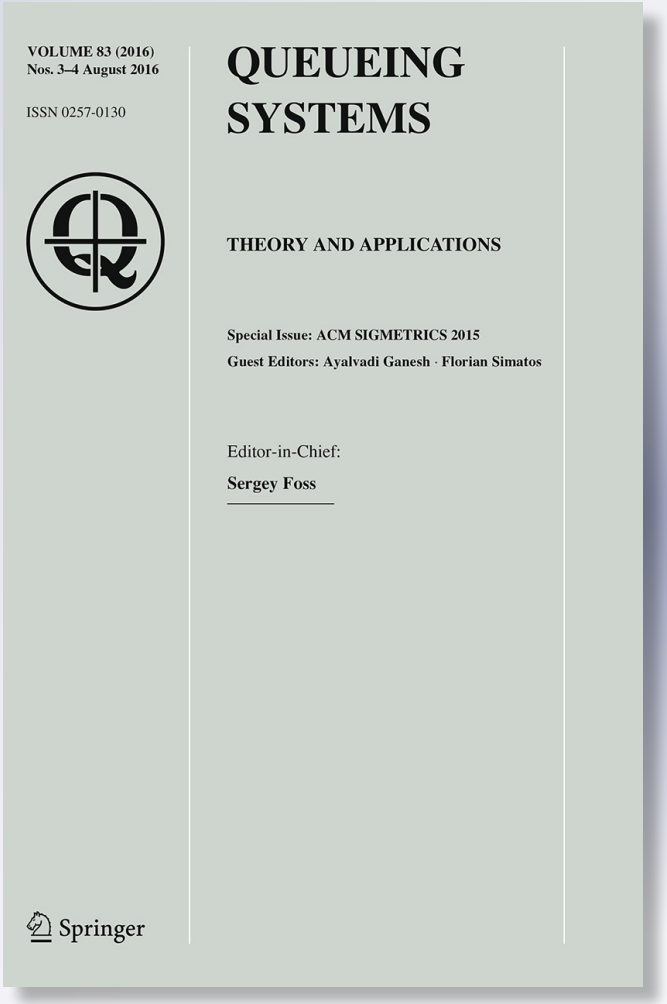
Queueing with redundant requests: exact analysis

Kristen Gardner, Samuel Zbarsky, Sherwin Doroudi, Mor Harchol-Balter, Esa Hyytiä & Alan Scheller-Wolf

Queueing Systems
Theory and Applications

ISSN 0257-0130
Volume 83
Combined 3-4

Queueing Syst (2016) 83:227-259
DOI 10.1007/s11134-016-9485-y



Springer

Queueing Syst (2016) 83:227–259
DOI 10.1007/s11134-016-9485-y

Queueing with redundant requests: exact analysis

Kristen Gardner[1] · **Samuel Zbarsky**[2] · **Sherwin Doroudi**[3] · **Mor Harchol-Balter**[1] · **Esa Hyytiä**[4] · **Alan Scheller-Wolf**[3]

Received: 11 September 2015 / Revised: 20 April 2016 / Published online: 2 July 2016

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 of 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.

Keywords Redundancy · Markov chain analysis · Dispatching

✉ Kristen Gardner
ksgardne@cs.cmu.edu

1 School of Computer Science, Carnegie Mellon University, Pittsburgh, USA

2 Mathematics Department, Carnegie Mellon University, Pittsburgh, USA

3 Tepper School of Business, Carnegie Mellon University, Pittsburgh, USA

4 Department of Computer Science, University of Iceland, Reykjavik, Iceland

Mathematics Subject Classification 60J27

1 Introduction

Reducing latency has always been a primary concern for computer systems designers. Recent papers have proposed a new approach to reducing latency in multi-server systems: using *redundant requests* [2,3,10,28,35]. The motivation behind this approach comes from the observation that response times at servers can be highly variable [12,35]. Two servers in the same system often differ in their current loads, their network congestion, and the configuration of their storage systems. Even if both servers are idle, the same request might experience a far lower service time at one server than another, for example because the disk seek time could be much lower at one server than another (seek times often dominate service times [19]). The solution is to send the same request to the queues at multiple servers simultaneously (i.e., redundantly). When any copy of the request completes service, all remaining copies of the request are killed.

Using redundant requests is not free. First, data must be replicated among the set of servers to which the copies must be sent. Furthermore, using redundant requests adds to the system load. Nonetheless, using redundant requests has been shown to vastly improve latency in distributed systems, for example, Google's BigTable service shows a 20-fold improvement in tail latency using redundant requests [12].

Unfortunately, there is almost no work analyzing the benefits of redundant requests. Even a two-server system with one redundant class and one non-redundant class has not been analyzed. The first attempts to analyze systems with redundancy are as recent as 2014, but this work derives only bounds and approximations [21].

Redundant requests require a new queueing paradigm: there is no longer a single copy of each job, and redundant copies disappear instantly as soon as one copy completes. While redundant jobs bear some resemblance to fork–join systems, the two models are actually quite different because all copies must complete service in a fork–join system, whereas a redundant job only needs one copy to complete. Likewise, while redundant jobs bear some resemblance to coupled processor systems, they differ in that the redundant copies can occupy multiple servers even when these servers have non-empty queues. Likewise, redundant jobs are not the same as flexible servers (see Sect. 2 for more details).

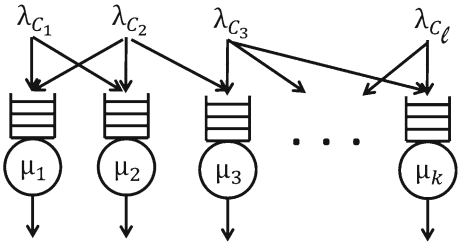


Fig. 1 The general redundancy model. Each server j provides service at rate μ_j. Each class of jobs C_i arrives to the system as a Poisson process with rate λ_{C_i} and joins the queue at *all* servers in $S_{C_i} = \{j \mid \text{server } j \text{ can serve class } C_i\}$

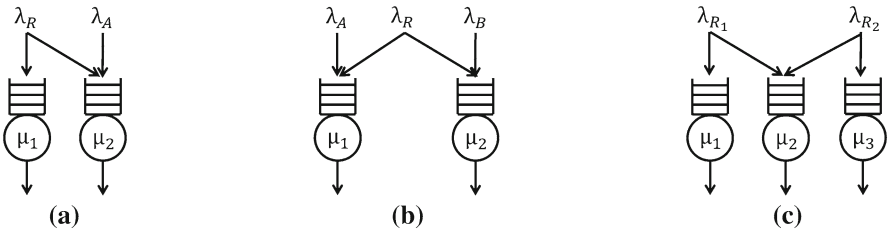


Fig. 2 **a** The $\mathbb{N}$ model. Class A jobs join the queue at server 2 only, while class R jobs are redundant at both servers. **b** The $\mathbb{W}$ model. Class A jobs join the queue at server 1 only, class B jobs join the queue at server 2 only, and class R jobs are redundant at both servers. **c** The $\mathbb{M}$ model. Class R_1 jobs are redundant at servers 1 and 2, and class R_2 jobs are redundant at servers 2 and 3

The state space required to capture systems with redundant jobs is very complex. It is not enough to know the number of jobs in each queue, or even the number of jobs of each class (redundant or non-redundant) within each queue. Rather, one needs to track the exact position and type of every job in every queue, so that one knows which jobs to delete when a copy of a redundant job completes service.

This paper provides the first closed-form exact analysis of redundant queues. We derive the limiting distribution of the full queue state as well as (in some cases) the distribution of response time of each class of jobs. Our analysis assumes exponential service times and Poisson arrivals. Our result applies to systems with any number of queues, k, any number of classes of jobs, ℓ, and any redundancy structure (see Fig. 1). A *class* of jobs is associated with a set of servers that hold replicated data; the jobs of a class can be run on any of the servers associated with the class.

We also investigate how the approach of redundant requests compares to other common approaches for job assignment. For example, how does making m redundant copies of each request compare with optimally probabilistically splitting load among m queues (Opt-Split), or with joining the shortest of m queues (JSQ)? Furthermore, while redundancy may benefit the redundant class, what is the response time penalty to the other jobs in the system? Do other approaches create less of a penalty? Finally, if one class of jobs creates redundant copies, does that class suffer when others "join the redundancy game" and start creating redundant copies of their jobs as well?

We first investigate these questions in the context of three simple models, shown in Fig. 2. In the $\mathbb{N}$ model (Fig. 2a), there are two arrival streams of jobs, each with its own server. However, one class is redundant at both servers. The $\mathbb{N}$ model illuminates the response time benefit to the redundant class and the pain to the non-redundant class. We derive the exact distribution of response time for both classes, and explore what happens when the non-redundant class decides that it too wants to become redundant. In the $\mathbb{W}$ model (Fig. 2b), we imagine that we have a stable system, where each server is serving its own stream of jobs, when a new stream of jobs arrives which can be processed at either server. We ask how to best deal with this new stream: redundancy, splitting, or dispatching to the shortest queue? We then turn to the $\mathbb{M}$ model (Fig. 2c), 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 $\mathbb{N}$, $\mathbb{W}$, and $\mathbb{M}$ 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 Sect. 2 we describe related work and how it differs from the present work. In Sect. 3 we formalize our model and state our main theorem for the general system. In Sects. 4, 5, and 6 we discuss detailed results for the $\mathbb{N}$, $\mathbb{M}$, and $\mathbb{W}$ models. Section 7 addresses how these models scale as the number of servers increases. In Sect. 8 we consider what happens when we relax our modeling assumptions. In Sect. 9 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 Bs (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 [13,24], 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 [11] using a similar approach. In [18], 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, [7]).

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 be idle (the servers do not "couple"). The donor–beneficiary model has been studied, in approximation, in a variety of settings [17,27]. 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 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, [4,5,23,26,37]). Exact analysis remains an open problem except for the two-server case [14,15]; see [9] 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 [6,31–33]. 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 [1,34]. 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 [29], 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 [20,21,29], and the optimal value of k has been determined for different system loads and costs of deleting extra redundant requests [30]. Additionally, other variants have been proposed where a job might wait before issuing redundant requests [36]. 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/∞ queue, so there is no queueing [20].

In the special case $r = 1$, which mostly closely matches our model, approximations have been found assuming that the queues are independent [35]. The optimal degree of replication and the optimal service discipline also have been studied [25].

3 Model

We consider a system with k servers, denoted as $1, 2, \ldots, k$, and ℓ classes of jobs, denoted as $C_1, C_2, \ldots, C_\ell$. (see Fig. 1). The service time at server j is distributed exponentially with rate μ_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 λ_{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 Fig. 1, it is difficult to figure out an appropriate state space. One might think that you could track the number of jobs of each 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 Fig. 3). Each server processes jobs from this central queue in FCFS order, skipping over those jobs it cannot serve. For example, in Fig. 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 $(c_n, c_{n-1}, \ldots, c_1)$, where there are n jobs in the system, and c_i is the class of the ith 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 *When* $\forall\, \mathcal{C} \subseteq \{C_1, \ldots, C_\ell\}$,

$$\sum_{C \in \mathcal{C}} \lambda_C < \sum_{m \in \bigcup_{C \in \mathcal{C}} S_C} \mu_m,$$

the system is stable, and the limiting probability of being in state $(c_n, c_{n-1}, \ldots, c_1)$ *is*

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

where C *is a normalizing constant.*

Proof Deferred to the end of the section. □

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,

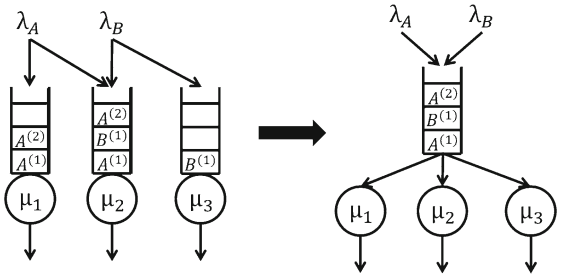


Fig. 3 Let A and B be two job classes, where $A^{(i)}$ is the ith arrival of class A. We can view the general redundancy system (*left*) as having a single central queue from which each server works in FCFS order, skipping over those jobs it cannot serve (*right*). The central queue is an interleaving of the individual servers' queues, where each job appears only once, and appears in the order in which it arrived

or as a product of independent marginal per-class terms. In fact, the form is quite unusual, as illustrated in Example 1.

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

$$\pi_{(A,B,A)} = C\left(\frac{\lambda_A}{\mu_1+\mu_2}\right)\left(\frac{\lambda_B}{\mu_1+\mu_2+\mu_3}\right)\left(\frac{\lambda_A}{\mu_1+\mu_2+\mu_3}\right).$$

Note that the denominator for the first class-A job is $\mu_1+\mu_2$ whereas the denominator for the second class-A job is $\mu_1+\mu_2+\mu_3$. The denominator for the ith job in the queue depends on the classes of all jobs ahead of it, and so the limiting probability cannot be written as a product of per-class terms.

In Sects. 4, 5, and 6 we use the result of Theorem 1 to study the $\mathbb{N}$, $\mathbb{W}$, and $\mathbb{M}$ models, defined below.

$\mathbb{N}$ Model

The $\mathbb{N}$ model is the simplest non-trivial example of a redundancy system where there are both redundant and non-redundant classes. In an $\mathbb{N}$ model there are two servers running at rates μ_1 and μ_2 and two classes of jobs (see Fig. 2a). Class A jobs are non-redundant; they arrive with rate λ_A and join the queue at server 2 only ($S_A = \{2\}$). Class R jobs are redundant; they arrive with rate λ_R and join the queue at both servers ($S_R = \{1,\ 2\}$).

$\mathbb{W}$ 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 $\mathbb{W}$ model helps us understand how the presence of this redundant class affects the existing non-redundant classes. In a $\mathbb{W}$ model, there are two servers running at rates μ_1 and μ_2 and three classes of jobs (see Fig. 2b). Class A jobs arrive with rate λ_A and join the queue at server 1 only ($S_A = \{1\}$), class B jobs arrive with rate λ_B and join the queue at server 2 only ($S_B = \{2\}$), and class R jobs arrive with rate λ_R and join the queue at both servers ($S_R = \{1,\ 2\}$).

$\mathbb{M}$ 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 $\mathbb{M}$ model helps us understand how best to use the new server. In an $\mathbb{M}$ model, there are three servers with rates μ_1, μ_2, and μ_3 and two job classes (see Fig. 2c). Class R_1 jobs arrive with rate λ_{R_1} and join the queue at servers 1 and 2 ($S_{R_1} = \{1,\ 2\}$), and class R_2 jobs arrive with rate λ_{R_2} and have $S_{R_2} = \{2,\ 3\}$.

3.1 Proof of Theorem 1

Proof (Theorem 1) The stability condition can be seen as a generalization of Hall's Theorem [16], where the proof follows from max-flow–min-cut. To prove the formation of the limiting probabilities, we begin by writing local balance equations for our states.

The local balance equations are:

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

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

$$\pi_{()}\lambda_c = \pi_{(c)} \sum_{m \in S_c} \mu_m. \tag{1}$$

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

$$A = \pi_{(c_{n-1},\ldots,c_1)}\lambda_{c_n} = \pi_{(c_n,\ldots,c_1)} \sum_{m \in \bigcup_{j \le n} S_{c_j}} \mu_m = A' \tag{2}$$

$$B_c = \sum_{i=0}^{n} \sum_{\substack{m \in S_c, \\ m \notin S_{c_j}, \\ 1 \le j \le i}} \pi_{(c_n,\ldots,c_{i+1},c,c_i,\ldots,c_1)} \mu_m = \pi_{(c_n,\ldots,c_1)} \lambda_c = B'_c. \tag{3}$$

We guess the following form for $\pi_{(c_n,\ldots,c_1)}$:

$$\pi_{(c_n,\ldots,c_1)} = C \prod_{i=1}^{n} \frac{\lambda_{c_i}}{\sum\limits_{m \in \bigcup\limits_{j \le i} S_{c_j}} \mu_m}. \tag{4}$$

We will prove inductively that our guess satisfies the balance equations. The base case is state $()$. Substituting the guess from (4) into the left-hand side of (1), we get

$$\begin{aligned} \pi_{(c)} \sum_{m \in S_c} \mu_m &= C \frac{\lambda_c}{\sum\limits_{m \in S_c} \mu_m} \sum_{m \in S_c} \mu_m \\ &= C \lambda_c \\ &= \pi_{()} \lambda_c, \end{aligned}$$

which is exactly the right-hand side of (1), letting $C = \pi_{()}$.

Now, assume that (2) and (3) hold for some $n - 1 \ge 0$. We will show that both hold for n.

1. $\mathbf{A} = \mathbf{A}'$. From (2), we have

$$\begin{aligned} A = \pi_{(c_{n-1},\ldots,c_1)} \lambda_{c_n} &= C \prod_{i=1}^{n-1} \frac{\lambda_{c_i}}{\sum\limits_{m \in \bigcup\limits_{j \le i} S_{c_j}} \mu_m} \lambda_{c_n} \\ &= \pi_{(c_n,\ldots,c_1)} \frac{\sum_{m \in \bigcup\limits_{j \le n} S_{c_j}} \mu_m}{\lambda_{c_n}} \lambda_{c_n} \\ &= \pi_{(c_n,\ldots,c_1)} \sum_{m \in \bigcup\limits_{j \le n} S_{c_j}} \mu_m \\ &= A'. \end{aligned}$$

2. $\mathbf{B_c} = \mathbf{B'_c}$. From (3), we have

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

$$
\begin{aligned}
&= \sum_{i=0}^{n-1} \sum_{m \in S_c \setminus \bigcup_{j\le i} S_{c_j}} \pi_{(c_n,\ldots,c_{i+1},c,c_i,\ldots,c_1)}\mu_m + \sum_{m \in S_c \setminus \bigcup_{j\le n} S_{c_j}} \pi_{(c,c_n,\ldots,c_1)}\mu_m \\
&= \sum_{i=0}^{n-1} \sum_{m \in S_c \setminus \bigcup_{j\le i} S_{c_j}} \frac{\lambda_{c_n}\pi_{(c_{n-1},\ldots,c_{i+1},c,c_i,\ldots,c_1)}}{\sum_{t \in \bigcup_{j\le n} S_{c_j} \cup S_c} \mu_t}\mu_m + C\frac{\lambda_c \sum_{m \in S_c \setminus \bigcup_{j\le n} S_{c_j}} \mu_m}{\sum_{m \in \bigcup_{j\le n} S_{c_j} \cup S_c} \mu_m} \prod_{i=1}^{n} \frac{\lambda_{c_i}}{\sum_{m \in \bigcup_{j\le i} S_{c_j}} \mu_m} \\
&= \frac{\lambda_{c_n}}{\sum_{m \in \bigcup_{j\le n} S_{c_j} \cup S_c} \mu_m} \sum_{i=0}^{n-1} \sum_{m \in S_c \setminus \bigcup_{j\le i} S_{c_j}} \pi_{(c_{n-1},\ldots,c_{i+1},c,c_i,\ldots,c_1)}\mu_m \\
&\quad + \lambda_c \pi_{(c_n,\ldots,c_1)} \frac{\sum_{m \in S_c \setminus \bigcup_{j\le n} S_{c_j}} \mu_m}{\sum_{m \in \bigcup_{j\le n} S_{c_j} \cup S_c} \mu_m} \\
&= \frac{\lambda_{c_n}}{\sum_{m \in \bigcup_{j\le n} S_{c_j} \cup S_c} \mu_m} \pi_{(c_{n-1},\ldots,c_1)}\lambda_c + \lambda_c \pi_{(c_n,\ldots,c_1)} \frac{\sum_{m \in S_c \setminus \bigcup_{j\le n} S_{c_j}} \mu_m}{\sum_{m \in \bigcup_{j\le n} S_{c_j} \cup S_c} \mu_m} \\
&= C\lambda_c \frac{\lambda_{c_n}}{\sum_{m \in \bigcup_{j\le n} S_{c_j} \cup S_c} \mu_m} \prod_{i=1}^{n-1} \frac{\lambda_{c_i}}{\sum_{m \in \bigcup_{j\le i} S_{c_j}} \mu_m} + \lambda_c \pi_{(c_n,\ldots,c_1)} \frac{\sum_{m \in S_c \setminus \bigcup_{j\le n} S_{c_j}} \mu_m}{\sum_{m \in \bigcup_{j\le n} S_{c_j} \cup S_c} \mu_m} \\
&= \lambda_c \pi_{(c_n,\ldots,c_1)} \frac{\sum_{m \in \bigcup_{j\le n} S_{c_j}} \mu_m}{\sum_{m \in \bigcup_{j\le n} S_{c_j} \cup S_c} \mu_m} + \lambda_c \pi_{(c_n,\ldots,c_1)} \frac{\sum_{m \in S_c \setminus \bigcup_{j\le n} S_{c_j}} \mu_m}{\sum_{m \in \bigcup_{j\le n} S_{c_j} \cup S_c} \mu_m} \\
&= \lambda_c \pi_{(c_n,\ldots,c_1)} \\
&= B'_c.
\end{aligned}
$$

Hence the local balance equations hold for all n, and so the guess for the limiting probabilities from (4) is correct. □

4 The $\mathbb{N}$ model

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

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

$$\pi_{(c_n,\ldots,c_1)} = C_{\mathbb{N}} \left(\frac{\lambda_A}{\mu_2}\right)^{a_0} \left(\frac{\lambda_R}{\mu_1+\mu_2}\right)^{r} \left(\frac{\lambda_A}{\mu_1+\mu_2}\right)^{a_1},$$

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 $C_{\mathbb{N}} = \left(1-\frac{\lambda_A}{\mu_2}\right)\left(1-\frac{\lambda_R}{\mu_1+\mu_2-\lambda_A}\right)$ 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 queue. Specifically, the distribution of response time is the same as that in an M/M/1 queue where the arrival rate is λ_R and the service rate is $\mu' = \mu_1+\mu_2-\lambda_A$.

Note that μ' can be viewed as giving the R jobs the full μ_1, and the *portion* of μ_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 queue with arrival rate $\lambda_A + \lambda_R$ and service rate $\mu_1 + \mu_2$.[1]

Theorem 2 *In the* $\mathbb{N}$ *model,*

1. *The number of class R jobs in the system,* N_R*, is distributed as* Geometric$(1-\rho)-1$*, where* $\rho = \frac{\lambda_R}{\mu_1+\mu_2-\lambda_A}$.
2. *The response time of class R jobs,* T_R*, is distributed as* $\text{Exp}(\mu_1 + \mu_2 - \lambda_A - \lambda_R)$.

Proof This is a special case of the more general result in Theorem 10, which is proved in Sect. 7.1. □

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 queue with arrival rate λ_A and service rate μ_2, plus a penalty term that captures the extent to which the redundant jobs hurt the As (Eq. 7). In Sect. 4.2 we give an alternative interpretation for T_A which provides intuition for this penalty term.

Theorem 3 *In the* $\mathbb{N}$ *model,*

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

$$\Pr\{N_A = n_A\} = \zeta_{\mathbb{N}1}\left(\frac{\lambda_A}{\mu_2}\right)^{n_A} + \zeta_{\mathbb{N}2}\left(\frac{\lambda_A}{\mu_1+\mu_2-\lambda_R}\right)^{n_A}, \tag{5}$$

where

$$\zeta_{\mathbb{N}1} = C_{\mathbb{N}}\left(\frac{\mu_1}{\mu_1-\lambda_R}\right),$$
$$\zeta_{\mathbb{N}2} = C_{\mathbb{N}}\left(\frac{\lambda_R}{\mu_1+\mu_2-\lambda_R} - \frac{\lambda_R}{\mu_1-\lambda_R}\right),$$

and $C_{\mathbb{N}}$ *is as in Lemma 1.*

2. *The distribution of response time of class A jobs is*

$$T_A \sim H_2(\nu_{\mathbb{N}1}, \nu_{\mathbb{N}2}, \omega_{\mathbb{N}}),$$

[1] 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] A generalized hyperexponential $H_2(\nu_1, \nu_2, \omega)$ is defined as the weighted mixture of two exponentials with rates ν_1 and ν_2, where the first exponential is given weight $1 + \omega$ and the second is given weight $-\omega$. Note that ω can be any real number; it need not be a probability [8].

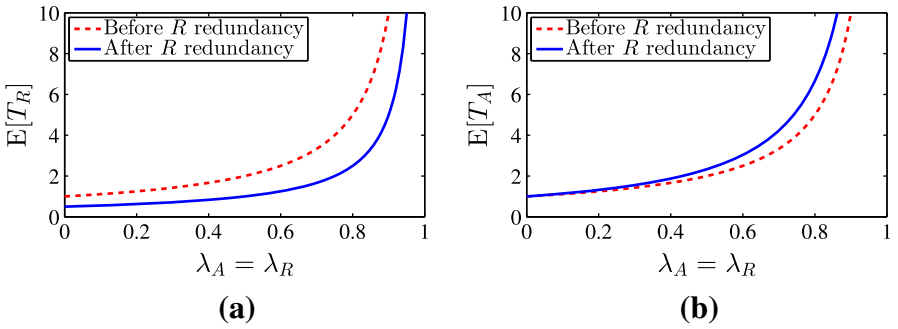


Fig. 4 Comparing mean response time before and after class R becomes redundant when $\mu_1 = \mu_2 = 1$ and $\lambda_A = \lambda_R$ for **a** class R, and **b** class A. The mean response time for the overall system is the weighted average of these two classes

where

$$\begin{aligned} \nu_{\mathbb{N}1} &= \mu_2 - \lambda_A, \\ \nu_{\mathbb{N}2} &= \mu_1 + \mu_2 - \lambda_A - \lambda_R, \\ \nu_{\mathbb{N}3} &= \mu_1 + \mu_2 - \lambda_A, \\ \omega_{\mathbb{N}} &= \frac{\lambda_R \nu_{\mathbb{N}1}}{(\mu_1 - \lambda_R)\nu_{\mathbb{N}3}}. \end{aligned} \tag{6}$$

The expected response time of class A jobs is

$$\mathbb{E}[T_A] = \underbrace{\frac{1}{\nu_{\mathbb{N}1}}}_{\text{M/M/1}} + \underbrace{\frac{1}{\nu_{\mathbb{N}2}} - \frac{1}{\nu_{\mathbb{N}3}}}_{\text{penalty}}. \tag{7}$$

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 queue), 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 (Fig. 4a), but can hurt class A by up to 50 % (Fig. 4b).

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 transforms the system into an M/M/1 queue 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 As become redundant, they receive the benefit of having two servers.

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

Proof 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 queue with arrival rate $\lambda_A + \lambda_R$ and service rate $\mu_1 + \mu_2$. □

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

1. $\mathbb{E}[T_A]^{\text{Fully Redundant}} \leq \mathbb{E}[T_A]^{\text{Redundant}}$
2. $T_R^{\text{Fully Redundant}} \stackrel{d}{=} T_R^{\text{Redundant}}$.

Proof From Theorem 2, in the $\mathbb{N}$ model, $T_R \sim \text{Exp}(\mu_1 + \mu_2 - \lambda_A - \lambda_B)$, which is the response time distribution in an M/M/1 queue with arrival rate $\lambda_A + \lambda_R$ and service rate $\mu_2 + \mu_2$. From Theorem 3, in the $\mathbb{N}$ model,

$$\mathbb{E}[T_A] = \frac{1}{\mu_1 + \mu_2 - \lambda_A - \lambda_R} + \frac{1}{\mu_2 - \lambda_A} - \frac{1}{\mu_1 + \mu_2 - \lambda_A},$$

which is at least the mean response time in an M/M/1 queue with arrival rate $\lambda_A + \lambda_R$ and service rate $\mu_1 + \mu_2$ since $\frac{1}{\mu_2 - \lambda_A} - \frac{1}{\mu_1 + \mu_2 - \lambda_A}$ is non-negative. □

Going back to the $\mathbb{N}$ 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 probabilistically split jobs among all allowable servers (Opt-Split), or join the shortest queue among allowable servers (JSQ).

In Fig. 5, we compare these other options for the $\mathbb{N}$ 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.

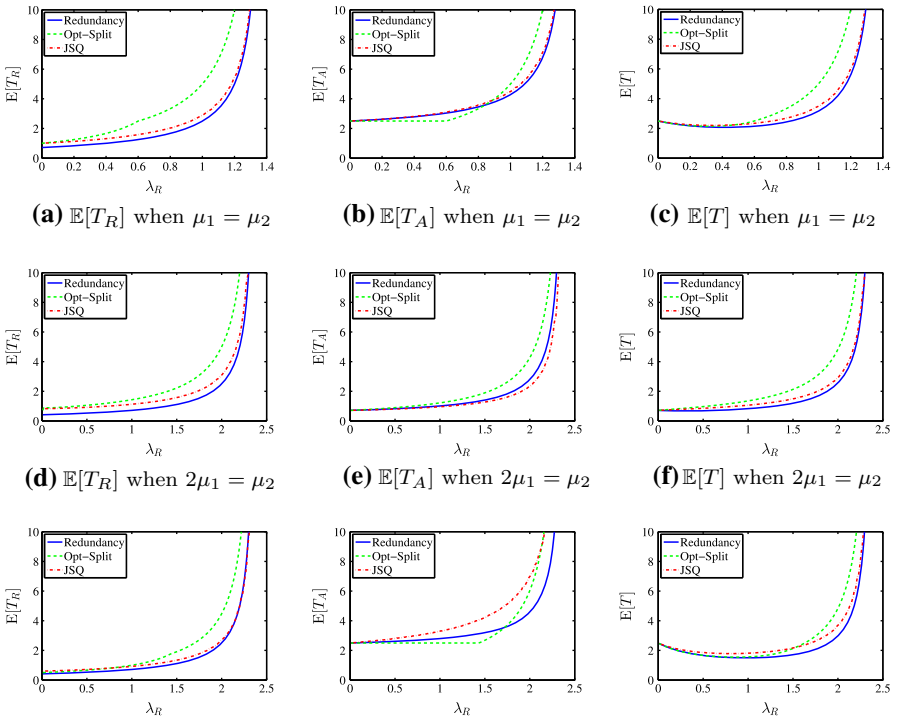
Surprisingly, the non-redundant class A often prefers redundancy of the other class to Opt-Split or JSQ. This is because the non-redundant class wants the redundant class to spend as little time as possible blocking the A jobs at server 2, and redundancy helps with this.

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

From the overall system's perspective, redundancy is always 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).

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



(a) $\mathbb{E}[T_R]$ when $\mu_1 = \mu_2$ **(b)** $\mathbb{E}[T_A]$ when $\mu_1 = \mu_2$ **(c)** $\mathbb{E}[T]$ when $\mu_1 = \mu_2$

(d) $\mathbb{E}[T_R]$ when $2\mu_1 = \mu_2$ **(e)** $\mathbb{E}[T_A]$ when $2\mu_1 = \mu_2$ **(f)** $\mathbb{E}[T]$ when $2\mu_1 = \mu_2$

(g) $\mathbb{E}[T_R]$ when $\mu_1 = 2\mu_2$ **(h)** $\mathbb{E}[T_A]$ when $\mu_1 = 2\mu_2$ **(i)** $\mathbb{E}[T]$ when $\mu_1 = 2\mu_2$

Fig. 5 Comparing redundancy (*solid blue*), Opt-Split (*dashed green*), and JSQ (*dashed red*) for the $\mathbb{N}$ model as λ_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 (Color figure online)

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

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

1. $\frac{1}{2} \le \frac{\mathbb{E}[T_R]^{\text{Redundant}}}{\mathbb{E}[T_R]^{\text{Opt}-\text{Split}}} \le 1$. *If $\lambda_R > \lambda_A$, then* $\frac{\mathbb{E}[T_R]^{\text{Redundant}}}{\mathbb{E}[T_R]^{\text{Opt}-\text{Split}}} = \frac{1}{2}$.
2. $\frac{\mathbb{E}[T_A]^{\text{Redundant}}}{\mathbb{E}[T_A]^{\text{Opt}-\text{Split}}} \le \frac{3}{2}$.
3. $\frac{1}{2} \le \frac{\mathbb{E}[T]^{\text{Redundant}}}{\mathbb{E}[T]^{\text{Opt}-\text{Split}}} \le 1$.

Proof Deferred to the end of the section. □

4.1 Why redundancy helps

The analysis presented above demonstrates that redundancy leads to a significant response time improvement for both redundant class R jobs and non-redundant class A jobs. Redundancy helps because it takes advantage of two sources of variability in the system.

First, queueing times can be very different at different servers. Sending redundant requests enables a job to wait in multiple queues and therefore to obtain the shortest possible queueing time. The queueing time experienced under redundancy is even better than that under JSQ because JSQ considers only the number of jobs in the queue, not the actual duration of work in the queue. In fact, redundancy can be seen as obtaining the same queueing time benefit as the least work left (LWL) dispatching policy, which assumes that job sizes are known in advance and an arriving job is dispatched to the queue with the least work. Importantly, redundancy achieves the LWL queueing time benefit without having to know job sizes.

The second source of variability that redundancy leverages is variability in the same job's service times on different servers. If a job is in service at multiple servers at the same time, it may experience very different service times on these different servers. The job departs the system as soon as its first copy completes service, so it receives the minimum service time across many servers. The benefits of inter-server variability are unique to redundancy. Policies such as JSQ and LWL, which dispatch only one copy of each job, do not take advantage of the potential response time gains from running multiple copies of the same job. Hence even though JSQ and LWL load balance successfully to overcome queueing time variability, redundancy provides even lower response times by also leveraging service time variability.

4.2 Alternative interpretation of T_A

From Theorem 3, we know that T_A follows a generalized hyperexponential distribution. The Laplace transform of T_A is

$$\widetilde{T}_A(s) = (1+\omega_{\mathbb{N}})\frac{\nu_{\mathbb{N}1}}{\nu_{\mathbb{N}1}+s} - \omega_{\mathbb{N}}\frac{\nu_{\mathbb{N}2}}{\nu_{\mathbb{N}2}+s}, \tag{8}$$

where $\nu_{\mathbb{N}1}$, $\nu_{\mathbb{N}2}$, and $\omega_{\mathbb{N}}$ are as defined in (6). An alternative way of writing the transform is

$$\widetilde{T}_A(s) = \left(\frac{\mu_1+\mu_2-\lambda_A-\lambda_R}{\mu_1+\mu_2-\lambda_A-\lambda_R+s}\right)\left(\frac{\mu_1+\mu_2-\lambda_A+s}{\mu_1+\mu_2-\lambda_A}\right)\left(\frac{\mu_2-\lambda_A}{\mu_2-\lambda_A+s}\right). \tag{9}$$

Note that the Laplace transform of response time in an M/M/1 queue with arrival rate λ and service rate μ is

$$\widetilde{T}(s) = \frac{\mu-\lambda}{\mu-\lambda+s},$$

and the Laplace transform of queueing time in such an M/M/1 queue is

$$\widetilde{T}_Q(s) = \left(\frac{\mu - \lambda}{\mu - \lambda + s}\right)\left(\frac{\mu + s}{\mu}\right).$$

Hence we can interpret the response time for class A jobs in the $\mathbb{N}$ model as consisting of two components:

1. $\left(\frac{\mu_1+\mu_2-\lambda_A-\lambda_R}{\mu_1+\mu_2-\lambda_A-\lambda_R+s}\right)\left(\frac{\mu_1+\mu_2-\lambda_A+s}{\mu_1+\mu_2-\lambda_A}\right)$ is the transform of the queueing time in an M/M/1 queue with arrival rate λ_R and service rate $\mu_1 + \mu_2 - \lambda_A$.
2. $\left(\frac{\mu_2-\lambda_A}{\mu_2-\lambda_A+s}\right)$ is the transform of the response time in an M/M/1 queue with arrival rate λ_R and service rate μ_2.

The intuition behind this result is as follows. Consider a tagged class A job arriving to the $\mathbb{N}$ system. The tagged A sees both class A jobs and class R jobs ahead of it in the queue. We first consider the work made up of class R jobs. Before the class A job can enter service, all class R jobs ahead of it must depart from the system. Thus an arriving A jobs has to wait behind all work in its queue made up of class R jobs. That total work is the same as the queueing time that a class R arrival would experience. Hence the first thing the class A job experiences is the queueing time for a class R job. From Theorem 2, we know that class R jobs experience the same response time distribution as in an M/M/1 queue with arrival rate λ_R and service rate $\mu_1 + \mu_2 - \lambda_A$. Hence the class A job first experiences the queueing time in such an M/M/1 queue.

After all of the class R jobs ahead of the tagged class A job depart, there may still be other As ahead of the tagged job. Hence the tagged A then experiences the response time in a dedicated class-A M/M/1 queue with arrival rate λ_A and service rate μ_2.

4.3 Proofs for $\mathbb{N}$ model

Proof (Theorem 3) We first consider the case $n_A = 0$, noting that there can be any number of R jobs in the system.

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

Now assume $n_A > 0$. We consider three cases:

1. There are no R jobs in the system. Then

$$\Pr\{N_A = n_A \text{ and no } R \text{ jobs in system}\} = C_{\mathbb{N}} \left(\frac{\lambda_A}{\mu_2}\right)^{n_A}. \tag{10}$$

2. There are both R and A jobs in the system, and there is an R at the head of the queue. Let $r_0 + 1$ be the number of R jobs before the first A, and r_i be the number of R jobs following the ith A, $i > 0$. Then

$$
\begin{aligned}
&\Pr\{N_A = n_A,\ R \text{ at head}\} \\
&\quad = \sum_{r_0=0}^{\infty}\sum_{r_1=0}^{\infty}\cdots\sum_{r_{n_A}=0}^{\infty} C_{\mathbb{N}}\left(\frac{\lambda_R}{\mu_1+\mu_2}\right)^{1+r_0+\cdots+r_{n_A}}\left(\frac{\lambda_A}{\mu_1+\mu_2}\right)^{n_A} \\
&\quad = C_{\mathbb{N}}\left(\frac{\lambda_A}{\mu_1+\mu_2}\right)^{n_A}\left(\sum_{r_0=0}^{\infty}\left(\frac{\lambda_R}{\mu_1+\mu_2}\right)^{r_0}\right)\cdots\left(\sum_{r_{n_A}=0}^{\infty}\left(\frac{\lambda_R}{\mu_1+\mu_2}\right)^{r_{n_A}}\right)\left(\frac{\lambda_R}{\mu_1+\mu_2}\right) \\
&\quad = C_{\mathbb{N}}\left(\frac{\lambda_A}{\mu_1+\mu_2}\right)^{n_A}\left(\frac{1}{1-\frac{\lambda_R}{\mu_1+\mu_2}}\right)^{n_A+1}\left(\frac{\lambda_R}{\mu_1+\mu_2}\right) \\
&\quad = C_{\mathbb{N}}\left(\frac{\lambda_A}{\mu_1+\mu_2}\right)^{n_A}\left(\frac{\mu_1+\mu_2}{\mu_1+\mu_2-\lambda_R}\right)^{n_A+1}\left(\frac{\lambda_R}{\mu_1+\mu_2}\right) \\
&\quad = C_{\mathbb{N}}\left(\frac{\lambda_R}{\mu_1+\mu_2-\lambda_R}\right)\left(\frac{\lambda_A}{\mu_1+\mu_2-\lambda_R}\right)^{n_A}. \qquad (11)
\end{aligned}
$$

3. There are both R and A jobs in the system, and there is an A at the head of the queue. Let $\ell+1 = a_0$ be the number of A jobs before the first R, let r_1+1 be the number of Rs following these initial As, and let r_i be the number of Rs following the $(\ell+i)$th A, $i>1$. Then

$$
\begin{aligned}
&\Pr\{N_A = n_A,\ A \text{ at head}\} \\
&\quad = \sum_{\ell=0}^{n_A-1}\sum_{r_1=0}^{\infty}\cdots\sum_{r_{n_A-\ell}=0}^{\infty} C_{\mathbb{N}}\left(\frac{\lambda_A}{\mu_2}\right)^{\ell+1} \\
&\qquad \times\left(\frac{\lambda_R}{\mu_1+\mu_2}\right)^{1+r_1+\cdots+r_{n_A-\ell}}\left(\frac{\lambda_A}{\mu_1+\mu_2}\right)^{n_A-\ell-1} \\
&\quad = C_{\mathbb{N}}\left(\frac{\lambda_R}{\mu_2}\right)\sum_{\ell=0}^{n_A-1}\left(\frac{\lambda_A}{\mu_2}\right)^{\ell}\left(\frac{\lambda_A}{\mu_1+\mu_2}\right)^{n_A-\ell}\left(\sum_{r_1=0}^{\infty}\left(\frac{\lambda_R}{\mu_1+\mu_2}\right)^{r_1}\right) \\
&\qquad \times\cdots\left(\sum_{r_{n_A-\ell}=0}^{\infty}\left(\frac{\lambda_R}{\mu_1+\mu_2}\right)^{r_{n_A-\ell}}\right) \\
&\quad = C_{\mathbb{N}}\left(\frac{\lambda_R}{\mu_2}\right)\sum_{\ell=0}^{n_A-1}\left(\frac{\lambda_A}{\mu_2}\right)^{\ell}\left(\frac{\lambda_A}{\mu_1+\mu_2}\right)^{n_A-\ell}\left(\frac{1}{1-\frac{\lambda_R}{\mu_1+\mu_2}}\right)^{n_A-\ell} \\
&\quad = C_{\mathbb{N}}\left(\frac{\lambda_R}{\mu_2}\right)\sum_{\ell=0}^{n_A-1}\left(\frac{\lambda_A}{\mu_2}\right)^{\ell}\left(\frac{\lambda_A}{\mu_1+\mu_2-\lambda_R}\right)^{n_A-\ell} \\
&\quad = C_{\mathbb{N}}\left(\frac{\lambda_R}{\mu_2}\right)\frac{\mu_2\left[\left(\frac{\lambda_A}{\mu_2}\right)^{n_A}-\left(\frac{\lambda_A}{\mu_1+\mu_2-\lambda_R}\right)^{n_A}\right]}{\mu_1-\lambda_R} \\
&\quad = C_{\mathbb{N}}\left(\frac{\lambda_R}{\mu_1-\lambda_R}\right)\left[\left(\frac{\lambda_A}{\mu_2}\right)^{n_A}-\left(\frac{\lambda_A}{\mu_1+\mu_2-\lambda_R}\right)^{n_A}\right]. \qquad (12)
\end{aligned}
$$

Finally, we add (10), (11), and (12) to get the result in (5).

We now obtain the Laplace transform of the response time for class A jobs, $\widetilde{T}_A(s)$, via distributional Little's Law [22]. First, we find the z-transform of the number of class A jobs in the system, $\widehat{N}_A(z)$:

$$\begin{aligned}\widehat{N}_A(z) &= \sum_{i=0}^{\infty} \Pr\{N_A = i\} z^i \\ &= \zeta_{\mathbb{N}1} \sum_{i=0}^{\infty} \left(\frac{\lambda_A}{\mu_2}\right)^i z^i + \zeta_{\mathbb{N}2} \sum_{i=0}^{\infty} \left(\frac{\lambda_A}{\mu_1+\mu_2-\lambda_R}\right)^i z^i \\ &= \frac{\zeta_{\mathbb{N}1}\mu_2}{\mu_2-\lambda_A z} + \frac{\zeta_{\mathbb{N}2}(\mu_1+\mu_2-\lambda_R)}{\mu_1+\mu_2-\lambda_R-\lambda_A z}.\end{aligned}$$

Observe that class A jobs depart the system in the same order in which they arrive, so A_{T_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 $\widehat{A}_{T_A}(z) = \widetilde{T}_A(\lambda_A - \lambda_A z)$, we have

$$\begin{aligned}\widetilde{T}_A(\lambda_A - \lambda_A z) &= \widehat{N}_A(z) \\ &= \frac{\zeta_{\mathbb{N}1}\mu_2}{\mu_2-\lambda_A z} + \frac{\zeta_{\mathbb{N}2}(\mu_1+\mu_2-\lambda_R)}{\mu_1+\mu_2-\lambda_R-\lambda_A z}.\end{aligned}$$

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

$$\begin{aligned}\widetilde{T}_A(s) &= \frac{\zeta_{\mathbb{N}1}\mu_2}{\mu_2-\lambda_A(1-\frac{s}{\lambda_A})} + \frac{\zeta_{\mathbb{N}2}(\mu_1+\mu_2-\lambda_R)}{\mu_1+\mu_2-\lambda_R-\lambda_A(1-\frac{s}{\lambda_A})} \\ &= (1+\omega_{\mathbb{N}})\frac{\nu_{\mathbb{N}1}}{\nu_{\mathbb{N}1}+s} - \omega_{\mathbb{N}}\frac{\nu_{\mathbb{N}2}}{\nu_{\mathbb{N}2}+s}.\end{aligned}$$

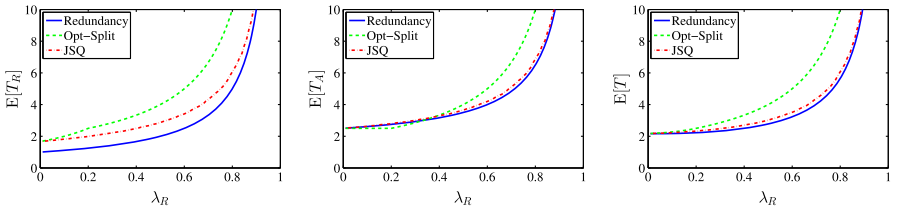
This is the transform of a generalized hyperexponential distribution, $H_2(\nu_{\mathbb{N}1}, \nu_{\mathbb{N}2}, \omega_{\mathbb{N}})$. Finally,

$$\begin{aligned}\mathbb{E}[T_A] &= -\widetilde{T}_A'(s)|_{s=0} \\ &= -\left[(1+\omega_{\mathbb{N}})\frac{-\nu_{\mathbb{N}1}}{(\nu_{\mathbb{N}1}+s)^2} + \omega_{\mathbb{N}}\frac{\nu_{\mathbb{N}2}}{(\nu_{\mathbb{N}2}+s)^2}\right]\Bigg|_{s=0} \\ &= \frac{1}{\nu_{\mathbb{N}1}} + \frac{1}{\nu_{\mathbb{N}2}} - \frac{1}{\nu_{\mathbb{N}3}}.\end{aligned}$$

□

Proof (Theorem 4) Definition 1 gives us $\mathbb{E}[T_R]^{\text{Opt}-\text{Split}}$, $\mathbb{E}[T_A]^{\text{Opt}-\text{Split}}$, and $\mathbb{E}[T]^{\text{Opt}-\text{Split}}$. We know $\mathbb{E}[T_A]^{\text{Redundant}}$ from Theorem 3. Theorem 2 tells us that

$$T_R^{\text{Redundant}} \sim \text{Exp}(\mu_1 + \mu_2 - \lambda_A - \lambda_R),$$



(a) $\mathbb{E}[T_R]$ **(b)** $\mathbb{E}[T_A]$ **(c)** $\mathbb{E}[T]$

Fig. 6 Comparing redundancy (*solid blue*), Opt-Split (*dashed green*), and JSQ (*dashed red*) for the $\mathbb{W}$ model as λ_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 μ_1 and μ_2 are similar (Color figure online)

so we know that $\mathbb{E}[T_R]^{\text{Redundant}} = \frac{1}{\mu_1+\mu_2-\lambda_A-\lambda_R}$. Finally,

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

Thus, $\frac{\mathbb{E}[T_R]^{\text{Redundant}}}{\mathbb{E}[T_R]^{\text{Opt-Split}}}$, $\frac{\mathbb{E}[T_A]^{\text{Redundant}}}{\mathbb{E}[T_A]^{\text{Opt-Split}}}$, and $\frac{\mathbb{E}[T]^{\text{Redundant}}}{\mathbb{E}[T]^{\text{Opt-Split}}}$, and the desired results follow after some minor algebra. □

5 The $\mathbb{W}$ model

We now consider the $\mathbb{W}$ model (see Fig. 2b). The $\mathbb{W}$ 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 $\mathbb{W}$ model.

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

$$\pi_{(c_n,\ldots,A)} = C_{\mathbb{W}}\left(\frac{\lambda_A}{\mu_1}\right)^{a_0}\left(\frac{\lambda_A}{\mu_1+\mu_2}\right)^{a_1}\left(\frac{\lambda_B}{\mu_1+\mu_2}\right)^{b_1}\left(\frac{\lambda_R}{\mu_1+\mu_2}\right)^{r}$$
$$\pi_{(c_n,\ldots,B)} = C_{\mathbb{W}}\left(\frac{\lambda_B}{\mu_2}\right)^{b_0}\left(\frac{\lambda_A}{\mu_1+\mu_2}\right)^{a_1}\left(\frac{\lambda_B}{\mu_1+\mu_2}\right)^{b_1}\left(\frac{\lambda_R}{\mu_1+\mu_2}\right)^{r}$$
$$\pi_{(c_n,\ldots,R)} = C_{\mathbb{W}}\left(\frac{\lambda_A}{\mu_1+\mu_2}\right)^{a_1}\left(\frac{\lambda_B}{\mu_1+\mu_2}\right)^{b_1}\left(\frac{\lambda_R}{\mu_1+\mu_2}\right)^{r},$$

where a_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

$$C_{\mathbb{W}} = \left(1 - \frac{\lambda_A}{\mu_1}\right)\left(1 - \frac{\lambda_B}{\mu_2}\right)\left(1 - \frac{\lambda_R}{\mu_1 + \mu_2 - \lambda_A - \lambda_B}\right)$$

is a normalizing constant.

Like in the $\mathbb{N}$ 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 queue. Nonetheless, the response time for the redundant class is stochastically equivalent to the response time in an M/M/1 queue with arrival rate λ_R and service rate $\mu' = \mu_1 + \mu_2 - \lambda_A - \lambda_B$. We can interpret μ' as the remaining service capacity in the system after λ_A and λ_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 queue with arrival rate $\lambda_A + \lambda_B + \lambda_R$ and service rate $\mu_1 + \mu_2$.

Theorem 5 *In the $\mathbb{W}$ model,*

1. *The number of class R jobs in the system, N_R, is distributed as* Geometric$(1-\rho)-1$, *where $\rho = \frac{\lambda_R}{\mu_1+\mu_2-\lambda_A-\lambda_B}$.*
2. *The response time of class R jobs, T_R, is distributed as* Exp$(\mu_1 + \mu_2 - \lambda_A - \lambda_B - \lambda_R)$.

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).

Theorem 6 *In the $\mathbb{W}$ model,*

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

$$\Pr\{N_A = n_A\} = \zeta_{\mathbb{W}1}\left(\frac{\lambda_A}{\mu_1}\right)^{n_A} + \zeta_{\mathbb{W}2}\left(\frac{\lambda_A}{\mu_1 + \mu_2 - \lambda_B - \lambda_R}\right)^{n_A},$$

where

$$\zeta_{\mathbb{W}1} = \frac{(\mu_1 - \lambda_A)(\mu_2 - \lambda_B)(\mu_1 + \mu_2 - \lambda_A - \lambda_B - \lambda_R)}{\mu_1(\mu_2 - \lambda_B - \lambda_R)(\mu_1 + \mu_2 - \lambda_A - \lambda_B)},$$
$$\zeta_{\mathbb{W}2} = \frac{-\lambda_R(\mu_1 - \lambda_A)(\mu_1 + \mu_2 - \lambda_A - \lambda_B - \lambda_R)}{(\mu_2 - \lambda_B - \lambda_R)(\mu_1 + \mu_2 - \lambda_A - \lambda_B)}.$$

2. *The distribution of response time of class A jobs is*

$$T_A \sim H_2(\nu_{\mathbb{W}1}, \nu_{\mathbb{W}2}, \omega_{\mathbb{W}}),$$

where

$$\nu_{\mathbb{W}1} = \mu_1 - \lambda_A,$$
$$\nu_{\mathbb{W}2} = \mu_1 + \mu_2 - \lambda_A - \lambda_B - \lambda_R,$$
$$\nu_{\mathbb{W}3} = \mu_1 + \mu_2 - \lambda_A - \lambda_B,$$

$$\omega_{\mathbb{W}} = \frac{(\mu_2 - \lambda_B)\nu_{\mathbb{W}2}}{(\mu_2 - \lambda_B - \lambda_R)\nu_{\mathbb{W}3}}.$$

The mean response time of class A jobs is

$$\mathbb{E}[T_A] = \underbrace{\frac{1}{\nu_{\mathbb{W}1}}}_{\text{M/M/1}} + \underbrace{\frac{1}{\nu_{\mathbb{W}2}} - \frac{1}{\nu_{\mathbb{W}3}}}_{\text{penalty}}, \tag{13}$$

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

Like in the $\mathbb{N}$ model, we find that T_A follows a generalized hyperexponential distribution. The mean response time of class A (or class B) jobs can be interpreted as that in an M/M/1 queue with arrival rate λ_A (respectively, λ_B) and service rate μ_1 (respectively, μ_2), plus a penalty term that captures the extent to which the redundant class hurts the As (or Bs) (Eq. 13). 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. The intuition behind this surprising result comes from the fact that, like in the $\mathbb{N}$ model, the distribution of response time for class A can be rewritten as the distribution of time in queue in an M/M/1 queue with arrival rate λ_R and service rate $\mu_1 + \mu_2 - \lambda_A - \lambda_B - \lambda_R$, plus the response time in an M/M/1 queue with arrival rate λ_A and service rate μ_1 (class B is symmetric). That is, both classes experience the queueing time in an M/M/1 queue with arrival rate λ_R and service rate $\mu_1 + \mu_2 - \lambda_A - \lambda_B - \lambda_R$: the mean of this distribution is exactly the penalty term incurred by both class A and class B.

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 Fig. 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 (Fig. 6a).

For the non-redundant classes (Fig. 6b), 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 $\mathbb{N}$ model, redundancy is always better for the overall system (Fig. 6c).

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, a fraction p of class R jobs go to server 1, and a fraction $1 - p$ go to server 2, where p is chosen to minimize $\mathbb{E}[T]$. The mean response times

for class R jobs, class A jobs, and the overall system are:

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

$$\mathbb{E}[T_A]^{\text{Opt}-\text{Split}} = \frac{1}{\mu_1 - \lambda_A - p\lambda_R},$$

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

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. $\frac{1}{2} \le \frac{\mathbb{E}[T_R]^{\text{Redundant}}}{\mathbb{E}[T_R]^{\text{Opt}-\text{Split}}} \le 1$. *If $\lambda_R \ge |\lambda_A - \lambda_B|$, then* $\frac{\mathbb{E}[T_R]^{\text{Redundant}}}{\mathbb{E}[T_R]^{\text{Opt}-\text{Split}}} = \frac{1}{2}$.
2. $\frac{1}{2} \le \frac{\mathbb{E}[T_A]^{\text{Redundant}}}{\mathbb{E}[T_A]^{\text{Opt}-\text{Split}}} \le \frac{3}{2}$. *The same inequality holds for class B.*
3. $\frac{1}{2} \le \frac{\mathbb{E}[T]^{\text{Redundant}}}{\mathbb{E}[T]^{\text{Opt}-\text{Split}}} \le 1$.

Proof We have $\mathbb{E}[T_R]^{\text{Opt}-\text{Split}}$, $\mathbb{E}[T_A]^{\text{Opt}-\text{Split}}$, and $\mathbb{E}[T]^{\text{Opt}-\text{Split}}$ from Definition 2. We also know $\mathbb{E}[T_A]^{\text{Redundant}}$ from Theorem 6. Theorem 5 tells us that

$$T_R^{\text{Redundant}} \sim \text{Exp}(\mu_1 + \mu_2 - \lambda_A - \lambda_B - \lambda_R),$$

so

$$\mathbb{E}[T_R]^{\text{Redundant}} = \frac{1}{\mu_1 + \mu_2 - \lambda_A - \lambda_B - \lambda_R}.$$

Finally,

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

We use these expressions to find $\frac{\mathbb{E}[T_R]^{\text{Redundant}}}{\mathbb{E}[T_R]^{\text{Opt}-\text{Split}}}$, $\frac{\mathbb{E}[T_A]^{\text{Redundant}}}{\mathbb{E}[T_A]^{\text{Opt}-\text{Split}}}$, and $\frac{\mathbb{E}[T]^{\text{Redundant}}}{\mathbb{E}[T]^{\text{Opt}-\text{Split}}}$, and the desired results follow after some minor algebra. □

6 The $\mathbb{M}$ model

Finally, we consider the $\mathbb{M}$ model (Fig. 2c). 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

$$\begin{aligned}\mu_{1,2,3} &= \mu_1 + \mu_2 + \mu_3,\\ \mu_{1,2} &= \mu_1 + \mu_2,\\ \mu_{2,3} &= \mu_2 + \mu_3.\end{aligned}$$

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:*

$$\begin{aligned}\pi_{(c_n,\ldots,R_1)} &= C_{\mathbb{M}} \left(\frac{\lambda_{R_1}}{\mu_{1,2}}\right)^{r_{1,0}} \left(\frac{\lambda_{R_1}}{\mu_{1,2,3}}\right)^{r_{1,1}} \left(\frac{\lambda_{R_2}}{\mu_{1,2,3}}\right)^{r_{2,1}},\\ \pi_{(c_n,\ldots,R_2)} &= C_{\mathbb{M}} \left(\frac{\lambda_{R_2}}{\mu_{2,3}}\right)^{r_{2,0}} \left(\frac{\lambda_{R_1}}{\mu_{1,2,3}}\right)^{r_{1,1}} \left(\frac{\lambda_{R_2}}{\mu_{1,2,3}}\right)^{r_{2,1}},\end{aligned}$$

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 R_2 (R_1) job, and

$$C_{\mathbb{M}} = \frac{(\mu_{1,2} - \lambda_{R_1})(\mu_{1,2,3} - \lambda_{R_1} - \lambda_{R_2})(\mu_{2,3} - \lambda_{R_2})}{\mu_{1,2}\mu_{2,3}(\mu_{1,2,3} - \lambda_{R_1} - \lambda_{R_2}) + \lambda_{R_1}\lambda_{R_2}\mu_2}$$

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 an 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.*

$$\Pr\{N_{R_1} = n\} = \zeta_{\mathbb{M}1} \left(\frac{\lambda_{R_1}}{\mu_{1,2}}\right)^n + \zeta_{\mathbb{M}2} \left(\frac{\lambda_{R_1}}{\mu_{1,2,3} - \lambda_{R_2}}\right)^n,$$

where

$$\zeta_{\mathbb{M}1} = C_{\mathbb{M}} \frac{\mu_3}{\mu_3 - \lambda_{R_2}},$$

$$\zeta_{\mathbb{M}2} = C_{\mathbb{M}} \left(\frac{\lambda_{R_2}}{\mu_{2,3} - \lambda_{R_2}} - \frac{\lambda_{R_1}}{\mu_{1,2,3} - \lambda_{R_2}} \right).$$

2. *The distribution of response time of class R_1 jobs is*

$$T_{R_1} \sim H_2(\nu_{\mathbb{M}1}, \nu_{\mathbb{M}2}, \omega_{\mathbb{M}}),$$

where

$$\begin{aligned} \nu_{\mathbb{M}1} &= \mu_{1,2} - \lambda_{R_1}, \\ \nu_{\mathbb{M}2} &= \mu_{1,2,3} - \lambda_{R_1} - \lambda_{R_2}, \\ \omega_{\mathbb{M}} &= \zeta_{\mathbb{M}1} \frac{\mu_{1,2}}{\mu_{1,2} - \lambda_{R_1}}. \end{aligned}$$

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

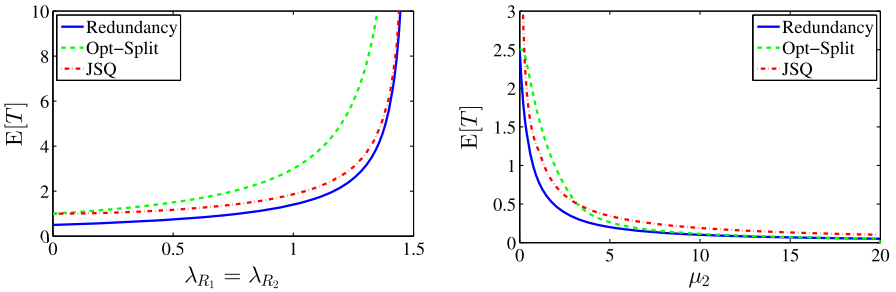
Both classes obviously benefit from issuing redundant requests on a shared server rather than each class having a single dedicated server. However, one might wonder whether mean response time could be further reduced using some other policy, like Opt-Split or JSQ, instead of redundancy. In Fig. 7 we 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, a fraction p of class R_1 jobs go to server 2, and a fraction $1 - p$ go to server 1; and a fraction q of class R_2 jobs go to server 2, and a fraction $1 - q$ go to server 3. We choose p and q to minimize the overall mean response time, given by

$$\begin{aligned} \mathbb{E}[T]^{\text{Opt-Split}} = &\frac{(1-p)\lambda_{R_1}}{\lambda_{R_1} + \lambda_{R_2}} \cdot \frac{1}{\mu_1 - (1-p)\lambda_{R_1}} + \frac{p\lambda_{R_1} + q\lambda_{R_2}}{\lambda_{R_1} + \lambda_{R_2}} \cdot \frac{1}{\mu_2 - p\lambda_{R_1} - q\lambda_{R_2}} \\ &+ \frac{(1-q)\lambda_{R_2}}{\lambda_{R_1} + \lambda_{R_2}} \cdot \frac{1}{\mu_3 - (1-q)\lambda_{R_2}}. \end{aligned}$$

In all cases, redundancy outperforms both Opt-Split and JSQ. For homogeneous servers (Fig. 7a), 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 (Fig. 7b), 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:



(a) $\mu_1 = \mu_2 = \mu_3 = 1$ **(b)** $\mu_1 = \mu_3 = 1$, $\lambda_{R_1} = \lambda_{R_2} = 0.6$.

Fig. 7 Comparing redundancy, Opt-Split, and JSQ for the $\mathbb{M}$ 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 $\lambda_{R_1} = \lambda_{R_2}$, **b** increasing μ_2 (Color figure online)

Theorem 9 *In the $\mathbb{M}$ model, for any μ_1, μ_2, μ_3, λ_{R_1}, and λ_{R_2} such that the system is stable,*

1. *If $\mu_1 = \mu_2 = \mu_3$ and $\lambda_{R_1} = \lambda_{R_2}$, $\frac{1}{3} \leq \frac{\mathbb{E}[T]^{\text{Redundant}}}{\mathbb{E}[T]^{\text{Opt-Split}}} \leq \frac{1}{2}$.*
2. *For finite μ_1 and μ_3, $\lim_{\mu_2 \to \infty} \frac{\mathbb{E}[T]^{\text{Redundant}}}{\mathbb{E}[T]^{\text{Opt-Split}}} = 1$.*
3. *If $\mu_1 = \mu_3$, $\frac{\mathbb{E}[T]^{\text{Redundant}}}{\mathbb{E}[T]^{\text{Opt-Split}}} \leq 1$.*

Proof We know $\mathbb{E}[T]^{\text{Opt-Split}}$ from Definition 3, and

$$\mathbb{E}[T]^{\text{Redundant}} = \frac{\lambda_{R_1}}{\lambda_{R_1} + \lambda_{R_2}} \mathbb{E}[T_{R_1}]^{\text{Redundant}} + \frac{\lambda_{R_2}}{\lambda_{R_1} + \lambda_{R_2}} \mathbb{E}[T_{R_2}]^{\text{Redundant}},$$

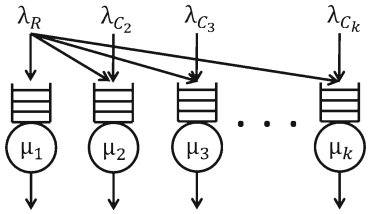
where we know $\mathbb{E}[T_{R_1}]^{\text{Redundant}}$ and $\mathbb{E}[T_{R_2}]^{\text{Redundant}}$ from Theorem 8. We can then write $\frac{\mathbb{E}[T]^{\text{Redundant}}}{\mathbb{E}[T]^{\text{Opt-Split}}}$, and the desired results follow after some minor algebra. □

7 Scale

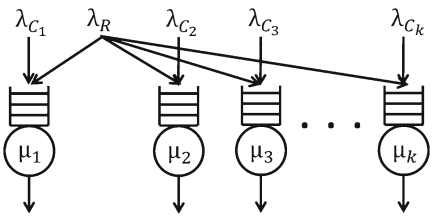
Thus far, we only have considered systems with two servers (the $\mathbb{N}$ and $\mathbb{W}$ models) and three servers (the $\mathbb{M}$ model). We now turn our attention to the question of scale.

The scaled $\mathbb{N}$, $\mathbb{W}$, and $\mathbb{M}$ models are shown in Fig. 8. In the scaled $\mathbb{N}$ model there are k servers and k classes of jobs (see Fig. 8a). 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 $\mathbb{W}$ 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 Fig. 8b). In the scaled $\mathbb{M}$ 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 Fig. 8c).

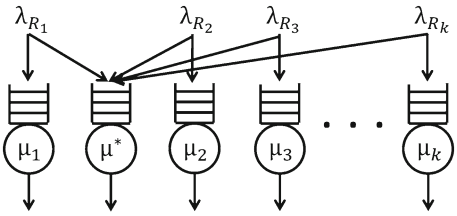
The limiting probabilities derived in Theorem 1 for the general redundancy system apply to the scaled $\mathbb{N}$, $\mathbb{W}$, and $\mathbb{M}$ models. In Theorem 10 we use this result to find that in both the scaled $\mathbb{N}$ and $\mathbb{W}$ models, response time for class R is exponentially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



(a) Scaled $\mathbb{N}$ model

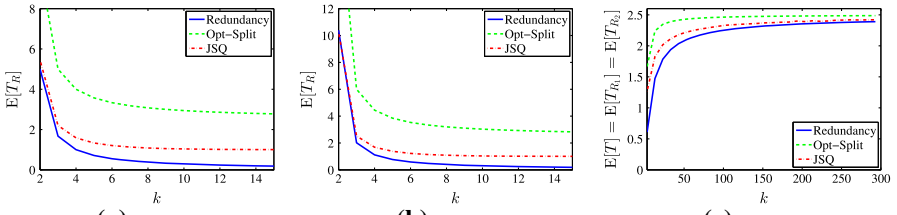


(b) Scaled $\mathbb{W}$ model



(c) Scaled $\mathbb{M}$ model

Fig. 8 Scaled versions of **a** the $\mathbb{N}$ model, **b** the $\mathbb{W}$ model, and **c** the $\mathbb{M}$ model



(a) Scaled $\mathbb{N}$ **(b)** Scaled $\mathbb{W}$ **(c)** Scaled $\mathbb{M}$

Fig. 9 Comparing $\mathbb{E}[T_R]$ under redundancy (*solid blue*), Opt-Split (*dashed green*), and JSQ (*dashed red*) in scaled systems with homogeneous servers, all with rate 1. **a** The scaled $\mathbb{N}$ model with $\lambda_{C_i} = 0.6$ for all non-redundant classes, and $\lambda_R = 1.2$. **b** The scaled $\mathbb{W}$ model with $\lambda_{C_i} = 0.6$ for all non-redundant classes, and $\lambda_R = 0.7$. **c** The scaled $\mathbb{M}$ model with $\lambda_{R_i} = 0.6$ for all classes (Color figure online)

Theorem 10

1. *In the scaled $\mathbb{N}$ 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 $\mathbb{W}$ 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 Deferred to the end of the section. □

For the $\mathbb{M}$ model and for the non-redundant classes in the $\mathbb{N}$ and $\mathbb{W}$ 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 $\mathbb{N}$ and $\mathbb{W}$ 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 $\mathbb{N}$ and $\mathbb{W}$ models (see Fig. 9a, 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 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 $\frac{1}{\sum_i \mu_i}$.

In the two-server $\mathbb{N}$ and $\mathbb{W}$ 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 $\mathbb{N}$ and $\mathbb{W}$ 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 queue with arrival rate λ_{C_i} and service rate μ_i.

In the three-server version of the $\mathbb{M}$ model (Sect. 6), we saw that redundancy significantly outperformed Opt-Split and JSQ. In Fig. 9c, we look at the relative performance of the three policies as k increases. In the scaled $\mathbb{M}$ 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.

7.1 Proof of Theorem 10

Proof (Theorem 10) To find $\Pr\{N_R = n_R\}$ in the $\mathbb{N}$ model, we will consider the non-R jobs in the queue as being split into two pieces: the non-R jobs before the first R in the queue, and the non-R jobs after the first R in the queue. We sum over all possible lengths of these two pieces, and all possible classes of these non-R jobs. Let x_0 be the number of non-R jobs before the first R in the queue, and let x_1 be the number of non-R jobs after the first R in the queue. Then we have

$$\begin{aligned}\Pr\{N_R = n_R\} &= \sum_{x_0=0}^{\infty}\sum_{x_1=0}^{\infty} C\eta_R^{n_R} X_0 \binom{x_1 + n_R - 1}{x_1} \prod_{\substack{j \geq x_0 \\ c_j \neq R}} X_1 \\ &= C\eta_R^{n_R}\left(\sum_{x_0=0}^{\infty} X_0\right)\left(\sum_{x_1=0}^{\infty} X_1^{x_1}\binom{x_1 + n_R - 1}{x_1}\right),\end{aligned}$$

where

$$\begin{aligned}\eta_R &= \frac{\lambda_R}{\sum_{m=1}^{k}\mu_m}, \\ X_0 &= \prod_{j=1}^{x_0}\frac{\sum_{c_i \neq R}\lambda_{c_i}}{\sum_{m \in \bigcup_{t \leq j} S_t}\mu_m}, \\ X_1 &= \frac{\sum_{c_i \neq R}\lambda_{c_i}}{\sum_{m=1}^{k}\mu_m}.\end{aligned}$$

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 $C_1 = \sum_{x_0=0}^{\infty} X_0$ (note that this is a constant with respect to n_R). Using the identity $\sum_{i=0}^{\infty} p^i \binom{i+n-1}{i} = \left(\frac{1}{1-p}\right)^n$ for $|p| < 1$, we have

$$\Pr\{N_R = n_R\} = CC_1 \eta_R^{n_R} \left(\frac{1}{1 - \frac{\sum_{c_i \neq R} \lambda_{c_i}}{\sum_m \mu_m}} \right)^{n_R}$$
$$= CC_1 \left(\frac{\lambda_R}{\sum_m \mu_m - \sum_{c_i \neq R} \lambda_{c_i}} \right)^{n_R}.$$

Using the normalization equation

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

we find

$$CC_1 = 1 - \frac{\lambda_R}{\sum_m \mu_m - \sum_{c_i \neq R} \lambda_{c_i}}.$$

Hence $N_R \sim \text{Geometric}\left(1 - \frac{\lambda_R}{\sum_m \mu_m - \sum_{c_i \neq R} \lambda_{c_i}}\right) - 1$.

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

$$\widetilde{T}_R(\lambda_R - \lambda_R z) = \widehat{N}_R(z).$$

We know that N_R is distributed as Geometric $(p) - 1$, where $p = 1 - \frac{\lambda_R}{\sum_m \mu_m - \sum_{c_i \neq R} \lambda_{c_i}}$. Hence we have

$$\widetilde{T}_R(\lambda_R - \lambda_R z) = \widehat{N}_R(z) = \frac{p}{1 - z(1-p)}.$$

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

$$\widetilde{T}_R(s) = \frac{p}{1 - \left(1 - \frac{s}{\lambda_R}\right)(1-p)},$$

which after some simplification gives

$$\widetilde{T}_R(s) = \frac{\sum_m \mu_m - \sum_{c_i \neq R} \lambda_{c_i} - \lambda_R}{\sum_m \mu_m - \sum_{c_i \neq R} \lambda_{c_i} - \lambda_R + s}.$$

Hence $T_R \sim \text{Exp}\left(\sum_m \mu_m - \sum_{c_i \neq R} \lambda_{c_i} - \lambda_R\right)$.

The derivation for the $\mathbb{W}$ model is very similar, and is omitted here. □

8 Relaxing assumptions

Thus far, we have assumed that service times are exponentially distributed and that when a job runs on multiple servers its service times are independent on the different servers. The independence assumption is appropriate in situations in which server-dependent factors such as network congestion, background load, and disk seek times dominate a job's actual computation time. For example, suppose that each server is actually a virtual machine (VM) running in the cloud, serving web queries. A VM running on one physical machine is completely independent from a VM running on a different physical machine. While a VM running on one physical machine might experience slowdown factors of up to 27 due to other traffic on its physical machine [38], this does not affect a VM running on a different physical machine, which may experience a slowdown factor of 1. In such cases, a job's running time depends primarily on the physical machine on which it runs, rather than on the job itself.

It is less clear that service times should be exponentially distributed. If the dominating component of service time is a disk seek time, for example, then the service times will have low variability. On the other hand, factors such as network congestion can lead to highly variable service times. In this section we consider the impact of redundancy when service times are more or less variable than the exponential. Unfortunately our exact analysis does not apply to non-exponential service times, hence we study this question via simulation.

The effect of service time variability on mean response time is very complicated. Recall that redundancy helps for two reasons: by allowing jobs to experience the minimum queueing time across servers, and by allowing jobs to experience the minimum service time across servers. Changing the service time variability reveals a tradeoff between these two factors. When service time variability is high, the redundant jobs experience a bigger service time benefit (which likewise helps the non-redundant jobs), but queueing times increase because a large non-redundant job can block a server for a long period of time. When service time variability is lower, queueing times are much lower, but redundant jobs experience less of a service time benefit and can even waste server capacity.

Figure 10 shows mean response time in the $\mathbb{N}$ model for the redundant jobs, non-redundant jobs, and overall system as λ_R increases. For the non-redundant jobs

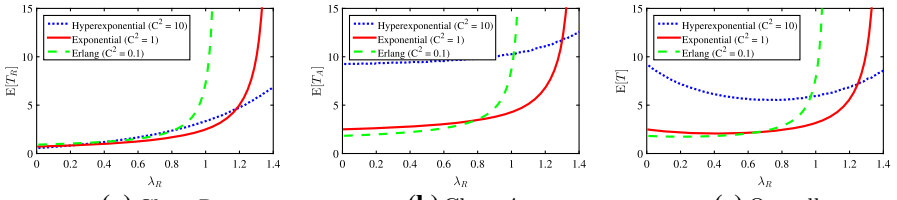


Fig. 10 Mean response time in the $\mathbb{N}$ model for **a** redundant jobs, **b** non-redundant jobs, and **c** the overall system as a function of λ_R when $\lambda_A = 0.6$ and mean service time is 1. Lines shown include exponential service times (*solid red line*), hyperexponential service times with $C^2 = 10$ (*dotted blue line*), and Erlang service times with $C^2 = 0.1$ (*dashed green line*) (Color figure online)

(Fig. 10b), at low load the queueing time effect is most pronounced: higher variability service times lead to higher mean response times. This is because non-redundant jobs can experience very long queueing times due to waiting behind other non-redundant jobs with long running times; the benefit of clearing the redundant jobs out of the system more quickly is insufficient to overcome this effect. For the redundant jobs (Fig. 10a), at most values of λ_R mean response time is lowest under exponentially distributed service times. This is because the exponential service times provide the right balance between reducing queueing time (for which we want low service time variability) and reducing service time (for which we want high service time variability). It is only at very high load that the service time benefit of high variability service times starts to dominate; when load becomes very high this benefit is enough to increase the system's stability region, benefiting both the redundant and non-redundant jobs.

9 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 queue, even though the system is not an M/M/1 queue ($\mathbb{N}$ and $\mathbb{W}$ 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 ($\mathbb{N}$ and $\mathbb{W}$ model).
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 ($\mathbb{N}$ and $\mathbb{W}$ models).
4. Given two classes of jobs, A and B, each with its own queue, if 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 ($\mathbb{W}$ 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 ($\mathbb{M}$ 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 $\mathbb{N}$ and scaled $\mathbb{W}$ models).
7. When service times are highly variable, redundancy yields an even bigger improvement, even increasing the system's stability region.

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.

Funding This material is based upon work supported by the National Science Foundation Graduate Research Fellowship under Grant No. DGE-1252522; was funded by NSF-CMMI-1334194, NSF-CSR-1116282, and NSF-CMMI-1538204, 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 FQ4BD and TOP-Energy projects (Grant Nos. 296206 and 268992).

Compliance with ethical standards

Conflict of interest The authors declare that they have no conflicts of interest.

References

1. Adan, I., Weiss, G.: A skill based parallel service system under FCFS-ALIS—steady state, overloads, and abandonments. Stoch. Syst. **4**(1), 250–299 (2014)
2. Ananthanarayanan, G., Ghodsi, A., Shenker, S., Stoica, I.: Effective straggler mitigation: attack of the clones. In *NSDI*, pp. 185–198, April 2013
3. Ananthanarayanan, G., Hung, M.C.C., Ren, X., Stoica, I., Wierman, A., Yu, M.: Grass: trimming stragglers in approximation analytics. In *Proceedings of the 11th USENIX Conference on Networked Systems Design and Implementation*, pp. 289–302. USENIX Association, 2014
4. Baccelli, F., Makowski, A.: Simple computable bounds for the fork-join queue. Technical Report RR-0394, Inria, 1985
5. Baccelli, F., Makowski, A.M., Shwartz, A.: The fork-join queue and related systems with synchronization constraints: stochastic ordering and computable bounds. Adv. Appl. Prob. **21**, 629–660 (1989)
6. Bassamboo, A., Randhawa, R.S., Mieghem, J.A.V.: A little flexibility is all you need: on the value of flexible resources in queueing systems. Oper. Res. **60**, 1423–1435 (2012)
7. Borst, S., Boxma, O., Uitert, M.V.: The asymptotic workload behavior of two coupled queues. Queueing Syst. **43**(1–2), 81–102 (2003)
8. Botta, R.F., Harris, C.M., Marchal, W.G.: Characterizations of generalized hyperexponential distribution functions. Commun. Stat. Stoch. Models **3**(1), 115–148 (1987)
9. Boxma, O., Koole, G., Liu, Z.: Queueing-theoretic solution methods for models of parallel and distributed systems. In *Performance Evaluation of Parallel and Distributed Systems Solution Methods. CWI Tract 105 & 106*, pp. 1–24, 1994
10. Casanova, H.: Benefits and drawbacks of redundant batch requests. J. Grid Comput. **5**(2), 235–250 (2007)
11. Cohen, J.W., Boxma, O.J.: Boundary Value Problems in Queueing System Analysis. North-Holland Publishing Company, Amsterdam (1983)
12. Dean, J., Barroso, L.A.: The tail at scale. Commun. ACM **56**(2), 74–80 (2013)
13. Fayolle, G., Iasnogorodski, R.: Two coupled processors: the reduction to a Riemann-Hilbert problem. Zeitschrift fur Wahrscheinlichkeitstheorie und vervandte Gebiete **47**(3), 325–351 (1979)
14. Flatto, L.: Two parallel queues created by arrivals with two demands II. SIAM J. Appl. Math. **45**(5), 1159–1166 (1985)
15. Flatto, L., Hahn, S.: Two parallel queues created by arrivals with two demands I. SIAM J. Appl. Math. **44**(5), 250–255 (1984)
16. Hall, P.: On representatives of subsets. J. Lond. Math. Soc. **10**(1), 26–30 (1935)
17. Harchol-Balter, M., Li, C., Osogami, T., Scheller-Wolf, A., Squillante, M.: Cycle stealing under immediate dispatch task assignment. In *Annual Symposium on Parallel Algorithms and Architectures*, pp. 274–285, June 2003

18. Hooghiemstra, G., Keane, M., de Ree, S.V.: Power series for stationary distributions of coupled processor models. SIAM J. Appl. Math. **48**(5), 861–878 (1988)
19. Huang, H., Hung, W., Shin, K.G.: FS2: dynamic data replication in free disk space for improving disk performance and energy consumption. In *Proceedings of the SOSP'05*, pp. 263–276, December 2005
20. Joshi, G., Liu, Y., Soljanin, E.: Coding for fast content download. In *Allerton Conference'12*, pp. 326–333, 2012
21. Joshi, G., Liu, Y., Soljanin, E.: On the delay-storage trade-off in content download from coded distributed storage systems. IEEE J. Sel. Areas Commun. **32**(5), 989–997 (2014)
22. Keilson, J., Servi, L.: A distributional form of Little's Law. Oper. Res. Lett. **7**(5), 223–227 (1988)
23. Kim, C., Agrawala, A.K.: Analysis of the fork-join queue. IEEE Trans. Comput. **38**(2), 1041–1053 (1989)
24. Konheim, A.G., Meilijson, I., Melkman, A.: Processor-sharing of two parallel lines. J. Appl. Prob. **18**(4), 952–956 (1981)
25. Koole, G., Righter, R.: Resource allocation in grid computing. J. Sched. **11**, 163–173 (2009)
26. Nelson, R., Tantawi, A.N.: Approximate analysis of fork/join synchronization in parallel queues. IEEE Trans. Comput. **37**(6), 739–743 (1988)
27. Osogami, T., Harchol-Balter, M., Scheller-Wolf, A.: Analysis of cycle stealing with switching times and thresholds. In *SIGMETRICS*, pp. 184–195, June 2003
28. Ren, X., Ananthanarayanan, G., Wierman, A., Yu, M.: Hopper: decentralized speculation-aware cluster scheduling at scale
29. Shah, N.B., Lee, K., Ramchandran, K.: The MDS queue: analysing latency performance of codes and redundant requests. Technical Report arXiv:1211.5405, November 2012
30. Shah, N.B., Lee, K., Ramchandran, K.: When do redundant requests reduce latency? Technical Report arXiv:1311.2851, June 2013
31. Stolyar, A.L., Tezcan, T.: Control of systems with flexible multi-server pools: a shadow routing approach. Queueing Syst. **66**, 1–51 (2010)
32. Tsitsiklis, J., Xu, K.: On the power of (even a little) resource pooling. Stoch. Syst. **2**, 1–66 (2012)
33. Tsitsiklis, J., Xu, K.: Queueing system topologies with limited flexibility. In *SIGMETRICS*, 2013
34. Visschers, J., Adan, I., Weiss, G.: A product form solution to a system with multi-type jobs and multi-type servers. Queueing Syst. **70**, 269–298 (2012)
35. Vulimiri, A., Godfrey, P.B., Mittal, R., Sherry, J., Ratnasamy, S., Shenker, S.: Low latency via redundancy. In *CoNEXT*, pp. 283–294, December 2013
36. Wang, D., Joshi, G., Wornell, G.: Efficient task replication for fast response times in parallel computation. Technical Report arXiv:1404.1328, April 2014
37. Xia, C., Liu, Z., Towsley, D., Lelarge, M.: Scalability of fork/join queueing networks with blocking. In *SIGMETRICS*, pp. 133–144, June 2007
38. Xu, Y., Bailey, M., Noble, B., Jahanian, F.: Small is better: avoiding latency traps in virtualized data centers. In *Proceedings of the 4th annual Symposium on Cloud Computing*, p. 7. ACM, 2013