Reducing Latency via Redundant Requests:
Exact Analysis

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

Samuel Zbarsky
Dept. of Mathematics
Carnegie Mellon University
szbarsky@andrew.cmu.edu

Sherwin Doroudi
Tepper School of Business
Carnegie Mellon University
doroudi@andrew.cmu.edu

Mor Harchol-Balter
School of Computer Science
Carnegie Mellon University
harchol@cs.cmu.edu

Esa Hyytiä
Dept. of Communications and
Networking
Aalto University
esa.hyytia@aalto.fi

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 run a request on multiple servers and wait for the first completion (discarding all remaining copies of the request). However there is no exact analysis of systems with redundancy. This paper presents the first exact analysis of systems with redundancy. We allow for any number of classes of redundant requests, any number of classes of non-redundant requests, any degree of redundancy, and any number of heterogeneous servers. In all cases we derive the limiting distribution on the state of the system.

In small (two or three server) systems, we derive simple forms for the distribution of response time of both the redundant classes and non-redundant classes, and we quantify the “gain” to redundant classes and “pain” to non-redundant classes caused by redundancy. We find some surprising results. First, the response time of a fully redundant class follows a simple Exponential distribution and that of the non-redundant class follows a Generalized Hyperexponential. Second, fully redundant classes are “immune” to any pain caused by other classes becoming redundant.

We also compare redundancy with other approaches for reducing latency, such as optimal probabilistic splitting of a class among servers (Opt-Split) and Join-the-Shortest-Queue (JSQ) routing of a class. We find that, in many cases, redundancy outperforms JSQ and Opt-Split with respect to overall response time, making it an attractive solution.

Categories and Subject Descriptors

G.3 [Mathematics of Computing]: Probability and Statistics—Queueing theory; Markov processes

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from Permissions@acm.org.

SIGMETRICS’15, June 15 - 19, 2015, Portland, OR, USA.
Copyright © 2015 ACM 978-1-4503-3486-0/15/06  ...$15.00.
http://dx.doi.org/10.1145/2745844.2745873.
We then turn to the $M$ model (Figure 2(c)), where there is a “shared server,” which can be used by all request streams. We ask how best to use this shared resource.

After exploring these questions for small systems, we turn to issues of scale, investigating scaled versions of the $N, W$, and $M$ models. We use our exact closed form limiting distribution to derive the response time distribution for a fully redundant class in any size system.

The remainder of this paper is organized as follows. In Section 2, we describe related work and how it differs from the present work. In Section 3, we formalize our model and state our main theorem for the general system. In Sections 4, 5, and 6, we discuss detailed results for the $N, M$, and $W$ models. Section 7 addresses how these models scale as the number of servers increases. In Section 8, we conclude.

2. PRIOR WORK

In this section, we review several models that are related to redundant requests. All of these models differ from ours in critical ways that change both the mathematical techniques available to analyze the system, and the results obtained. Nonetheless, we hope that the results in this paper might shed some light on the problems below, many of which are notoriously difficult.

### Coupled Processor/Cycle Stealing

In a coupled processor system, there are two servers and two classes of jobs, $A$ and $B$. Server 1 works on class $A$ jobs in FCFS order, and server 2 works on class $B$ jobs in FCFS order. However, if there are only jobs of one class in the system, the servers “couple” to serve that class at a faster rate: unlike in the redundancy model, class $A$ jobs only get to use server 2 when the system is empty of $B$’s (and vice-versa). Generating functions for the stationary distribution of the queue lengths in a two-server system with Exponential service times were derived in [12, 23], but this required solving complicated boundary value problems and provided little intuition for the performance of the systems. The stationary distribution of the workload in the two-server system was derived in [10] using a similar approach. In [17], a power-series approach was used to numerically compute the queue-length stationary distribution in systems with more than two servers under Exponential service times. Much of the remaining work on coupled processor models involves deriving bounds and asymptotic results (for example, [6]).

In the donor-beneficiary model (one-way cycle stealing), only one class of jobs (the beneficiary) receives access to both servers, typically only when no jobs of the other class are present. In addition, if there is only one beneficiary job present, one server must idle (the servers do not “couple”). The donor-beneficiary model has been studied, in approximation, in a variety of settings [16, 26]. However, it differs from the redundancy model because a job is never in service at more than one server, and because donor jobs often have full preemptive priority at their server.

### Fork-Join

Another related model is the fork-join system, in which each job that enters a system with $k$ servers splits into $k$ pieces, one of which goes to each server. The job is considered complete only when all $k$ pieces have completed service. This is different from the redundancy model because only one redundant request needs to finish service in the redundancy
Figure 2: (a) The $\mathcal{N}$ model. Class $A$ jobs join the queue at server 2 only, while class $R$ jobs are redundant at both servers. (b) The $\mathcal{W}$ model. Class $A$ jobs join the queue at server 1 only, class $B$ jobs join the queue at server 2 only, and class $R$ jobs are redundant at both servers. (c) The $\mathcal{M}$ model. Class $R_1$ jobs are redundant at servers 1 and 2, and class $R_2$ jobs are redundant at servers 2 and 3.

model. Furthermore, a fork-join job sends work to all $k$ servers, whereas a redundant job of class $C_i$ only sends copies to the servers in $S_{C_i}$, where $S_{C_i}$ is a subset of the $k$ servers. The fork-join model is known to be very difficult to analyze. Many papers have derived bounds and approximations for such a system (for example, [3, 4, 22, 25, 35]). Exact analysis remains an open problem except for the two-server case [13, 14]; see [8] for a more detailed overview.

Flexible Server Systems
A third related model is the flexible server system, in which each class of jobs has its own queue, and each server can serve some subset of classes. The design and performance of flexible server systems has been studied in [5, 29, 30, 31]. In a flexible server system, traditionally, when a server becomes available, it chooses the queue from which to take its next job according to some policy. By contrast in redundancy systems, each server has its own FCFS queue and jobs are routed to a subset of servers upon arrival. The key difference between flexible server systems and redundancy systems is that redundant jobs may be served by multiple servers simultaneously, whereas in a flexible server system, each job may be processed by only one server.

A special case of the flexible server system uses the following policy. When a server becomes available, it chooses the job that arrived earliest from among the jobs it can serve. This policy is similar to the redundant system because each server works in FCFS order among the jobs it can serve. However, there are no redundant jobs in this flexible server system; jobs cannot be in service at two servers at once. For this model, under a specific routing assumption when an arriving job sees multiple idle servers, the stationary distribution that satisfies the balance equations is given [32, 1]. Our redundancy model requires no such routing assumption, because arriving redundant jobs enter service at all idle servers. Finally, mean response times are lower in a redundant system than in an FCFS flexible server system; our exact analysis allows us to quantify this performance gap.

Redundancy Models
Recently, in 2012, the $(n, k, r)$ system was proposed [27], where there are $n$ servers, and each job sends a request to $k \leq n$ of these servers. When $r \leq k$ requests complete, the job is considered finished. If we view the $k$ requests as $k$ "redundant" copies of a job, the problem can be seen as similar to ours, although in our model, jobs can only be redundant at specific subsets of servers. Various bounds and approximations have been derived for the $(n, k, r)$ model [19, 20, 27], and the optimal value of $k$ has been determined for different system loads and costs of deleting extra redundant requests [28]. Additionally, other variants have been proposed where a job might wait before issuing redundant requests [34]. Unfortunately, the only exact analysis of the $(n, k, r)$ system is for a highly simplified model in which each server is actually an $M/M/\infty$ queue, so there is no queueing [19].

In the special case $r = 1$, which mostly closely matches our model, approximations have been found assuming that the queues are independent [33]. The optimal degree of replication and service discipline also has been studied [24].

3. MODEL
We consider a system with $k$ servers, denoted $1, 2, \ldots, k$, and $\ell$ classes of jobs, denoted $C_1, C_2, \ldots, C_\ell$. (see Figure 1). The service time at server $j$ is distributed Exponentially with rate $\mu_j$ for all $1 \leq j \leq k$, and each server processes the jobs in its queue in FCFS order. Each class of jobs $C_i$ arrives to the system as a Poisson process with rate $\lambda_{C_i}$, and replicates itself by joining the queue at some fixed subset of the servers $S_{C_i} = \{j \mid \text{server } j \text{ can serve class } C_i\}$. Jobs in class $C_i$ cannot join the queue at any server $j \notin S_{C_i}$. A job may be in service at multiple servers at the same time; if a job is in service at both servers $i$ and $j$, it receives service at combined rate $\mu_i + \mu_j$.

Looking at Figure 1, it is difficult to figure out an appropriate state space. One might think that you could track the number of jobs of class at each queue, but this state space is missing information about which specific jobs are in multiple queues. Furthermore, servers are not independent, and job classes are not independent, so typical product-form type state spaces and solutions are unlikely to work.

The key insight that allows us to model this system is that we can view the system as having a single central queue in which all jobs wait in the order that they arrived (see Figure 3). Each server processes jobs from this central queue in FCFS order, skipping over those jobs it cannot serve. For example, in Figure 3, server 3 will skip over job $A^{(1)}$ and move to job $B^{(1)}$ when choosing its next job. We can write the state of the system as $(c_0, c_{n-1}, \ldots, c_1)$, where there are $n$ jobs in the system, and $c_i$ is the class of the $i^{th}$ job in this central queue; $c_1$ is the class of the job at the head of the queue, which is also in service at all servers in $S_{C_1}$.
Theorem 1. The limiting probability of being in state \((c_n, c_{n-1}, \ldots, c_1)\) is

\[ \pi(c_n, c_{n-1}, \ldots, c_1) = C \prod_{i=1}^{n} \frac{\lambda_i}{\mu_1^{c_1} \mu_2^{c_2} \cdots \mu_m^{c_m}}, \]

where \(C\) is a normalizing constant.

Proof. See Appendix. \(\square\)

Although \(\pi(c_n, c_{n-1}, \ldots, c_1)\) looks like a product-form solution, it is not; we cannot write the limiting probabilities as a product of independent marginal per-server terms, or as a product of independent marginal per-class terms. In fact, the form is quite unusual, although the form is quite unusual.

Example 1. Consider the system shown in Figure 3. Here, the current state is \((B, B, A)\), where the head of the queue is the right, job \(A^{(1)}\) is currently in service at server 1 and 2, and job \(B^{(1)}\) is currently in service at server 3. From Theorem 1, the limiting probability of this state is

\[ \pi(B, B, A) = C \left( \frac{\lambda_A}{\mu_1^2} \frac{\lambda_B}{\mu_1 + \mu_2 + \mu_3} \right)^2. \]

Corollary 1. The general system with redundant requests is stable when \(\forall C \subseteq \{C_1, \ldots, C_l\} \),

\[ \sum_{C \subseteq C} \lambda_C < \sum_{m \in C} \mu_m. \]

Proof. All limiting probabilities \(\pi(c_n, c_{n-1}, \ldots, c_1)\) from Theorem 1 are in \((0, 1)\) when \(\forall C \subseteq \{C_1, \ldots, C_l\}\), \(1\) holds. \(\square\)

In Sections 4, 5, and 6, we use the result of Theorem 1 to study the \(N, W,\) and \(M\) models, defined below.

\(N\) Model

The \(N\) model is the simplest non-trivial example of a redundancy system where there are both redundant and non-redundant classes. In an \(N\) model there are two servers running at rates \(\mu_1\) and \(\mu_2\) and two classes of jobs (see Figure 2(a)). Class \(A\) jobs are non-redundant; they arrive with rate \(\lambda_A\) and join the queue at server 2 only \((S_A = \{2\})\). Class \(R\) jobs are redundant; they arrive with rate \(\lambda_R\) and join the queue at both servers \((S_R = \{1, 2\})\).

\(W\) Model

Consider a two-server, two-class system in which each class of jobs has its own dedicated server (no redundancy). Now suppose that a third class of jobs enters the system and chooses to be redundant at both servers. The \(W\) model helps us understand how the presence of this redundant class affects the existing non-redundant classes. In a \(W\) model, there are two servers running at rates \(\mu_1\) and \(\mu_2\) and three classes of jobs (see Figure 2(b)). Class \(A\) jobs arrive with rate \(\lambda_A\) and are served at server 1 only \((S_A = \{1\})\), class \(B\) jobs arrive with rate \(\lambda_B\) and are served at server 2 only \((S_B = \{2\})\), and class \(R\) jobs arrive with rate \(\lambda_R\) and may be served at both servers \((S_R = \{1, 2\})\).

\(M\) Model

Again consider the two-server, two-class system in which each class of jobs has its own dedicated server. Suppose that a new server is added to the system and all jobs issue redundant requests at this server. The \(M\) model helps us understand how best to use the new server. In an \(M\) model, there are three servers with rates \(\mu_1, \mu_2,\) and \(\mu_3\) and two job classes (see Figure 2(c)). Class \(R_1\) jobs arrive with rate \(\lambda_{R_1}\) and join the queue at servers 1 and 2 \((S_{R_1} = \{1, 2\})\), and class \(R_2\) jobs arrive with rate \(\lambda_{R_2}\) and have \(S_{R_2} = \{2, 3\}\).

4. \(N\) Model

We first turn our attention to the \(N\) model (Figure 2(a)). An immediate consequence of Theorem 1 is Lemma 1, which gives the limiting distribution of the \(N\) model.

Lemma 1. In the \(N\) model, the limiting probability of being in state \((c_n, c_{n-1}, \ldots, c_1)\) is:

\[ \pi(c_n, c_{n-1}, \ldots, c_1) = C_N \left( \frac{\lambda_A}{\mu_2} \right)^{a_0} \left( \frac{\lambda_R}{\mu_1 + \mu_2} \right)^r \left( \frac{\lambda_A}{\mu_1 + \mu_2} \right)^{a_1}, \]

where \(a_0\) is the number of class \(A\) jobs before the first class \(R\) job, \(a_1\) is the number of class \(A\) jobs after the first class \(R\) job, \(r\) is the total number of class \(R\) jobs in the queue, and

\[ C_N = \left( \frac{\mu_2}{\mu_1 + \mu_2 - \lambda_A} \right)^{a_1} \]

is a normalizing constant.

We use this result to find (Theorem 2) that for the redundant class (class \(R\)), response time is Exponentially distributed, which is pleasantly surprising because the system is not an \(M/M/1\). Specifically, the distribution of response time is the same as that in an \(M/M/1\), where the arrival rate is \(\lambda_R\) and the service rate is \(\mu' = \mu_1 + \mu_2 - \lambda_A\). Note that \(\mu'\) can be viewed as giving the \(R\) jobs the full \(\mu_1\), and the portion of \(\mu_2\) that is not appropriated for the class \(A\) jobs \((\mu_2 - \lambda_A)\). Equivalently, this is the response time in an \(M/M/1\) with arrival rate \(\lambda_A + \lambda_R\) and service rate \(\mu_1 + \mu_2\).

Theorem 2. In the \(N\) model,

1. The number of class \(R\) jobs in the system, \(N_R\), is distributed Geometric \((1 - \rho)\), where \(\rho = \frac{\lambda_R}{\mu_1 + \mu_2 - \lambda_A}\), or

\[ \frac{1}{1 - \rho} \],

This is counterintuitive because as we will see in Lemma 3, the distribution of response time for class \(R\) does not depend on whether class \(A\) is redundant or non-redundant.
2. The response time of class $R$ jobs, $T_R$, is distributed
$\text{Exp}(\mu_1 + \mu_2 - \lambda_A - \lambda_R)$.

Proof. This is a special case of the more general result in
Theorem 10, which is proved in Appendix B. □

In Theorem 3, we find that the response time for the non-
redundant class, $T_A$, follows a Generalized Hyperexponential
distribution\(^2\). We can view the mean response time of class
$A$ jobs as that of an $M/M/1$ with arrival rate $\lambda_A$ and service
rate $\mu_2$, plus a penalty term that captures the extent to
which the redundant jobs hurt the $A$’s (Equation 3).

**Theorem 3.** In the $N$ model,

1. The number of class $A$ jobs in the system, $N_A$, has p.m.f.
$$\Pr\{N_A = n_A\} = \zeta_{n_1} \left(\frac{\lambda_A}{\mu_1}\right)^{n_A} + \zeta_{n_2} \left(\frac{\lambda_A}{\mu_1 + \mu_2 - \lambda_R}\right)^{n_A}$$
   where
   $$\zeta_{n_1} = C_0 \left(\frac{\mu_1}{\mu_1 - \lambda_R}\right),$$
   $$\zeta_{n_2} = C_0 \left(\frac{\mu_1}{\mu_1 + \mu_2 - \lambda_R} - \frac{\lambda_R}{\mu_1 - \lambda_R}\right),$$
   and $C_0$ is as in Lemma 1.

2. The distribution of response time of class $A$ jobs

   $$T_A \sim H_2(\nu_{n_1}, \nu_{n_2}, \omega_n),$$

   where
   $$\nu_{n_1} = \mu_2 - \lambda_A$$
   $$\nu_{n_2} = \mu_1 + \mu_2 - \lambda_A - \lambda_R$$
   $$\nu_{n_3} = \mu_1 + \mu_2 - \lambda_A$$
   $$\omega_n = \frac{\lambda_R\nu_{n_1}}{(\mu_1 - \lambda_R)\nu_{n_3}}.$$ 

   The expected response time of class $A$ jobs is
   $$\mathbb{E}[T_A] = \frac{1}{\nu_{n_1}} + \frac{1}{\nu_{n_2}} - \frac{1}{\nu_{n_3}}.$$ (3)

Proof. Deferred to the end of the section. □

Figure 4 compares mean response time before class $R$
jobs become redundant (each class sees its own independent
$M/M/1$), and after class $R$ jobs become redundant. We hold
$\mu_1 = \mu_2 = 1$ and vary the load by increasing $\lambda_R = \lambda_A$.
We find redundancy helps class $R$ jobs by a factor of two (Figure
4(a)), but can hurt class $A$ by up to 50% (Figure 4(b)).

In Lemma 3, we ask what happens if class $A$ jobs decide they
too should be redundant. That is, all jobs can be served
at both servers - the system is fully redundant. This trans-
forms the system into an $M/M/1$ with arrival rate $\lambda_A + \lambda_R$
and service rate $\mu_1 + \mu_2$ (Lemma 2). Surprisingly, class $R$
is immune to pain when class $A$ also becomes redundant: as
Lemma 3 shows, the distribution of response time for class
$R$ is the same before and after class $A$ becomes redundant.
Of course, when the $A$’s become redundant, they receive the
benefit of having two servers.

\(^2\)A Generalized Hyperexponential, $H_2(\nu_1, \nu_2, \omega)$ is defined as
the weighted mixture of two Exponentials with rates $\nu_1$ and
$\nu_2$, where the first Exponential is given weight $1 + \omega$ and
the second is given weight $-\omega$. Note that $\omega$ can be any real
number; it need not be a probability [7].

**Figure 4:** Comparing mean response time before and
after class $R$ becomes redundant when $\mu_1 = \mu_2 = 1$
and $\lambda_A = \lambda_R$ for (a) class $R$, and (b) class $A$. The
mean response time for the overall system is the weighted average of these two classes.

**Lemma 2.** The fully redundant system, in which all jobs
issue redundant requests at both servers, is equivalent to an
$M/M/1$ with arrival rate $\lambda_A + \lambda_R$ and service rate $\mu_1 + \mu_2$.

Proof. Deferred to the end of the section. □

**Lemma 3.** With respect to response time, both classes of
jobs do at least as well in the fully redundant model as in the
$N$ model. In particular,

1. $\mathbb{E}[T_A]$ Fully Redundant $\leq \mathbb{E}[T_A]$ Redundant

2. $T_R$ Fully Redundant $\leq T_R$ Redundant.

Proof. Deferred to the end of the section. □

Going back to the $N$ model with only one redundant class,
redundancy clearly helps the redundant class considerably.
But there are alternative latency-reducing strategies. For
example, each redundant class could optimally probabilisti-
cally split jobs among all allowable servers (Opt-Split), or
join the shortest queue among allowable servers (JSQ).

In Figure 5, we compare these other options for the $N$
model, where the mean response times under Opt-Split are
derived analytically (Definition 1), but JSQ is simulated. We
find that, for the redundant class $R$, redundancy beats JSQ,
which beats Opt-Split. Redundancy often is not much better
than JSQ, yet they can differ by a factor of 2, depending on
the load of class $R$ and the relative server speeds.

Surprisingly, the non-redundant class $A$ often prefers re-
dundancy of the other class to Opt-Split or JSQ. This is
because the non-redundant class wants the redundant class
to spend as little time as possible blocking the $A$’s jobs on
server $2$, and redundancy helps with this.

Note that under Opt-Split we see an inflection point in
mean response time for both class $R$ and class $A$. For ex-
ample, in Figures 5(a) and (b), there is an inflection point
at $\lambda_R = 0.6$, when $\lambda_R = \lambda_A$. This phase change occurs be-
cause when $\lambda_R < \lambda_A$, no class $R$ jobs go to server 2 under
Opt-Split, but when $\lambda_R > \lambda_A$ the $R$’s compete with the $A$’s.
Also observe that $\mathbb{E}[T]$ is not monotonically increasing; this is
because as $\lambda_R$ increases, the redundant class contributes more
to the weighted average.

From the overall system’s perspective, redundancy is al-
ways preferable to Opt-Split and JSQ because it optimizes
overall server utilization.

When $\mu_1 = \mu_2$, even when non-redundant jobs prefer Opt-
Split, redundancy is never more than 50% worse than Opt-
Split for the non-redundant jobs (Theorem 4).
Figure 5: Comparing redundancy (solid blue), Opt-Split (dashed green), and JSQ (dashed red) for the $N$ model as $\lambda_R$ increases with $\lambda_A = 0.6$. We plot mean response time for the redundant $R$ class (left column), the non-redundant $A$ class (middle column), and the overall system (right column). Rows represent different ratios of server speeds.

Definition 1. Under Opt-Split, $p$ fraction of class $R$ jobs go to server 2, and $1-p$ fraction go to server 1, where $p$ is chosen to minimize $E[T]$. The mean response times for class $R$ jobs, class $A$ jobs, and the system are respectively:

$$E[T_R]^{\text{Opt-Split}} = \frac{1-p}{\mu_1 - (1-p)\lambda_R} + \frac{p}{\mu_2 - \lambda_A - p\lambda_R}$$

$$E[T_A]^{\text{Opt-Split}} = \frac{1}{\mu_2 - \lambda_A - p\lambda_R}$$

$$E[T]^{\text{Opt-Split}} = \frac{\lambda_A}{\lambda_A + \lambda_R} E[T_A]^{\text{Opt-Split}} + \frac{\lambda_R}{\lambda_A + \lambda_R} E[T_R]^{\text{Opt-Split}}.$$

Theorem 4. If $\mu_1 = \mu_2$, then the following are true:

1. $\frac{1}{2} \leq \frac{E[T_R]^{\text{Redundant}}}{E[T_R]^{\text{Opt-Split}}} \leq 1$.

2. $\frac{E[T_A]^{\text{Redundant}}}{E[T_A]^{\text{Opt-Split}}} \leq \frac{3}{2}$.

3. $\frac{1}{2} \leq \frac{E[T]^{\text{Redundant}}}{E[T]^{\text{Opt-Split}}} \leq 1$.

Proof. Deferred to the end of the section.

4.1 Proofs for $N$ Model

Proof. [Theorem 3] We first consider the case $n_A = 0$.

$$\Pr\{N_A = 0\} = C \sum_{i=0}^{\infty} \left(\frac{\lambda_R}{\mu_1 + \mu_2}\right)^i = C \frac{\mu_1 + \mu_2}{\mu_1 + \mu_2 - \lambda_R}.$$  

For $n_A > 0$, we compute $\Pr\{N_A = n_A\}$ by summing over all values of $r_i$ and all values of $a_0$ and $a_1$ so that $a_0 + a_1 = n_A$:

$$\Pr\{N_A = n_A\} = C \beta_A^n + \sum_{a_0=0}^{n_A} \sum_{r=1}^{\infty} C \beta_A^{a_0} \alpha_R^{a_1-r} \left(\frac{n_A-a_0+r-1}{n_A-a_0}\right),$$

where $\beta_A = \frac{\lambda_A}{\mu_1 + \mu_2}$, $\alpha_A = \frac{\lambda_A}{\mu_2}$, and $\alpha_R = \frac{\lambda_R}{\mu_1 + \mu_2}$, which after some simplification yields the result in (2). We omit the details due to space considerations; for the full proof, see the associated technical report [15].

We now obtain the Laplace transform of the response time for class $A$ jobs, $T_A(s)$, via distributional Little’s Law [21]. First, we find the $z$-transform of the number of class $A$ jobs in the system, $\tilde{N}_A(z)$:

$$\tilde{N}_A(z) = \sum_{i=0}^{\infty} \Pr\{N_A = i\} z^i$$
service rate

Also observe that class A jobs depart the system in the same order in which they arrive, so \( \bar{N}_A \), the number of class A arrivals during a class A response time, is equivalent to \( N_A \), the number of class A jobs seen by an A departure. Then since \( \bar{T}_A = \bar{T}_A(\lambda_A - \lambda_A z) \), we have:

\[
\bar{T}_A(\lambda_A - \lambda_A z) = \bar{N}_A(z)
\]

\[
= \zeta_{\lambda_1, \mu_2} \mu_2 - \lambda_A z + \zeta_{\lambda_2}(\mu_1 + \mu_2 - \lambda_R) \mu_1 + \mu_2 - \lambda_R - \lambda_A z.
\]

Let \( s = \lambda_A - \lambda_A z \), so \( z = 1 - \frac{s}{\lambda_A} \). Then we have:

\[
\bar{T}_A(s) = \frac{\zeta_{\lambda_1, \mu_2}}{\mu_2 - \lambda_A z} + \zeta_{\lambda_2}(\mu_1 + \mu_2 - \lambda_R) \mu_1 + \mu_2 - \lambda_R - \lambda_A(z - \frac{s}{\lambda_A})
\]

\[
= (1 + \omega_s) \frac{\nu_{\lambda_1}}{\nu_{\lambda_1} + s} - \omega_s \frac{\nu_{\lambda_2}}{\nu_{\lambda_2} + s}.
\]

This is the transform of a Generalized Hyperexponential distribution, \( H_2(\nu_{\lambda_1}, \nu_{\lambda_2}, \omega_s) \). Finally,

\[
E[T_A] = -\bar{T}_A(s)|_{s=0}
\]

\[
= -(1 + \omega_s) \left( \frac{-\nu_{\lambda_1}}{(\nu_{\lambda_1} + s)^2} + \omega_s \frac{-\nu_{\lambda_2}}{(\nu_{\lambda_2} + s)^2} \right) |_{s=0}
\]

\[
= \frac{1}{\nu_{\lambda_1}^2} + \frac{1}{\nu_{\lambda_2}^2} - \frac{1}{\nu_{\lambda_3}^2}.
\]

**Proof.** [Lemma 2] In the fully redundant model, all jobs enter the FCFS queue at both servers and depart from both servers immediately upon completion at either server. This is exactly an M/M/1 with arrival rate \( \lambda_A + \lambda_B \) and service rate \( \mu_1 + \mu_2 \).

**Proof.** [Lemma 3] From Theorem 2, in the N model, \( T_R \sim \text{Exp}(\mu_1 + \mu_2 - \lambda_A - \lambda_B) \), which is the response time distribution in an M/N/1 arrival rate \( \lambda_A + \mu_R \) and service rate \( \mu_2 + \mu_2 \). From Theorem 3, in the N model, \( E[T_A] \) is given by

\[
E[T_A] = \frac{1}{\mu_1 + \mu_2 - \lambda_A - \lambda_B} + \frac{1}{\mu_1 + \mu_2 - \lambda_A - \lambda_B} - \frac{1}{\mu_1 + \mu_2 - \lambda_A - \lambda_B}.
\]

which is at least the mean response time in an M/M/1 with arrival rate \( \lambda_A + \lambda_B \) and service rate \( \mu_1 + \mu_2 \) since \( \frac{1}{\mu_2 - \lambda_A} - \frac{1}{\mu_2 - \lambda_A} = \frac{1}{\mu_2 - \lambda_A} + \frac{1}{\mu_2 - \lambda_A} = \frac{1}{\mu_2 - \lambda_A} \).

**Proof.** [Theorem 4] Definition 1 gives us \( E[T_R^{\text{Opt-Split}}, E[T_A^{\text{Opt-Split}} \text{, and } E[T]^{\text{Opt-Split}} \text{. We know } E[T_A]^{\text{Redundant}} \text{ from Theorem 3.} \text{ } Theorem 2 tells us that } T_R^{\text{Redundant}} \sim \text{Exp}(\mu_1 + \mu_2 - \lambda_A - \lambda_B), \text{ so we know that } E[T_R]^{\text{Redundant}} = \frac{\lambda_R}{\mu_1 + \mu_2 - \lambda_A - \lambda_B} \lambda_A + \lambda_B. \text{ Finally, } \]

\[
E[T]^{\text{Redundant}} = \frac{\lambda_R}{\lambda_A + \lambda_R} E[T_R]^{\text{Redundant}} + \frac{\lambda_A}{\lambda_A + \lambda_R} E[T_A]^{\text{Redundant}}
\]

Thus, \( E[T]^{\text{Redundant}}, E[T_A]^{\text{Redundant}} \), and \( E[T_R]^{\text{Redundant}} \), and the desired results follow after some minor algebra.

**5. W MODEL**

We now consider the W model (see Figure 2(b)). The W model has two non-redundant classes, A and B, each with its own server. A third class, R, enters the system and issues redundant requests at both servers. We study how this redundant class affects the performance of the system.

An immediate consequence of Theorem 1 is Lemma 4, which gives the limiting distribution of the W model.

**Lemma 4.** In the W model, the limiting probability of being in state \( (c_n, c_{n-1}, \ldots, c_1) \) depends on \( c_1 \), as follows:

\[
\pi(c_n, \ldots, c_1) = \frac{R(\lambda_A - \lambda_A c_1)}{\mu_1 + \mu_2 - \lambda_A - \lambda_B}.
\]

where \( c_0 \) is the number of class A jobs before the first class B or R job, \( b_0 \) is the number of class B jobs before the first class A or R job, \( a_1 \) (respectively, \( b_1 \)) is the number of class A (class B) jobs after the first job of class R or B (A), \( r \) is the total number of class R jobs, and

\[
C_W = \frac{(\lambda_1 - \lambda_A)(\mu_1 + \mu_2 - \lambda_A - \lambda_B)}{\mu_1 \mu_2 (\mu_1 + \mu_2 - \lambda_A - \lambda_B)} + \frac{(\lambda_2 - \lambda_B)(\mu_1 + \mu_2 - \lambda_A - \lambda_B)}{\mu_1 \mu_2 (\mu_1 + \mu_2 - \lambda_A - \lambda_B)}
\]

is a normalizing constant.

Like in the N model, the redundant class (class R) has an Exponentially distributed response time (Theorem 5). This is again surprising because the system is not an M/M/1. Nonetheless, the response time for the redundant class is stochastically equivalent to the response time in an M/M/1 with arrival rate \( \lambda_R \) and service rate \( \mu' = \mu_1 + \mu_2 - \lambda_A - \lambda_B \). We can interpret \( \mu' \) as the remaining service capacity in the system after \( \lambda_A \) and \( \lambda_B \) have been apportioned to classes A and B respectively. Alternatively, we can view the response time for the redundant class as that in an M/M/1 with arrival rate \( \lambda_A + \lambda_B + \lambda_B \) and service rate \( \mu_1 + \mu_2 \).

**Theorem 5.** In the W model,

1. The number of class R jobs in the system, \( N_R \), is distributed Geometric(1−\( \rho \))−1, where \( \rho = \frac{\lambda_1 + \lambda_2}{\mu_1 + \mu_2 - \lambda_A - \lambda_B} \).

2. The response time of class R jobs, \( T_R \), is distributed

\[
\text{Exp}(\mu_1 + \mu_2 - \lambda_A - \lambda_B).
\]

**Proof.** The proof follows the same approach as that of Theorem 10, and is omitted.

In Theorem 6, we derive the distribution of response time for the non-redundant class A (class B is symmetric). Again like in the N model, we find that \( T_A \) follows a Generalized Hyperexponential distribution. In addition, the mean response time of class A (or class B) jobs can be interpreted as that in an M/M/1 with arrival rate \( \lambda_A \) and service rate \( \mu_1 \), plus a penalty term that captures the extent to which the redundant class hurts the A’s (or B’s) (Equation 4). Surprisingly, this penalty is the same for class A and class B even if they have different loads: the pain caused by the redundant class is shared equally among the non-redundant classes.

**Theorem 6.** In the W model,
We now ask what would happen if class dant jobs compete for service with the non-redundant jobs. Outperforms Opt-Split (Figure 6(a)).

Figure 6: Comparing redundancy (solid blue), Opt-Split (dashed green), and JSQ (dashed red) for the \( W \) model as \( \lambda_R \) increases, for \( \mu_1 = \mu_2 = 1, \lambda_A = 0.6, \) and \( \lambda_B = 0.4. \) Lines shown include mean response time for (a) class \( R \), (b) class \( A \), and (c) the system. Results for other values of \( \mu_1 \) and \( \mu_2 \) are similar.

1. The number of class \( A \) jobs in the system, \( N_A, \) has p.m.f.

\[
\Pr\{N_A = n_A\} = \zeta_{W1}\left(\frac{\lambda_A}{\mu_1}\right)^{n_A} + \zeta_{W2}\left(\frac{\lambda_A}{\mu_1 + \mu_2 - \lambda_A - \lambda_B - \lambda_R}\right)^{n_A},
\]

where

\[
\zeta_{W1} = \frac{(\mu_1 - \lambda_A)(\mu_2 - \lambda_A)(\mu_1 + \mu_2 - \lambda_A - \lambda_B - \lambda_R)}{\mu_1(\mu_2 - \lambda_B - \lambda_R)(\mu_1 + \mu_2 - \lambda_A - \lambda_B - \lambda_R)}
\]

\[
\zeta_{W2} = -\frac{\lambda_R(\mu_1 - \lambda_A)(\mu_1 + \mu_2 - \lambda_A - \lambda_B - \lambda_R)}{(\mu_2 - \lambda_B - \lambda_R)(\mu_1 + \mu_2 - \lambda_A - \lambda_B - \lambda_R)}.
\]

2. The distribution of response time class \( A \) jobs is

\[ T_A \sim H_2(\nu_{W1}, \nu_{W2}, \omega_W), \]

where

\[ \nu_{W1} = \mu_1 - \lambda_A, \]

\[ \nu_{W2} = \mu_1 + \mu_2 - \lambda_A - \lambda_B - \lambda_R, \]

\[ \omega_W = \frac{(\mu_2 - \lambda_B)\nu_{W2}}{(\mu_2 - \lambda_B - \lambda_R)\nu_{W3}}, \]

The mean response time of class \( A \) jobs is

\[
E[T_A] = \frac{1}{\nu_{W1}} + \frac{1}{\nu_{W2}} - \frac{1}{\nu_{W3}}
\]

where

\[ \nu_{W3} = \mu_1 + \mu_2 - \lambda_A - \lambda_B. \]

Proof. The proof follows the same approach as that of Theorem 3, and is omitted.

The introduction of a new redundant class clearly hurts the existing non-redundant classes, because the new redundant jobs compete for service with the non-redundant jobs. We now ask what would happen if class \( R \) chose which queue(s) to join according to some alternative policy, for example, Opt-Split or JSQ.

In Figure 6, we compare these options, where the mean response time under Opt-Split is derived analytically (Definition 2), but JSQ is simulated. We find that for the redundant class, redundancy outperforms JSQ, which in turn outperforms Opt-Split (Figure 6(a)).

For the non-redundant classes (Figure 6(b)), mean response time is often lower under redundancy than under Opt-Split or JSQ, particularly under higher loads of redundant jobs. This is because even though a larger number of \( R \) jobs compete with class \( A \) at server 1, some of these \( R \) jobs depart the system without ever using server 1 (they complete service at server 2 before entering service at server 1), and some of these \( R \) jobs receive service on both servers at once, thus departing the system faster. As in the \( N \) model, redundancy is always better for the overall system (Figure 6(c)).

When the servers are homogeneous, in the few cases in which mean response time of class \( A \) or \( B \) is lower under Opt-Split than under redundancy, we show that redundancy is never more than 50% worse for the \( A \) or \( B \) jobs.

**Definition 2.** Under Opt-Split, \( p \) fraction of class \( R \) jobs go to server 1, and \( 1 - p \) fraction go to server 2, where \( p \) is chosen to minimize \( E[T] \). The mean response times for class \( R \) jobs, class \( A \) jobs, and the overall system are:

\[
E[T_R]_{\text{Opt-Split}} = \frac{p}{\mu_1 - \lambda_A - p \lambda_R} + \frac{1 - p}{\mu_2 - \lambda_B - (1 - p) \lambda_R},
\]

\[
E[T_A]_{\text{Opt-Split}} = \frac{\mu_1 - \lambda_A - p \lambda_R}{\mu_1 - \lambda_A - p \lambda_R},
\]

\[
E[T]_{\text{Opt-Split}} = \frac{\lambda_A}{\lambda_A + \lambda_B + \lambda_R} E[T_R]_{\text{Opt-Split}} + \frac{\lambda_B}{\lambda_A + \lambda_B + \lambda_R} E[T_A]_{\text{Opt-Split}} + \frac{\lambda_R}{\lambda_A + \lambda_B + \lambda_R} E[T_R]_{\text{Opt-Split}}.
\]

The mean response time for class \( B \) is symmetric to that of class \( A \).

**Theorem 7.** If \( \mu_1 = \mu_2 \), then the following are true:

1. \( \frac{1}{2} \leq \frac{E[T_R]_{\text{Redundant}}}{E[T_R]_{\text{Opt-Split}}} \leq 1. \)

If \( \lambda_R \geq |\lambda_A - \lambda_B| \), then \( \frac{E[T_R]_{\text{Redundant}}}{E[T_R]_{\text{Opt-Split}}} = \frac{1}{2}. \)

2. \( \frac{1}{2} \leq \frac{E[T_A]_{\text{Redundant}}}{E[T_A]_{\text{Opt-Split}}} \leq 1. \)

3. \( \frac{1}{2} \leq \frac{E[T]_{\text{Redundant}}}{E[T]_{\text{Opt-Split}}} \leq 1. \)

Proof. We have \( E[T_R]_{\text{Opt-Split}}, E[T_A]_{\text{Opt-Split}}, \) and \( E[T]_{\text{Opt-Split}} \) from Definition 2. We also know \( E[T]_{\text{Redundant}} \) from Theorem 6. Theorem 5 tells us that \( T_R_{\text{Redundant}} \sim \exp(\mu_1 + \mu_2 - \lambda_A - \lambda_B - \lambda_R), \)

The mean response time for class \( B \) is symmetric to that of class \( A \).
\[ \mathbb{E}[T_R^{\text{Redundant}}] = \frac{1}{\mu_1 + \mu_2 - \lambda_A - \lambda_B - \lambda_R}. \]

Finally,
\[ \mathbb{E}[T]^{\text{Redundant}} = \frac{\lambda_R}{\lambda_A + \lambda_B + \lambda_R} \mathbb{E}[T_A^{\text{Redundant}}] + \frac{\lambda_A}{\lambda_A + \lambda_B + \lambda_R} \mathbb{E}[T_B^{\text{Redundant}}]. \]

We use these expressions to find \( \mathbb{E}[T]^{\text{Redundant}} \), and the desired results follow after some minor algebra. \( \square \)

6. \( \mathbb{M} \) MODEL

Finally, we consider the \( \mathbb{M} \) model (Figure 2(c)). Unlike the \( \mathbb{N} \) and \( \mathbb{W} \) models, there are two redundant classes in an \( \mathbb{M} \) model, classes \( R_1 \) and \( R_2 \). We study how to best use a shared server to which both classes issue redundant requests. For convenience, throughout the remainder of this section we use the notation
\[ \mu_{1,2,3} = \mu_1 + \mu_2 + \mu_3 \]
\[ \mu_{1,2} = \mu_1 + \mu_2 \]
\[ \mu_{2,3} = \mu_2 + \mu_3. \]

An immediate consequence of Theorem 1 is Lemma 5, which gives the limiting distribution of the \( \mathbb{M} \) model.

**Lemma 5.** In the \( \mathbb{M} \) model, the limiting probability of being in state \( (c_n, c_{n-1}, \ldots, c_1) \) depends on \( c_1 \), as follows:
\[
\begin{align*}
\pi(c_n, \ldots, R_1) &= C_M \left( \frac{\lambda_{R_1}}{\mu_{1,2,3}} \right)^{r_{1,0}} \left( \frac{\lambda_{R_2}}{\mu_{1,2,3}} \right)^{r_{1,1}} \left( \frac{\lambda_{R_2}}{\mu_{1,2,3}} \right)^{r_{2,1}}, \\
\pi(c_n, \ldots, R_2) &= C_M \left( \frac{\lambda_{R_2}}{\mu_{2,3}} \right)^{r_{2,0}} \left( \frac{\lambda_{R_1}}{\mu_{1,2,3}} \right)^{r_{1,1}} \left( \frac{\lambda_{R_2}}{\mu_{1,2,3}} \right)^{r_{2,1}},
\end{align*}
\]

where \( r_{1,0} \) (respectively, \( r_{2,0} \)) is the number of class \( R_1 \) (\( R_2 \)) jobs before the first class \( R_2 \) (\( R_1 \)) job, \( r_{1,1} \) (respectively, \( r_{2,1} \)) is the number of class \( R_1 \) (\( R_2 \)) jobs after the first class \( R_1 \) (\( R_2 \)) job, and
\[ C_M = \frac{(\mu_{1,2} - \lambda_{R_1})(\mu_{1,2,3} - \lambda_{R_1} - \lambda_{R_2})(\mu_{2,3} - \lambda_{R_2})}{\mu_{1,2,3}^2\mu_{1,2,3} - \lambda_{R_1} - \lambda_{R_2} - \lambda_{R_3} + \lambda_{R_1}\lambda_{R_2}\mu_2} \]
is a normalizing constant.

In Theorem 8, we derive the distribution of response time for class \( R_1 \) (class \( R_2 \) is symmetric). The response time for class \( R_1 \) follows a Generalized Hyperexponential distribution.

Note that in the \( \mathbb{N} \) and \( \mathbb{W} \) models, the redundant class had Exponentially distributed response time and the response time distribution for non-redundant classes was a Generalized Hyperexponential, whereas in the \( \mathbb{M} \) model, the redundant class has a Generalized Hyperexponential response time distribution. We hypothesize that the response time distribution is related to the degree of redundancy: fully redundant classes see Exponentially distributed response time, and partially redundant or non-redundant classes see Generalized Hyperexponentially distributed response times.

**Theorem 8.** In the \( \mathbb{M} \) model,

1. The number of class \( R_1 \) jobs, \( N_{R_1} \), has p.m.f.
\[
\Pr\{N_{R_1} = n\} = \zeta_M \left( \frac{\lambda_{R_1}}{\mu_{1,2}} \right)^n + \zeta_M \left( \frac{\lambda_{R_1}}{\mu_{1,2,3} - \lambda_{R_2}} \right)^n,
\]

where
\[ \zeta_M = C_M \frac{\mu_3}{\mu_3 - \lambda_{R_2}} \]

\[ \zeta_M = C_M \left( \frac{\lambda_{R_2}}{\mu_{1,2,3} - \lambda_{R_2}} - \frac{\lambda_{R_1}}{\mu_{1,2,3} - \lambda_{R_2}} \right). \]

2. The distribution of response time of class \( R_1 \) jobs is
\[
T_{R_1} \sim H_2(\nu_{M1}, \nu_{M2}, \omega_M),
\]

where
\[ \nu_{M1} = \mu_{1,2} - \lambda_{R_1} \]
\[ \nu_{M2} = \mu_{1,2,3} - \lambda_{R_1} - \lambda_{R_2} \]
\[ \omega_M = \zeta_M \frac{\mu_{1,2}}{\mu_{1,2} - \lambda_{R_1}} \]

**Proof.** The proof follows the same approach as that of Theorem 3, and is omitted. \( \square \)

Both classes obviously benefit from issuing redundant requests on a shared server rather than each class having a single dedicated server. However, one might wonder whether mean response time could be further reduced by using some other policy, like Opt-Split or JSQ, instead of redundancy. In Figure 7 we investigate the relative performance of these alternative policies. Mean response time under Opt-Split is derived analytically (Definition 3); JSQ is simulated.

**Definition 3.** Under Opt-Split, \( p \) fraction of class \( R_1 \) jobs go to server 2, and \( 1 - p \) fraction go to server 1, and \( q \) fraction of class \( R_2 \) jobs go to server 3, and \( 1 - q \) fraction go to server 3. We choose \( p \) and \( q \) to minimize the overall mean response time, given by
\[
\mathbb{E}[T]^{\text{Opt-Split}} = \frac{(1 - p)\lambda_{R_1}}{\lambda_{R_1} + \lambda_{R_2}} + \frac{1}{\mu_1 - (1 - p)\lambda_{R_1}} + \frac{p\lambda_{R_1} + q\lambda_{R_2}}{\lambda_{R_1} + \lambda_{R_2}} + \frac{1}{\mu_2 - p\lambda_{R_1} - q\lambda_{R_2}} + \frac{(1 - q)\lambda_{R_2}}{\lambda_{R_1} + \lambda_{R_2}} + \frac{1}{\mu_3 - (1 - q)\lambda_{R_2}}.
\]

In all cases, redundancy outperforms both Opt-Split and JSQ. For homogeneous servers (Figure 7(a)), mean response time under JSQ approaches that under redundancy at high load, but at low load, redundancy is better by a factor of 2. For heterogeneous servers (Figure 7(b)), as the service rate of the shared server increases, mean response time under Opt-Split approaches that under redundancy (Theorem 9), but JSQ is worse by a factor of 2. As the system is symmetric, the response times of the individual classes are the same as that of the overall system, and thus are not shown.

We analytically prove performance bounds for the \( \mathbb{M} \) model:

**Theorem 9.** In the \( \mathbb{M} \) model, for any \( \mu_1, \mu_2, \mu_3, \lambda_{R_1}, \lambda_{R_2} \), and \( \lambda_{R_3} \), such that the system is stable,
1. If \( \mu_1 = \mu_2 = \mu_3 \) and \( \lambda_{R_1} = \lambda_{R_2} \), \( \frac{1}{3} \leq \frac{\mathbb{E}[T]^{\text{Redundant}}}{\mathbb{E}[T]^{\text{Opt-Split}}} \leq \frac{1}{2} \).
In the scaled model, there are $k$ classes and $k$ servers. The scaled $N$ model is similar; there are $k$ servers and $k$ classes of jobs (see Figure 8(a)). Class $R$ jobs replicate at all servers, while jobs from class $C_i$ join only the queue at server $i$ for $2 \leq i \leq k$. The scaled $W$ model is similar; there are $k$ servers and $k + 1$ classes of jobs, with class $R$ replicating at all servers, and class $C_i$ going only to server $i$, $1 \leq i \leq k$ (see Figure 8(b)). In the scaled $M$ model, each class $R_i$, $1 \leq i \leq k$, joins the queue at its own dedicated server and at a single server shared by all classes (see Figure 8(c)).

**Figure 7:** Comparing redundancy, Opt-Split, and JSQ for the $M$ model. Lines shown include mean response time for the overall system under redundancy (solid blue), Opt-Split (dashed green), and JSQ with tiebreaking in favor of the faster server (dashed red). Mean response time as a function of (a) increasing $\lambda_R = \lambda_R$, (b) increasing $\mu_2$.

**Figure 8:** Scaled versions of (a) the $N$ model, (b) the $W$ model, and (c) the $M$ model.

The limiting probabilities derived in Theorem 1 for the general redundancy system apply to the scaled $N$, $W$, and $M$ models. In Theorem 10, we use this result to find that in both the scaled $N$ and $W$ models, response time for class $R$ is Exponentially distributed, extending the results of Theorem 2 and Theorem 5 respectively.

**THEOREM 10.**

1. In the scaled $N$ model, the distribution of the number of class $R$ jobs in the system is

\[ N_R \sim \text{Geometric} \left(1 - \frac{\lambda_R}{\sum_{i=1}^{k} \mu_i - \sum_{i=2}^{k} \lambda_i}\right) - 1, \]

and the distribution of the response time of class $R$ jobs is

\[ T_R \sim \text{Exp} \left(\sum_{i=1}^{k} \mu_i - \sum_{i=2}^{k} \lambda_i - \lambda_R\right). \]

2. In the scaled $W$ model, the distribution of the number of class $R$ jobs in the system is

\[ N_R \sim \text{Geometric} \left(1 - \frac{\lambda_R}{\sum_{i=1}^{k} \mu_i - \sum_{i=1}^{k} \lambda_i}\right) - 1, \]
and the distribution of the response time of class $R$ jobs is

$$T_R \sim \text{Exp} \left( \sum_{i=1}^{k} \mu_i - \sum_{i=1}^{k} \lambda_i - \lambda_R \right).$$

**Proof.** See Appendix. □

For the $M$ model and for the non-redundant classes in the $N$ and $W$ models, the result from Theorem 1 does not easily yield a closed form expression in the scaled models. The results discussed in the remainder of this section for these classes are obtained via simulation.

For the two-server $N$ and $W$ models, we saw that the redundant class had lower mean response time under redundancy than under both JSQ and Opt-Split, but often JSQ was very close to redundancy. Here, for scaled models, we investigate whether redundancy enjoys a greater advantage over JSQ and Opt-Split as the number of servers increases.

Indeed, we find that the redundant class sees a much greater benefit under redundancy than under Opt-Split and JSQ as $k$ increases for the scaled $N$ and $W$ models (see Figures 9(a) and (b)). In fact, as $k$ increases, the benefit grows unboundedly because when a class $R$ job enters a system with high $k$, it tends to see many idle servers. Under Opt-Split, this job may not be routed to one of the the idle servers. Under JSQ, the job goes to a single idle server $i$ and receives mean response time $\frac{1}{\mu_i}$. Under redundancy, the job gets to use all of the idle servers, thereby receiving mean response time $\frac{1}{\sum_{i=1}^{k} \frac{1}{\mu_i}}$.

In the two-server $N$ and $W$ models, we saw that the benefit that class $R$ received from redundancy came at a cost to the non-redundant class $A$. In the scaled $N$ and $W$ models, this cost approaches 0 because the pain caused by the redundant class is spread among all non-redundant classes, so the effect on any one of these classes is minimal; the response time for each non-redundant class $C_i$ approaches that of an $M/M/1$ with arrival rate $\lambda_{C_i}$ and service rate $\mu_i$. In the three-server version of the $M$ model (Section 6), we saw that redundancy significantly outperformed Opt-Split and JSQ. In Figure 9(c), we look at the relative performance of the three policies as $k$ increases. In the scaled $M$ model, at low $k$, redundancy indeed gives a lower mean response time than Opt-Split and JSQ. However, as $k$ increases, response time becomes the same under all three policies. As the load on the shared server becomes high, no class benefits from this server; each class experiences an independent $M/M/1$. Convergence to $k$ independent $M/M/1$ queues is slow; for example, at $k = 200$, redundancy still provides a 5% lower mean response time than independent $M/M/1$ queues.

8. CONCLUSION

In this paper we study a multi-server system with redundant requests. In such a system, each job that arrives joins the queue at some subset of the servers and departs the system as soon as it completes service at one of these servers. While recent empirical work in computer systems has demonstrated that redundant requests can greatly reduce response time, theoretical analysis of systems with redundancy has proved challenging.

We present the first exact analysis of systems with redundancy, deriving the limiting distribution of the queue state. Our state space is very complex and furthermore yields a non-product form, and non-obvious, limiting distribution. Nonetheless, we find very clean, simple results for response time distributions for both redundant and non-redundant classes in small systems. For large systems, we derive the response time distribution of a fully redundant class. Many of our results are counterintuitive:

1. The redundant class experiences a response time distribution identical to that in an $M/M/1$, even though the system is not an $M/M/1$ ($N$ and $W$ models).
2. Once a class is fully redundant, it is immune to additional classes becoming redundant: the distribution of its response time does not change ($N$ and $W$ models).
3. The non-redundant class often prefers the other class to be redundant as opposed to routing the other class according to Opt-Split or JSQ ($N$ and $W$ models).
4. Given two classes of jobs, $A$ and $B$, each with its own queue, if a class $R$ is redundant at both queues, the pain caused to class $A$ is equal to that caused to class $B$, even though $A$’s and $B$’s respective arrival rates and service rates may be different ($W$ model).
5. When multiple classes share a single server, redundancy can improve mean response time relative to Opt-Split and JSQ by a factor of 2 ($M$ model).
6. As the number of servers increases, redundancy gives an even greater benefit to the redundant class while...
causing less pain to the non-redundant classes (scaled N and scaled W models).

The redundancy system is closely related to many other queueing models for which exact analysis has long been elusive: Coupled processor systems, fork-join systems, and systems with flexible servers all bear a resemblance to redundancy systems in that they all involve jobs that can be processed by multiple servers. The specific mechanism that determines which jobs run on which servers, and whether jobs can run simultaneously on multiple servers, varies between models, but all of these models share the underlying theme of flexibility. We hope that the new analysis presented in this paper will provide insights on how to analyze these other difficult systems.

Acknowledgments

This material is based upon work supported by the National Science Foundation Graduate Research Fellowship under Grant No. DGE-1325222, was funded by 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; and has been supported by the Academy of Finland in TOP-Energy project (grant no. 268092).

9. REFERENCES

APPENDIX

A. PROOF OF THEOREM 1

Theorem 1. The limiting probability of being in state $(c_n, c_{n-1}, \ldots, c_1)$ is

$$\pi(c_n, \ldots, c_1) = C \prod_{i=1}^{n} \frac{\lambda_c}{\sum_{m \in \bigcup_{j \leq i} S_{c_j}} \mu_m},$$

where $C$ is a normalizing constant.

Proof. We begin by writing local balance equations for our states. The local balance equations are:

$$A \equiv \text{Rate entering state } (c_n, \ldots, c_1) \text{ due to an arrival} = \text{Rate leaving state } (c_n, \ldots, c_1) \text{ due to a departure} \equiv A'.$$

$$B_c \equiv \text{Rate entering state } (c_n, \ldots, c_1) \text{ due to a departure of class } c = \text{Rate leaving state } (c_n, \ldots, c_1) \text{ due to an arrival of class } c \equiv B'_c.$$

For an empty system, the state is $(\cdot)$. It is not possible to enter state $(\cdot)$ due to an arrival or to leave due to a departure, so we only have one local balance equation of the form $B_c = B'_c$:

$$\pi(i) \lambda_c = \pi(c) \sum_{m \in S_c} \mu_m. \quad (5)$$

For any other state $(c_n, c_{n-1}, \ldots, c_1)$, we have local balance equations of the form:

$$A = \pi(c_{n-1}, \ldots, c_1) \lambda_n = \pi(c_n, \ldots, c_1) \sum_{m \in \bigcup_{j \leq n} S_{c_j}} \mu_m = A' \quad (6)$$

$$B_c = \sum_{i=0}^{n} \sum_{m \in S_c, \ m \notin S_{c_j}, \ 1 \leq j \leq i} \pi(c_n, \ldots, c_{i+1}, c_i, \ldots, c_1) \mu_m$$

$$= \pi(c_n, \ldots, c_1) \lambda_c = B'_c. \quad (7)$$

We guess the following form for $\pi(c_n, \ldots, c_1)$:

$$\pi(c_n, \ldots, c_1) = C \prod_{i=1}^{n} \frac{\lambda_c}{\sum_{m \in \bigcup_{j \leq i} S_{c_j}} \mu_m}. \quad (8)$$

We will prove inductively that our guess satisfies the balance equations. The base case is state $(\cdot)$. Substituting the guess from (8) into the left-hand side of (5), we get:

$$\pi(c) \sum_{m \in S_c} \mu_m = C \sum_{m \in S_c} \frac{\lambda_c}{\mu_m} \sum_{m \in S_c} \mu_m$$

$$= C \lambda_c$$

$$= \pi(i) \lambda_c,$$

which is exactly the right-hand side of (5).

Now, assume that (6) and (7) hold for some $n - 1 \geq 0$. We will show that both hold for $n$.

1. $A = A'$. From (6), we have:

$$A = \pi(c_{n-1}, \ldots, c_1) \lambda_n = C \prod_{i=1}^{n-1} \frac{\lambda_c}{\sum_{m \in \bigcup_{j \leq i} S_{c_j}} \mu_m} \lambda_n$$

$$= \pi(c_n, \ldots, c_1) \lambda_n$$

$$= \pi(c_n, \ldots, c_1) \sum_{m \in \bigcup_{j \leq n} S_{c_j}} \mu_m$$

$$= \pi(c_n, \ldots, c_1) \lambda_c.$$
In the queue, and let the first guess for the limiting probabilities from (8) is correct.

\[ \Pr \{ N_R = n_R \} = \sum_{x_0=0}^{\infty} \sum_{x_1=0}^{\infty} C_{\eta R}(x_0, x_1) \prod_{j \geq x_0, c_j \neq R} X_j \]

where

\[ \eta_R = \frac{\lambda_R}{\sum_{m=1}^{\infty} \mu_m}, \]

\[ X_0 = \prod_{j=1}^{x_0} \sum_{m \in \cup \{ j \leq n \}} \mu_m, \]

\[ X_1 = \sum_{m \neq R} \lambda_m. \]

The sums in the numerators of \( X_0 \) and \( X_1 \) take into account all of the possible combinations of classes making up the \( x_0 \) and \( x_1 \) jobs, respectively.

Now let \( C_1 = \sum_{n_R=0}^{\infty} X_0 \) (note that this is a constant with respect to \( n_R \)). Using the identity \( \sum_{m=0}^{\infty} p \left( \frac{1}{1-p} \right)^m \) for \( |p| < 1 \), we have:

\[ \Pr \{ N_R = n_R \} = C C_{\eta R}^{\eta R} \left( \frac{1}{1 - \mu_m \lambda_m} \right)^{n_R}. \]

Using the normalization equation

\[ \sum_{n_R=0}^{\infty} \Pr \{ N_R = n_R \} = 1, \]

we find

\[ CC_1 = 1 - \frac{\lambda_R}{\sum_{m=1}^{\infty} \mu_m \sum_{i \neq R} \lambda_i}. \]

Hence \( N_R \sim \text{Geometric}(1 - \frac{\lambda_R}{\sum_{m=1}^{\infty} \mu_m \sum_{i \neq R} \lambda_i}) - 1. \)

Next, we obtain the Laplace transform of the response time for class \( R \) jobs, \( \tilde{T}_R(s) \), via distributional Little’s Law. First, we consider the \( z \)-transform of the number of class-\( R \) Poisson arrivals during \( T \), \( \tilde{A}_{T_R}(z) = \tilde{T}_R(\lambda_R - \lambda_R z) \). Class \( R \) jobs depart the system in the same order in which they arrive, so \( \tilde{A}_{T_R} \) is equivalent to \( N_R \), the number of jobs seen by an \( R \) departure. Hence

\[ \tilde{T}_R(\lambda_R - \lambda_R z) = \tilde{N}_R(z). \]

We know that \( N_R \) is distributed \( \text{Geometric}(p) - 1 \), where \( p = 1 - \frac{\lambda_R}{\sum_{m=1}^{\infty} \mu_m \sum_{i \neq R} \lambda_i}. \) Hence we have

\[ \tilde{T}_R(\lambda_R - \lambda_R z) = \tilde{N}_R(z) = \frac{p}{1 - z(1-p)}. \]

Let \( s = \lambda_R - \lambda_R z \), so that \( z = 1 - s/\lambda_R \). Then we have

\[ \tilde{T}_R(s) = \frac{p}{1 - (1 - s/\lambda_R)(1-p)}, \]

which after some simplification gives

\[ \tilde{T}_R(s) = \frac{\sum_{m \mu_m - \sum_{i \neq R} \lambda_i} - \lambda_R}{\sum_{m \mu_m - \sum_{i \neq R} \lambda_i} - \lambda_R + s}. \]

Hence \( \tilde{T}_R \sim \text{Exp} \left( \sum_{m \mu_m - \sum_{i \neq R} \lambda_i} - \lambda_R \right). \)

The derivation for the \( \mathbb{W} \) model is very similar, and is omitted here. \( \square \)