derivation of ρ to the end of the section.

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

$$\begin{aligned}\mathbf{E}[T \mid \text{job finds idle servers}] &= \sum_{i=1}^{d} \mathbf{P}\{\text{job finds } i \text{ idle servers} \mid \text{job finds idle servers}\} \cdot \mathbf{E}[R(i)] \\ &= \sum_{i=1}^{d} \frac{(1-\rho)^i \rho^{d-i} \binom{d}{i}}{1-\rho^d} \mathbf{E}[R(i)]. \end{aligned} \tag{2}$$

To find $\mathbf{E}[T \mid \text{job finds no idle servers}]$, 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 system is idle, arrivals occur as a Poisson process with rate λd. This is because jobs arrive to the system as a whole with rate λk, each arriving job polls our particular server with probability $\frac{d}{k}$, and each job that polls our server while it is idle runs on our server.

2. When the system is busy, arrivals occur as a Poisson process with rate $\lambda' = \lambda\rho^{d-1}$. This is because jobs arrive to the system as a whole with rate λk, each arriving job polls our particular server with probability $\frac{d}{k}$, and each job that polls our server while it is busy runs on our server if all $d-1$ other servers that the job polls are also busy (probability ρ^d) and then the job chooses our server (probability $\frac{1}{d}$).

3. All jobs that find our server idle experience runtime R_0, where

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

   Note that the probability used here is not the same as the probability that a job finds i idle servers in equation (2). This is because equation (2) looks at the system from the *job's* perspective, while equation (3) looks at the system from the *server's* perspective.

4. All jobs that find our server busy experience 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. (In a system with exceptional first service, the first job served during a busy period has a different service time distribution than all other jobs served during the busy period.)

We want to find $\mathbf{E}[T \mid \text{job finds no idle servers}]$. To do this, we first observe that

$$\mathbf{E}[T \mid \text{job finds no idle servers}] = \mathbf{E}[R(1)] + \mathbf{E}[T_Q \mid \text{queueing}]^{\text{M}^*/\text{G}/1/\text{efs}} \tag{4}$$

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

where $P_Q^{\text{M}^*/\text{G}/1/\text{efs}}$ is the probability that an arrival has to wait in the queue in an M*/G/1/efs.

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*

$$\mathbf{E}[T_Q]^{\text{M}^*/\text{G}/1/\text{efs}} = \mathbf{E}[T_Q]^{\text{M}/\text{G}/1/\text{efs}} = \mathbf{E}[T]^{\text{M}/\text{G}/1/\text{efs}} - \mathbf{E}[S]^{\text{M}/\text{G}/1/\text{efs}}, \tag{6}$$

where

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

is derived in [21] and

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

where $\mathbf{E}[N_B] = \lambda' \cdot \frac{\mathbf{E}[R_0]}{1-\lambda'\mathbf{E}[R(1)]}$ is the expected number of arrivals during a busy period with exceptional first service.

Proof. To find $\mathbf{E}[T]^{\text{M}^*/\text{G}/1/\text{efs}}$, we would like to be able to follow the same derivation that one usually uses to find the transform of response time in an M/G/1/efs. Unfortunately, the analysis does not apply 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, we observe that 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 of which is derived in [21]. □

Finally, in equation (5) we need to know the probability that a job has to queue in an M*/G/1/efs:

$$P_Q^{\text{M}^*/\text{G}/1/\text{efs}} = \frac{\mathbf{E}[N_B]}{\mathbf{E}[N_B]+1}, \tag{7}$$

where $\mathbf{E}[N_B]$ is the expected number of arrivals during an M*/G/1/efs busy period as in Lemma 1.

Combining equations (1)-(7) gives a closed-form expression for mean response time under RIQ, modulo a closed-form expression for ρ.

5.2.1 Deriving ρ

Our last step is to 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. Using the asymptotically independent idleness assumption, we can write

$$\rho = \lambda \left(\sum_{i=1}^{d} i \cdot (1-\rho)^i \rho^{d-i} \binom{d}{i} \mathbf{E}[R(i)] + \rho^d \cdot \mathbf{E}[R(1)] \right). \tag{8}$$

It is not immediately obvious that this expression gives the probability that a server is busy, hence we provide an alternative derivation using renewal-reward theory. We define a cycle as the moment from when a server becomes idle until it is next idle. We have

$$\rho = \frac{\mathbf{E}[B]}{\mathbf{E}[B] + \frac{1}{\lambda d}}, \tag{9}$$

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

Noting that $\mathbf{E}[R_0]$ is defined in terms of ρ, equation (9) gives us an expression for ρ in terms of itself. It is easy to verify algebraically that equations (8) and (9) are equivalent. When $d = 2$, the equation is 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, since we do not have a closed-form expression for ρ, it is difficult to understand intuitively what this stability condition means. In Section 7 we derive an upper bound on mean response time under RIQ which gives us an alternative condition: the system is stable when $\lambda \cdot \mathbf{E}[R(1)] < 1$.

5.3 Analysis: Transform of Response Time

Our approach in Section 5.2 allows us to find the transform of response time as well as mean response time.

As before, we begin by conditioning on whether an arrival to the system finds any idle servers:

$$\widetilde{T}(s) = (1-\rho^d)\widetilde{T}(s \mid \text{job finds idle servers}) + \rho^d \widetilde{T}(s \mid \text{job finds no idle servers}), \tag{10}$$

where ρ is the probability that a server is busy, derived in Section 5.2.1.

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:

$$\widetilde{T}(s \mid \text{job finds idle servers}) = \sum_{i=1}^{d} \frac{(1-\rho)^i \rho^{d-i} \binom{d}{i}}{1-\rho^d} \widetilde{R(i)}(s). \tag{11}$$

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 5.2:

$$\widetilde{T}(s \mid \text{job finds no idle servers}) = \widetilde{R(1)}(s) \cdot \widetilde{T_Q}(s \mid \text{queueing})^{\text{M*/G/1/efs}} \tag{12}$$

$$\widetilde{T_Q}(s \mid \text{queueing})^{\text{M}^*/\text{G}/1/\text{efs}} = \frac{\widetilde{T_Q}(s)^{\text{M}^*/\text{G}/1/\text{efs}}}{\mathbf{P}\left\{\text{queueing}\right\}^{\text{M}^*/\text{G}/1/\text{efs}}}, \tag{13}$$

where $P_Q^{\text{M}^*/\text{G}/1/\text{efs}}$ is given in (7).

Lemma 2. *The transform of 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 server is busy is λ' is*

$$\begin{aligned}\widetilde{T_Q}(s)^{\text{M}^*/\text{G}/1/\text{efs}} &= \widetilde{T_Q}(s)^{\text{M}^*/\text{G}/1/\text{efs}} \\ &= \frac{1-\lambda'\mathbf{E}\left[R(1)\right]}{1-\lambda'\mathbf{E}\left[R(1)\right]+\lambda'\mathbf{E}\left[R_0\right]} \cdot \frac{(\lambda'-s)\widetilde{R_0}(s)-\lambda'\widetilde{R(1)}(s)}{\lambda'-s-\lambda'\widetilde{R(1)}(s)} \cdot \frac{1}{\widetilde{S}(s)^{\text{M}^*/\text{G}/1/\text{efs}}},\end{aligned}$$

as derived in [21], where

$$\widetilde{S}(s)^{\text{M}^*/\text{G}/1/\text{efs}} = \frac{1}{\mathbf{E}\left[N_B\right]+1} \cdot \widetilde{R_0}(s) + \frac{\mathbf{E}\left[N_B\right]}{\mathbf{E}\left[N_B\right]+1} \cdot \widetilde{R(1)}(s) \tag{14}$$

and $\mathbf{E}\left[N_B\right]$ is as in Lemma 1.

In a few special cases, we can use the transform of response time to find an approximation for the distribution of response time under RIQ. We begin by conditioning on the number of idle servers an arrival to the system finds:

$$\begin{aligned}\mathbf{P}\left\{T>t\right\} &= \sum_{i=1}^{d} \mathbf{P}\left\{\text{job found } i \text{ idle servers}\right\} \cdot \mathbf{P}\left\{R(i)>t\right\} \\ &\quad + \mathbf{P}\left\{\text{job found no idle servers}\right\} \cdot \sum_{n=1}^{\infty} \mathbf{P}\left\{n \text{ jobs in queue}\right\} \cdot \mathbf{P}\left\{R(1)_e + \sum_{i=1}^{n} R(1) > t\right\} \\ &= \sum_{i=1}^{d} (1-\rho)^i \rho^{d-i} \cdot \mathbf{P}\left\{R(i)>t\right\} + \rho^d \cdot \sum_{n=1}^{\infty} \mathbf{P}\left\{n \text{ jobs in queue}\right\} \cdot \mathbf{P}\left\{R(1)_e + \sum_{i=1}^{n} R(1) > t\right\}.\end{aligned} \tag{15}$$

Note that the term $R(1)_e$ term is an approximation because here we assume that all jobs, including the first job served in the busy period, experience runtime $R(1)$. In reality, the first job experiences runtime $R(i)$, where $1 \leq i \leq d$.

To find $\mathbf{P}\left\{n \text{ jobs in queue}\right\}$, we use Distributional Little's Law to find the transform of the number of jobs in system in an M/G/1/efs:

$$\hat{N}(z) = \tilde{T}(\lambda(1-z)) = \frac{1-\lambda'\mathbf{E}\left[R(1)\right]}{1-\lambda'\mathbf{E}\left[R(1)\right]+\lambda'\mathbf{E}\left[R_0\right]} \cdot \frac{z\widetilde{R_0}(\lambda'(1-z)) - \widetilde{R(1)}(\lambda'(1-z))}{z-\widetilde{R(1)}(\lambda'(1-z))}.$$

Then we find $\mathbf{P}\left\{n \text{ jobs in queue}\right\}$ as:

$$\mathbf{P}\left\{n \text{ jobs in queue}\right\} = \frac{1}{j!} \cdot \hat{N}^{(j)}(z) \mid_{z=0}.$$

While the form in equation (15) holds for any S and X, computing the convolution to find $\mathbf{P}\left\{\sum_{i=1}^{n} R(1) > t\right\}$ is not always easy. In a few special cases, for example, if $R(1)$ is deterministic or exponential, the convolution is straightforward. In such cases, (15) gives an approximation for the distribution of response time which we can evaluate numerically. For generally distributed $R(1)$ (i.e., generally distributed X and S), finding the distribution of response time is challenging.

5.4 Cancellation Costs

When 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.

When we include the cancellation time, the analysis changes in only one place. We modify Equation (3) to define

$$R_0 = \begin{cases} R(1) & \text{w.p. } \rho^{d-1} \\ R(i) + z & \text{w.p. } (1-\rho)^{i-1}\rho^{d-1}\binom{d-1}{i-1}, \;\; 1 < i \leq d. \end{cases} \tag{16}$$

The rest of the analysis remains the same as above. Clearly it is easy to make z a random variable rather than a constant if desired.

5.5 Quality of Approximation

Our analysis relies on the assumption that the servers are idle independently. This assumption is not true in general: when a server is busy, the arrival rate to that server depends on whether other servers are busy or idle, since a job can only join the queue at a busy server if all of the servers it queried were busy. Furthermore, if d is very close to k then a single job can cause nearly all of the servers to become busy.

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 5 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 and more accurate; by the time $k = 500$ the analysis and simulation results are indistinguishable. 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.

6 Results: $d \ll k$

Our goal in this section is to evaluate the performance of RIQ using our analysis from Section 5. Throughout the section we set $R(1) = 1$, $k = 1000$ servers, and we only consider $d \ll k$. In

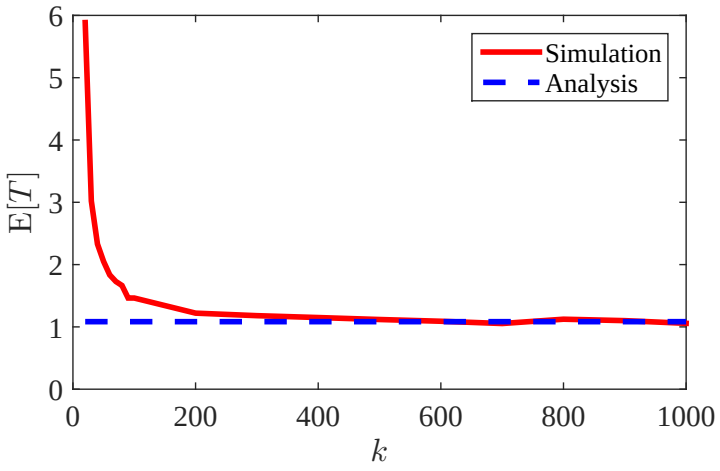


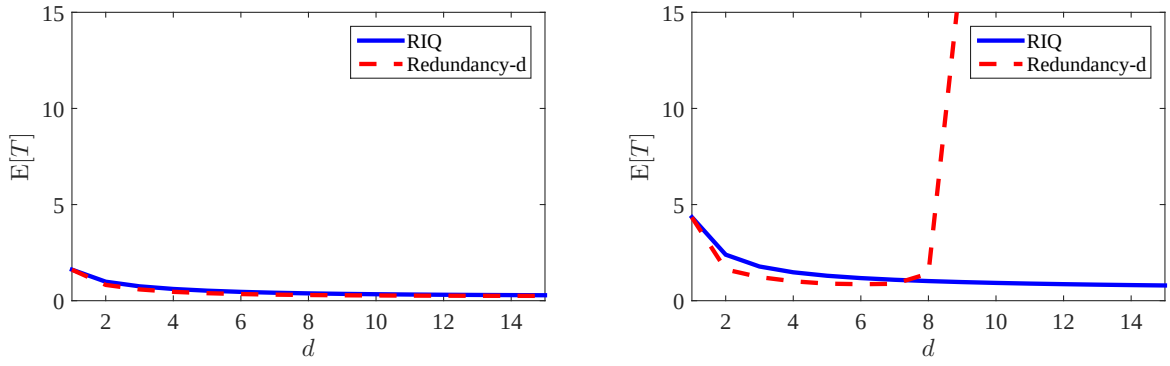
Figure 5: Comparing analytical (dashed blue line) and simulated (solid red line) mean response time under RIQ when $\lambda = 0.7$, $d = 20$, $X \sim \text{Hyperexponential}$ with $C_X^2 = 10$, and $S \sim \text{Dolly}(1, 12)$. As k increases, our analytical result becomes a very good approximation for mean response time.

the $d \ll k$ regime our analysis provides a good approximation for performance under RIQ – see Section 5.5. Later, in Section 7 we study what happens when d is close to k.

In Figure 6 we compare RIQ to Redundancy-d at different values of C_X^2 and at both low ($\lambda = 0.3$) and high ($\lambda = 0.7$) arrival rate. We assume that the server slowdown distribution follows the Dolly(1,12) empirical distribution shown in Table 1. The results for RIQ are from our analysis in Section 5; Redundancy-d is simulated. At low load RIQ and Redundancy-d perform similarly, which is unsurprising since under both policies jobs tend to find idle servers. Redundancy-d is perhaps a bit better since it allows all jobs to run multiple copies, not just jobs finding multiple idle servers; the arrival rate is low enough that the benefit that each job receives from running multiple copies outweighs any additional queueing that time later arrivals experience due to these extra copies. At high load, Redundancy-d becomes unstable at high values of d, whereas mean response time under RIQ continues to decrease as a function of d.

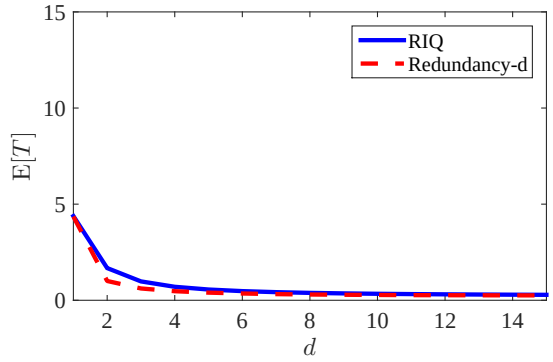
So far we have assumed that when one copy of a job completes, the other copies are cancelled instantaneously. Our analysis of RIQ also allows for non-zero cancellation times. In Figure 7 we see that under both RIQ and Redundancy-d, $\mathbf{E}[T]$ increases as the cancellation time z increases. Under Redundancy-d, as z increases the value of d at which the system becomes unstable decreases. Under RIQ, the system does not become unstable as a function of the cancellation time, indicating that RIQ is *more robust* than Redundancy-d.

Thus far we have only considered mean response time, but in Section 5.3 we analyze the distribution of response time under RIQ. Figure 8 compares response time under RIQ (from analysis) to that under Redundancy-d (simulated) at both the mean and the 95th percentile. Here $\lambda = 0.7$ and we assume that both X and S are deterministic; this enables us to compute the convolution required to find the response time distribution (see Section 5.3). Under both RIQ and Redundancy-

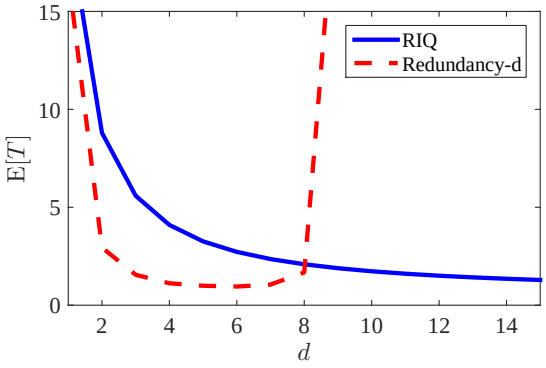


(a) $C_X^2 = 1,\ \lambda = 0.3$

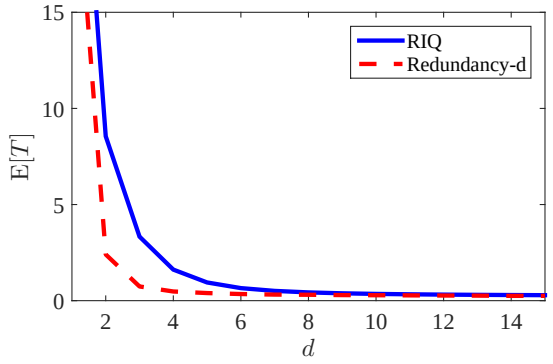
(b) $C_X^2 = 1,\ \lambda = 0.7$



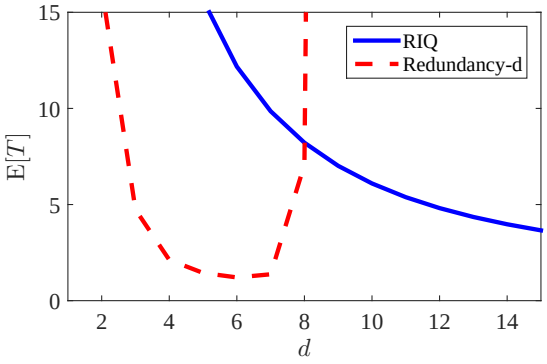
(c) $C_X^2 = 10,\ \lambda = 0.3$



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

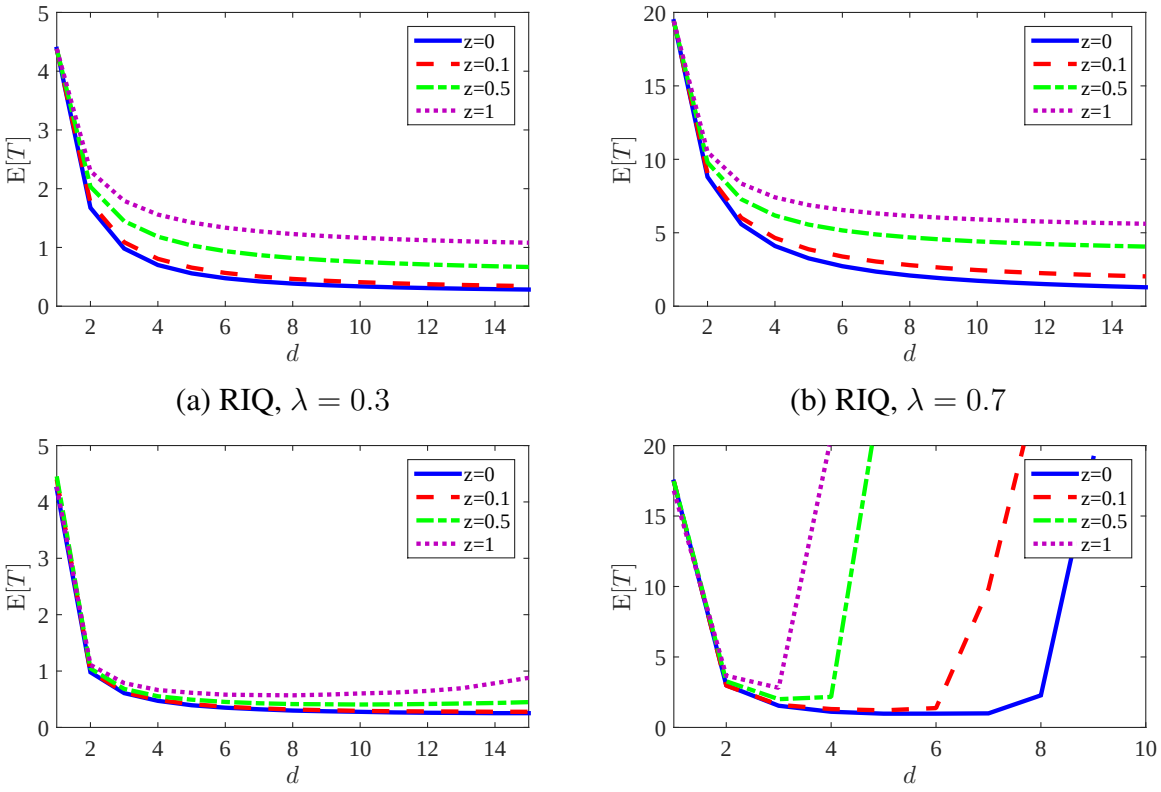


(e) $C_X^2 = 100,\ \lambda = 0.3$



(f) $C_X^2 = 100,\ \lambda = 0.7$

Figure 6: $\mathbf{E}[T]$ vs. d under Redundancy-d (simulated with $k = 1000$ servers) and RIQ for $X \sim H2$ with $C_X^2 = 1$ (top row), $C_X^2 = 10$ (middle row), and $C_X^2 = 100$ (bottom row), where $\mathbf{E}[R] = \mathbf{E}[X] \cdot \mathbf{E}[S] = 1$ and $S \sim$ Dolly(1,12), under low arrival rate ($\lambda = 0.3$, left) and high arrival rate ($\lambda = 0.7$, right). 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 mean response time.



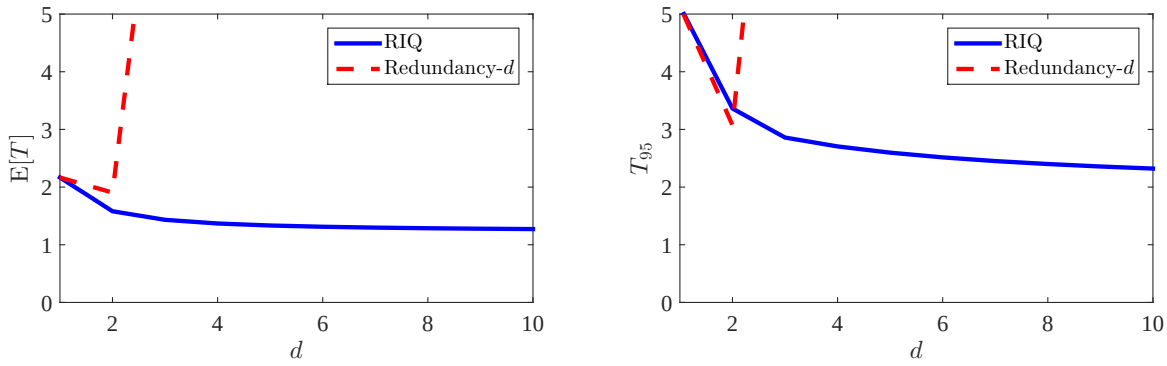
(a) RIQ, $\lambda = 0.3$

(b) RIQ, $\lambda = 0.7$

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

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

Figure 7: Mean response time as a function of d under RIQ (top) and Redundancy-d (bottom) when $k = 1000$ servers, job sizes are $X \sim \text{Hyperexponential}$ with $C_X^2 = 10$, and server slowdowns are $S \sim \text{Dolly}(1, 12)$, under low ($\lambda = 0.3$, left) and high ($\lambda = 0.7$, right) arrival rate for different cancellation times z. As the cancellation time 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) $\lambda = 0.7$

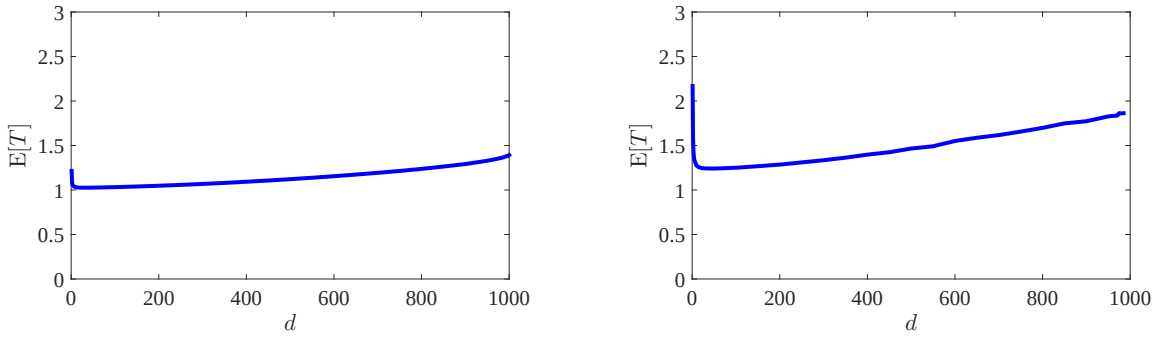
Figure 8: Comparing RIQ (from analysis) and Redundancy-d (simulated) when $k = 1000$ servers and $R = X \cdot S = 1$ is deterministic. (a) Mean response time, (b) 95th percentile of response time. As d increases both the mean and the 95th percentile of response time greatly decreases under RIQ, whereas after an initial improvement both increase as d further increases under Redundancy-d.

d, the improvement achieved in going from $d = 1$ to $d = 2$ is much greater at the 95th percentile of response time than at the mean. However, while Redundancy-d quickly becomes unstable, the 95th percentile of response time under RIQ continues to decrease as d increases. When $d = 10$, the 95th percentile of response time under RIQ is 2.32; this is only 7% higher than the *mean* response time when $d = 1$, indicating that RIQ is extremely effective at reducing the tail of response time. This is important given that in computer systems, the tail of response time is often a more critical metric than the mean.

7 Results: high d

Our analysis and results from Section 6 suggest that under RIQ $\mathbf{E}[T]$ decreases as a function of d, which means that setting $d = k$ should lead to the lowest mean response time. However, this intuitively does not make sense. As a job runs on more and more servers, it achieves decreasing marginal runtime benefit from querying one more server (both with respect to minimizing server slowdown and with respect to increasing the likelihood of finding an idle server). Eventually, the job is getting no runtime benefit from increasing d, and instead is only adding 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 because the extremely small service time benefit to the replicated job is not enough to overcome the pain caused by the extra load.

In this section, we ask whether, as d becomes closer to k, RIQ is subject to the same pitfalls as Redundancy-d. In Sections 5 and 6 we only considered cases in which $d \ll k$, which allowed us to derive approximate expressions for response time under RIQ. When d becomes high our analysis no longer applies, and the dependencies between different servers make it extremely difficult to approximate response time. Nonetheless, in this section we will see that, unlike Redundancy-d,



(a) $\lambda = 0.3$ (b) $\lambda = 0.7$

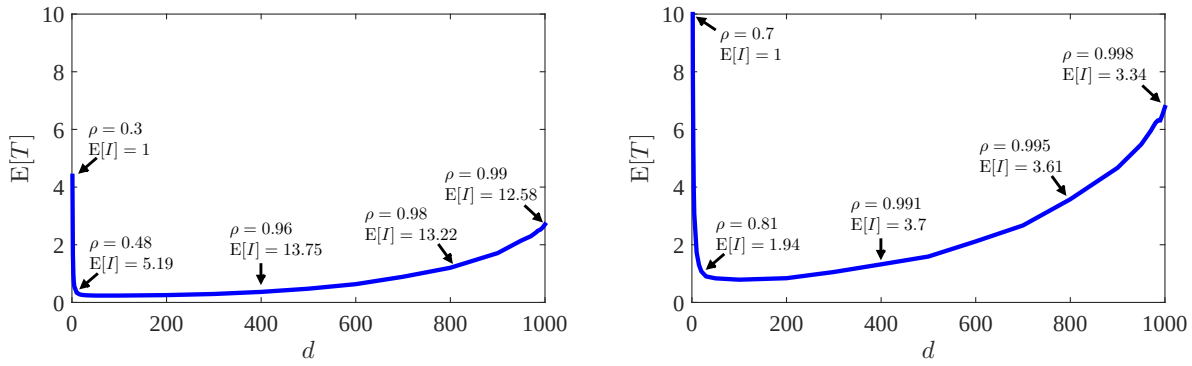
Figure 9: Mean response time under RIQ as a function of d when $k = 1000$, service times are deterministic with $R = X \cdot S = 1$, and (a) $\lambda = 0.3$, (b) $\lambda = 0.7$. When d is low, increasing d causes mean response time to decrease, as predicted by the analysis. When d is high (close to k), mean response time increases.

RIQ does not become unstable even as d gets large (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 begin by discussing simulation results that show the worst-case behavior of RIQ. 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 9 shows mean response time as a function of d when $k = 1000$ and running times are deterministically equal to 1. We consider deterministic running times because, intuitively, this shows the worst possible case for redundancy: with no server slowdown variability, there is no possible benefit from running the same job on multiple servers. The only benefit to RIQ is therefore that it help jobs find idle servers on which to run. While increasing d increases the probability that a job finds an idle server if there is one, having a high d also means that when a job does find idle servers, it makes many copies that *only* add load to the system without providing any benefit. Hence deterministic slowdown represents the case in which we would expect redundancy to yield the worst performance as d increases.

Two surprising observations are evident in Figure 9. First, we see that the system is never unstable, even when $d = k$. Second, at low λ mean response time is higher when $d = k$ than when $d = 1$, but at high λ mean response time is actually *lower* when $d = k$ than when $d = 1$. We discuss the second observation below. In Section 7.1 we explain intuitively why the system does not become unstable, and we formalize the stability conditions under RIQ in Section 7.2.

In Figure 9(a) we see that at low arrival rate ($\lambda = 0.3$), when d is small mean response time decreases as a function of d; this is consistent with our results in Section 6. The minimum $\mathbf{E}[T]$ occurs around $d = 30$, above which $\mathbf{E}[T]$ increases with d. At $d = k = 1000$, $\mathbf{E}[T]$ is even higher than at $d = 1$. To understand why this happens, consider the two opposing effects of increasing d. As d increases, each job gets to poll more and more servers, increasing the probability that the job finds an idle server if there is one. On the other hand, when d is high a job that finds multiple idle



(a) $\lambda = 0.3$ (b) $\lambda = 0.7$

Figure 10: Mean response time under RIQ as a function of d when $k = 1000$, $X \sim$ Hyperexponential with $C_X^2 = 10$, $S \sim$ 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.

servers can block many subsequent arrivals from finding an idle server (the total arrival rate to the system is λk, so with high probability many new jobs will arrive while a single job occupies most of the servers). When λ is low, the benefit obtained from polling many servers is small, since most jobs find idle servers even when dispatched randomly ($d = 1$). Likewise, the pain caused to those jobs that arrive while a single job occupies most of the servers is large, since these jobs were likely to find idle servers when $d = 1$.

As λ gets higher, this trend reverses and $d = 1000$ becomes better than $d = 1$. This is because at high λ only a small fraction of jobs find idle servers when $d = 1$. Almost all jobs that find idle servers at $d = 1000$ benefit greatly because they would not have found idle servers at $d = 1$. Furthermore, these jobs do not tend to find very many idle servers: when $d = 1000$ and $\lambda = 0.7$, the expected number of copies per job is only 3.34. This means that the jobs that replicate themselves do not waste too much server capacity; the primary benefit of high d is in finding idle servers rather than in running many copies. The jobs that do not find idle servers at $d = 1000$ are not hurt because with high probability they would not have found idle servers at $d = 1$ anyway. Figure 9(b) illustrates this; when $\lambda = 0.7$ mean response time is lower at $d = 1000$ than at $d = 1$. We observe similar trends under the more realistic Dolly(1,12) server slowdown distribution with hyperexponential inherent job sizes (see Figure 10).

7.1 Intuition for why RIQ is Stable

In Figures 9 and 10 we see that even as d gets high, the system does not become unstable. This is surprising: one might think that as each job queries more servers and makes more copies, the added work causes the servers to be always busy, thereby driving queue lengths to infinity. Indeed, we see in Figure 10 that ρ does approach 1 as d approaches 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. We formalize this intuition in Section 7.2, in which we derive an upper bound on mean response time under RIQ.

7.2 Formal Upper Bound on Mean Response Time under RIQ

Consider a system with k servers (unlike in the analysis in Section 5, here we do not need to assume k is large). Our dispatching policy is RIQ, where d is anything (in particular d may be close to k). We allow S and X to follow any distribution with finite first and second moments. As defined in Section 3, 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 λ, where $G = R(1) = S \cdot X$, and where the vacation time is $R(1) + z$:*

$$\mathbf{E}[T]^{\mathrm{RIQ}} \leq \mathbf{E}[T]^{\mathrm{M/G/1/vacation}} = \mathbf{E}[T]^{\mathrm{M/G/1}} + \mathbf{E}[(R(1) + z)_e].$$

Proof. Consider the analysis of mean response time under RIQ presented in Section 5.2. The only part of the analysis that is not exact is the asymptotically independent idleness assumption (see Section 5.1). We will upper bound response time under RIQ by upper bounding each component of the analysis that uses the asymptotically independent idleness assumption.

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

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

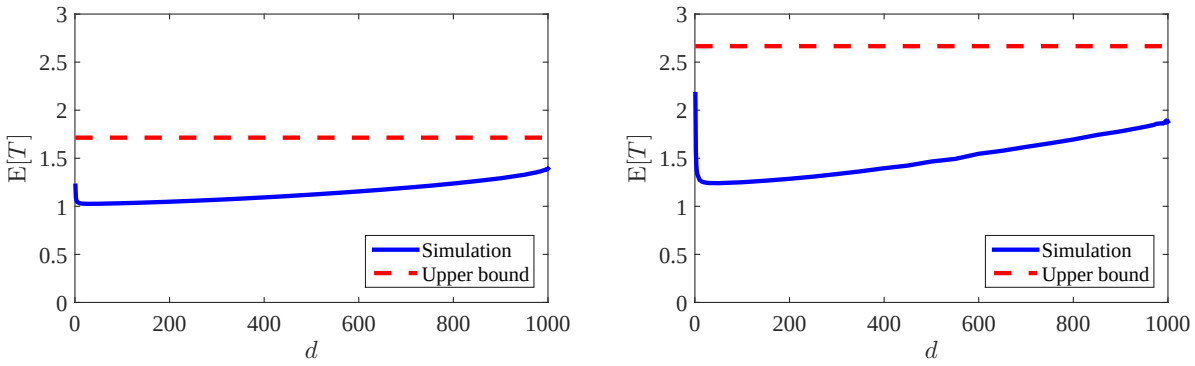
In the analysis in Section 5.2, we approximated $\mathbf{P}\{\text{job finds idle servers}\}$ using the asymptotically independent idleness. We can upper bound response time by instead assuming that no job ever finds an idle server:

$$\begin{aligned}\mathbf{E}[T] &\leq \mathbf{E}[T \mid \text{job finds no idle servers}] \\ &= \mathbf{E}[R(1)] + \mathbf{E}[T \mid \text{queueing}]^{\mathrm{M^*/G/1/efs}},\end{aligned} \tag{17}$$

where the second line follows from the same reasoning as in Section 5.2.

In Section 5.2, we further used the asymptotically independent idleness assumption in two places when analyzing the M*/G/1/efs. First, we said that when the 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, defined in (3). We will upper bound this by saying that the first job experiences runtime $R(1)$. This



(a) $\lambda = 0.3$ (b) $\lambda = 0.7$

Figure 11: Upper bound on mean response time under RIQ when $R = S \cdot X = 1$ is deterministic for both (a) low arrival rate ($\lambda = 0.3$) and (b) high arrival rate ($\lambda = 0.7$). 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.

is an upper bound because the first job actually experiences runtime $R(i) = X \cdot \min\{S_1, \ldots, S_i\} \leq X \cdot S = R(1)$. Note that since (17) forces all jobs to experience a non-zero queueing time, we can think of the first job in the busy period as being a "dummy" job rather than a real job that contributes to response time. Every time a server goes idle, we cause it to become busy working on a "dummy" job of size $R(1)$, so that 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, where the vacation time is $R(1)$. □

Figure 11 compares our upper bound to simulation results when $k = 1000$ and runtimes are deterministically equal to 1 and there is no cancellation cost ($z = 0$). The difference between the exact mean response time and the upper bound at $d = 1$ (which is equivalent to random dispatching; all jobs see an M/G/1 queue) is exactly the mean excess of $R(1)$. This indicates that RIQ can never perform more than an additive constant worse than random dispatching. We observe from simulation that RIQ actually does much better than at $d = 1$ for nearly all parameter settings.

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

Proof. The stability region for the M/G/1/vacation is the same as that for the M/G/1: the system is stable as long as $\lambda \cdot \mathbf{E}[R(1)] < 1$. 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 Conclusion

In this paper we introduced the $S\&X$ model, a new, more realistic model for computer systems with redundancy. The model is very general, allowing for any inherent job size distribution X, any server slowdown distribution S, and any cancellation time z. This model was motivated by

a common weakness of the existing theoretical work on redundancy: the assumption that a job's running times are independent across servers. This Independence model is unrealistic because in practice, each job consists of a certain inherent amount of work, X, that must be completed regardless of the particular server on which the job runs. In the Independence model, a job's running time becomes arbitrarily small as it runs on more and more servers; this cannot happen in practice.

In our new $S\&X$ model, the dispatching policies previously studied in the theory literature perform much worse than the existing theoretical analysis in the Independence model predicts. This motivates the development of new dispatching policies designed to perform well in the $S\&X$ model. We introduce the Redundant-to-Idle-Queue policy, under which each arriving job creates redundant copies only when the job finds idle servers on which to run these copies. We derive an approximate analysis of both the mean and the distribution 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 maximum number of copies per job 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 the redundancy degree d.

RIQ is able to achieve such good performance because it takes into account the system state when deciding whether to run redundant copies of a job. Awareness of system state is an important feature of many dispatching and scheduling policies used in redundancy systems in practice. Other system-aware redundancy policies proposed by practitioners include replicating only small jobs [1], delaying replication so as to only replicate jobs that are experiencing a long slowdown [3, 23], and reserving some servers on which to run replicas [16, 2]. We hope that the $S\&X$ model will provide a setting in which to analyze these policies in theory, leading to a better understanding of how to tune such policies to optimize performance.

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 jobs that are processed consecutively 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*, pages 185–198, April 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*, volume 10, page 24, 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. Technical report, 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 Journal on Selected Areas in Communications*, 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 Twenty-Fourth ACM Symposium on Operating Systems Principles*, pages 69–84. ACM, 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] 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.

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

[21] 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.

[22] 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.

[23] 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.