Scheduling for Efficiency and Fairness in Systems with Redundancy

Kristen Gardner[1], **Mor Harchol-Balter**[1],
Esa Hyytiä[2], **Rhonda Righter**[3]

April 14, 2017
CMU-CS-17-109

School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

[1]School of Computer Science, Carnegie Mellon University, Pittsburgh, PA, USA
[2]Department of Computer Science, University of Iceland, Reykjavik, Iceland
[3]Department of Industrial Engineering and Operations Research, UC Berkeley, Berkeley, CA, USA

This work was supported by the NSF under grants NSF-XPS-1629444, NSF-CMMI-1538204, and NSF-CMMI-1334194; by a Google Faculty Research Award; by a Facebook Faculty Gift; by a Google Anita Borg Memorial Scholarship; and by the Siebel Scholars Program.

Keywords: Redundancy, Replication, Parallel servers, Scheduling, Least Redundant First, LRF, Primaries First

Abstract

Server-side variability—the idea that the same job can take longer to run on one server than another due to server-dependent factors—is an increasingly important concern in many queueing systems. One strategy for overcoming server-side variability to achieve low response time is *redundancy*, under which jobs create copies of themselves and send these copies to multiple different servers, waiting for only one copy to complete service. Most of the existing theoretical work on redundancy has focused on developing bounds, approximations, and exact analysis to study the response time gains offered by redundancy. However, response time is not the only important metric in redundancy systems: the system should also be *fair* in the sense that no job class should have a worse mean response time in the system with redundancy than it did in the system before redundancy is allowed.

In this paper we use *scheduling* to address the simultaneous goals of (1) achieving low response time and (2) maintaining fairness across job classes. We develop new exact analysis for per-class response time under First-Come First-Served (FCFS) scheduling for a general type of system structure; our analysis shows that FCFS can be unfair in that it can hurt non-redundant jobs. We then introduce the Least Redundant First (LRF) scheduling policy, which we prove is optimal with respect to overall system response time, but which can be unfair in that it can hurt the jobs that become redundant. Finally, we introduce the Primaries First (PF) scheduling policy, which is provably fair and also achieves excellent overall mean response time.

1 Introduction

In many queueing systems, customers can experience long and highly variable response times because of server-side factors that make service times unpredictable. For example, in cloud computing many virtual machines (VMs) are co-located on the same physical machine; the VMs compete for resources, and the speed of any given VM at any time depends on the particular balance of resources being used by all of the co-located VMs. Another example is the deceased donor organ transplant waitlist, in which the waiting times for patients at the head of the queue in different regions can be very different because organs become available independently in different regions due to factors that are hard to predict. Even in smaller everyday systems such as daycare center waitlists or library book holds, it can be hard to tell which queue is the best to join, simply because waiting times depend largely on unobservable and unpredictable server-side factors (e.g., the reading speed of the person who currently has a library book checked out). In the face of this variability, traditional queueing paradigms often are ineffective at achieving low response times.

One increasingly popular strategy for overcoming server-side variability is *redundancy*, the idea of making multiple copies of a job and sending these copies to different servers. The job is considered complete as soon as the first copy completes service, at which time all other copies are cancelled. Redundancy helps for two reasons. First, by waiting in multiple queues, the job gets to experience the shortest queueing time without knowing in advance the service times that the jobs ahead of it will experience. Second, the job may end up in service at multiple servers simultaneously; when service times are unpredictable and largely server-dependent, running on multiple servers can help a job to experience a shorter service time. In many systems, such as computer systems and kidney transplant waitlists, redundancy has been demonstrated empirically to significantly reduce response times for the overall system.

However, in many applications there are some jobs that cannot be made redundant, for example because of data locality constraints or geographic constraints, and one of the major concerns about redundancy is that its potential benefits are not felt equally across different classes of jobs. In particular, jobs that can become redundant are likely to benefit from being redundant, whereas their redundancy might hurt some non-redundant jobs. This observation, which is particularly important in applications such as organ transplant waitlists where patients' ability to become redundant correlates with their socioeconomic status, leads us to a second goal: in addition to reducing overall mean response time, we want response time gains to be experienced *fairly* across classes. To understand what we mean by "fairness," consider two systems. In the first system, no jobs are redundant; each job is dispatched to a single server, where this server is determined by the job's class. In the second system, some jobs become redundant. The redundant jobs are dispatched to their original server as well as to one or more additional servers, where the additional servers again are determined by the job's new class. Our goal is to improve efficiency, i.e., overall response time, while ensuring that each job class experiences a mean response time in the second (redundant) system that is no worse than what that job class experiences in the first (non-redundant) system. This notion of fairness—that a "fair" scheduling policy ensures that no job performs worse than under a baseline policy—has also been used to develop fair variations on the Shortest Remaining Processing Time policy [3].

While there are many ways in which one could attempt to control the effects of job redundancy,

in this paper we focus on server-side scheduling policies. We assume that each job class replicates itself to a particular subset of the servers, where this subset is determined by, e.g., data locality constraints. Given this replication, we ask in what order each server should work on the jobs in its queue in order to achieve our dual goals of (1) achieving low overall mean response time (high system efficiency), and (2) maintaining per-class fairness. Throughout, we assume that job sizes are not known, hence we focus on non-size-based scheduling policies. We also assume that jobs are preemptible.

We focus on a particular system structure called a *nested* structure. In a nested redundancy system, if two classes of jobs share a server, then one of the class's servers must be a subset of the other class's servers. The nesting structure is common in operating systems such as Multics and UNIX for organizing access to resources via hierarchical protection domains, known as "nested protection rings." The most privileged classes of users have access to all resources. Less privileged classes of users have access to smaller and smaller subsets of the resources. Classes are nested so that, for any two user classes, A and B, that share access to some resource, either A's set of resources is a subset of B's set, or vice versa [10, 12]. In this paper we think of nested structures as existing not at the operating system level, but at the level of an entire data center, where data may be replicated across servers on the same rack, or across multiple racks.

We start with a system that has no redundancy in which each server schedules jobs in first-come first-served (FCFS) order, and ask what happens when some jobs become redundant and the servers continue to use first-come first-served scheduling (FCFS-R). We provide an exact analysis of the full distribution of response time for each class.

While FCFS-R can yield significant response time improvements relative to FCFS (with no redundancy), it turns out that FCFS-R is not the best way to leverage the available redundancy. We propose a new policy, Least Redundant First (LRF), which gives less redundant jobs preemptive priority over more redundant jobs. We prove that LRF is *optimal* in a very strong sense. It stochastically maximizes the cumulative job departure process, and hence it minimizes overall mean response time. While response time under LRF is not analytically tractable, we derive upper and lower bounds on per-class and overall system mean response time. Surprisingly, mean response time under FCFS-R is nearly as low as that under the optimal LRF policy.

The observation that unsophisticated policies like FCFS-R achieve near-optimal performance has important implications for where to focus our efforts in system design, because unfortunately, both FCFS-R and LRF fail to achieve our second goal: maintaining per-class fairness. Although FCFS-R always helps the jobs that become redundant, the non-redundant jobs can be hurt. On the other hand, LRF is so conservative about protecting non-redundant jobs that the redundant jobs can actually suffer and experience higher response times than they would if they were not redundant at all. To balance out the benefits of LRF, under which non-redundant jobs are isolated from competing redundant jobs, and FCFS-R, under which redundant jobs get to retain their place in line, we propose the Primaries First (PF) policy. Under PF, each job has a single primary copy which joins a queue and is served in FCFS order. If the job is redundant all new copies are designated secondaries and are given lowest preemptive priority. Like FCFS-R, PF provides overall mean response times that are nearly indistinguishable from the optimal LRF, thereby achieving our first goal. Furthermore, we prove that all classes of jobs are at least as well off under PF as under

non-redundant FCFS, thereby achieving our second goal.

2 Prior Work

As redundancy becomes increasingly common in computer systems, there is a growing body of theoretical work on analyzing systems with redundancy. Here we review the related work both on exact analysis and on scheduling in redundancy systems.

The limiting distribution on the state space was derived under FCFS-R for the same model we study in [5], but the authors only find per-class response time distributions for a few simple 2- and 3-server systems. In [2] this result is generalized to derive the limiting distribution on the state space in networks of redundant queues. The distribution of response time in systems with any number of identical servers in which each job sends copies to d servers chosen at random is derived in [4]. Nested structures do not fit this structure; in this paper we provide a new generalization of the results presented in [5] to derive the per-class response time distributions in nested redundancy systems with heterogeneous servers.

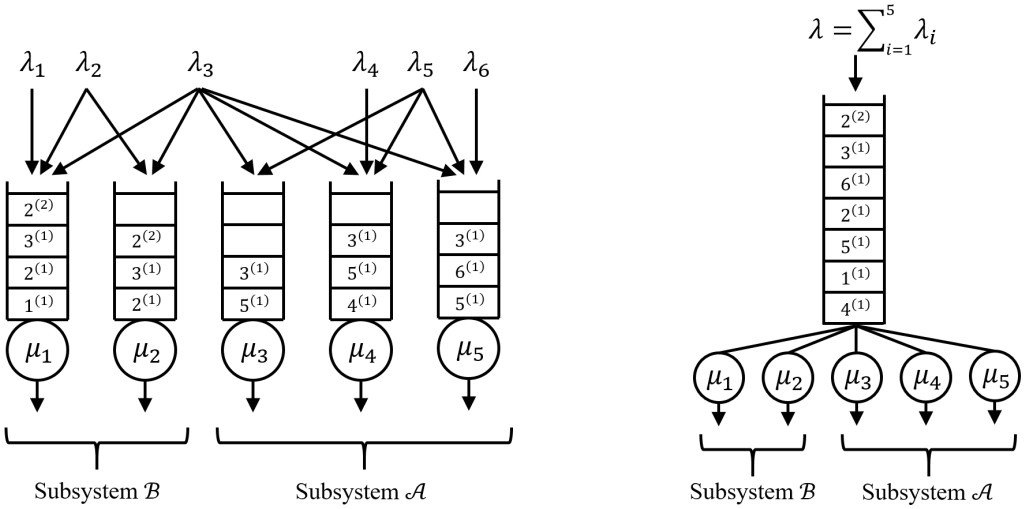
Visschers, Adan, and Weiss [1, 13] consider a similar model with different job and server classes that determine which jobs can be served on which servers, but without redundancy. They assume jobs are served in FCFS order, and a job that arrives to find multiple idle compatible servers is assigned to the compatible server that has been idle longest [1] or to a randomly chosen idle compatible server according to a specific routing probability [13]. Under the routing assumptions in [1, 13], the system exhibits a similar product-form limiting distribution to that in our redundancy system under FCFS-R scheduling, but with a different state space, and in both systems per-class response times can be expressed as a sum of M/M/1 queueing times (see Section 4).

On the scheduling side, when all jobs are capable of running on all servers and service times follow a new-better-than-used distribution, it is optimal to schedule the same job on all servers at the same time [8, 9]. Unlike in our model, in [8, 9] jobs do not have classes that restrict the set of servers to which each job can replicate, so the result does not extend to our system.

The setting in which each job can replicate to only a subset of the servers has been studied in [11]. Unlike in our model, in the system studied in [11] each job consists of k subtasks, all of which must complete service in order for the job to be complete (in our model, $k = 1$). The scheduling question in this setting is very different from the question we pose because [11] studies policies that choose both which job and which task to run, possibly allowing tasks to run redundant copies. Bounds and approximations for response time in this model, called the (n, k) system, have been derived [7, 14].

3 Model

We consider a general multi-server system consisting of k servers and ℓ classes of jobs, as shown in Figure 1. Jobs arrive to the system with average rate λ. Arrivals form an exogenous renewal process; in some cases we restrict ourselves to a Poisson arrival process. Each job belongs to class j independently with probability p_j; the arrival rate of class j is $\lambda_j = \lambda \cdot p_j$. Each job



(a) A nested redundancy structure (b) Central queue system representation

Figure 1: (a) A nested redundancy structure with $k = 5$ servers and $\ell = 6$ job classes. The most redundant class, in this case class 3, replicates itself to subsystem $\mathcal{A}$, which consists of servers 1 and 2 and job classes 1 and 2; and subsystem $\mathcal{B}$, which consists of servers 3, 4, and 5 and job classes 4, 5, and 6. The two subsystems are disjoint: they have no servers or job classes in common except for class 3, which replicates to all servers in both subsystems. In this example, the jobs arrived in the following order: $4^{(1)}, 1^{(1)}, 5^{(1)}, 2^{(1)}, 6^{(1)}, 3^{(1)}, 2^{(2)}$, where $i^{(j)}$ denotes the jth arrival of class-i, illustrated in the central queue in (b).

of class j replicates itself upon arrival by joining the queues at a fixed subset of the servers $S_j = \{s \mid \text{server } s \text{ can serve class } j\}$. A job is allowed to be in service at multiple servers simultaneously. Each job departs the system immediately as soon as its first replica completes service, at which time all remaining replicas are immediately cancelled.

The k servers are heterogeneous, where each server s provides service rate μ_s. The system as a whole has total service rate $\mu = \sum_{s=1}^{k} \mu_s$. Service times are assumed to be exponential and independent across jobs on the same server and across the same job on different servers. That is, if a job is in service at both servers s and r, its remaining time is distributed as $\min\{\text{Exp}(\mu_s), \text{Exp}(\mu_r)\} = \text{Exp}(\mu_s + \mu_r)$. We denote the set of job classes that a particular server s can serve by $J_s = \{j \mid s \in S_j\}$. Note that in our model, if we have multiple identical servers (identical in the sense that they can serve the same classes of jobs, not necessarily in service rates), we can think of these servers as a single fast server. This is because a single job will occupy all the identical servers, releasing them all at the same time for the next job to be assigned.

We restrict our attention to a specific type of redundancy structure called a *nested* system.

Definition 1 *In a nested system, for all job classes i and j, we have either (1) $S_i \subset S_j$, (2) $S_i \supset S_j$, or (3) $S_i \cap S_j = \emptyset$.*

Figure 1 shows an example of a nested system with 5 servers and 6 classes of jobs. As examples

of the above cases, we see that $S_4 \subset S_5$ (Case 1), $S_3 \supset S_6$ (Case 2), and $S_1 \cap S_5 = \emptyset$ (Case 3).

In Lemmas 1 and 2 we note two properties of nested systems that we use in the remainder of this paper.

Lemma 1 *In a nested redundancy system with k servers, either there exists a class-R such that $S_R = \{1, \ldots, k\}$ (i.e., class-R jobs replicate to all k servers), or the system can be separated into two or more independent nested subsystems.*

Proof. If there is some class of jobs that replicates to all k servers, we are done. So suppose this is not the case. Let class i denote the most redundant class, i.e., $|S_j| \leq |S_i|$ for all classes $j \neq i$, and let set $\mathcal{S} = \{s : \ s \notin S_i\}$ denote the set of servers to which class-i jobs do not replicate. For every job class j that shares a server with class i, it must be the case that $S_j \subset S_i$, and so there does not exist a server s such that $s \in S_j$ and $s \in \mathcal{S}$. Hence S_i and $\mathcal{S}$ do not have any servers or job classes in common, so they form two independent subsystems. □

From Lemma 1 we assume without loss of generality that the most redundant class, R, replicates to all k servers. In Lemma 2 we make the observation that the fully redundant class-R jobs can be viewed as replicating themselves to two independent subsystems, which we will call subsystems $\mathcal{A}$ and $\mathcal{B}$ (see Figure 1).

Lemma 2 *Let class-R be the most redundant job class in the nested system (i.e., class-R jobs replicate to all k servers). Then the remaining $\ell - 1$ job classes can be partitioned into two nested subsystems, denoted $\mathcal{A}$ and $\mathcal{B}$, such that for all classes $i \in \mathcal{A}$, $j \in \mathcal{B}$, $S_i \cap S_j = \emptyset$. That is, none of the classes in $\mathcal{A}$ have any servers in common with any of the classes in $\mathcal{B}$.*

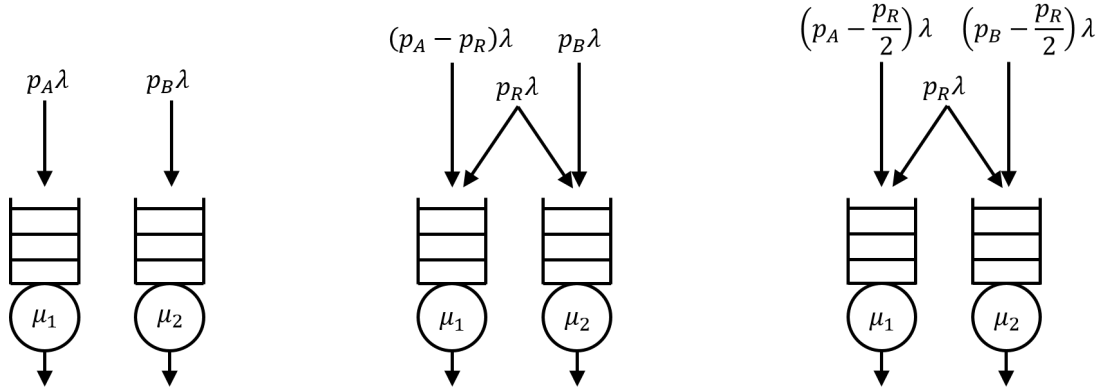
Proof. Assume the job classes are sorted with nonincreasing $|S_j|$, so that $|S_j| \geq |S_{j+1}|$ for $j = 2, \ldots, \ell - 1$. We will construct $\mathcal{A}$ and $\mathcal{B}$ as follows. Begin by initializing $\mathcal{A} = \{1\}$ and $\mathcal{B} = \emptyset$. For each class $j = 2, \ldots, \ell - 1$,

1. If there exists a class $i \in \mathcal{A}$ such that $S_j \subset S_i$, add class j to $\mathcal{A}$.
2. Otherwise, by the nesting property and the fact that all classes i already in $\mathcal{A}$ have $|S_i| \geq |S_j|$, we know that for all classes $i \in \mathcal{A}$ we have $S_i \cap S_j = \emptyset$. Hence we add class j to $\mathcal{B}$.

After each addition of a class to either set $\mathcal{A}$ or $\mathcal{B}$, we continue to have the property that for all classes $i \in \mathcal{A}$, $j \in \mathcal{B}$, $S_i \cap S_j = \emptyset$. □

Example 1 *In Figure 1, class 3 jobs are fully redundant. We sort the remaining job classes in nonincreasing order of redundancy degree: 5, 2, 1, 4, 6. Starting at the beginning of this list, we set $\mathcal{A} = \{5\}$ and $\mathcal{B} = \emptyset$. Since $S_2 \cap S_5 = \emptyset$, we add class 2 to $\mathcal{B}$; we then add class 1 to $\mathcal{B}$ for the same reason. Since $S_4 \subset S_5$, we add class 4 to $\mathcal{A}$. Finally, we add class 6 to $\mathcal{A}$. Observe that $\mathcal{A} = \{5, 4, 6\}$ and $\mathcal{B} = \{2, 1\}$ are disjoint.*

Throughout much of the remainder of this paper, we use a specific nested system called the $\mathbb{W}$ model as a case study to illuminate the effects of different scheduling policies on different classes of jobs. We begin with a system in which there are two classes of jobs, each with its own server



(a) No redundancy

(b) Some class-A jobs become class-R jobs

(c) Some class-A jobs and some class-B jobs become class-R jobs

Figure 2: 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 join both queues. Throughout this paper, we compare (a) The system in which there are no redundant jobs ($p_R = 0$) to (b) The system in which some class-A jobs become class-R jobs, and (c) The system in which some class-A jobs and some class-B jobs become class-R jobs.

(see Figure 2(a)). Class-A jobs are non-redundant and join the queue at server 1 only. Class-B jobs are non-redundant and join the queue at server 2 only. We let p_A and p_B denote the fraction of jobs that are class-A and class-B respectively, where $p_A + p_B = 1$. To understand the impact that different redundancy-based scheduling policies have on response time, we compare this system to the $\mathbb{W}$ model, in which some jobs become redundant class-R jobs, which join the queues at both servers 1 and 2. We consider two cases: the case in which p_R fraction of the jobs become redundant, where originally all of these jobs were class-A (Figure 2(b)), and the case in which p_R fraction of the jobs become redundant, where originally half of these jobs were class-A and half were class-B (Figure 2(c)).

For notational simplicity, we let λ_A, λ_B and λ_R denote the arrival rates of class-A, B, and R jobs respectively, where λ_i is defined differently in terms of λ, p_A, p_B, and p_R for each of the systems shown in Figure 2. For example, in Figure 2(a) $\lambda_A = p_A\lambda$, whereas in Figure 2(b) $\lambda_A = (p_A - p_R)\lambda$ and in Figure 2(c) $\lambda_A = (p_A - \frac{p_R}{2})\lambda$.

4 First-Come First-Served with Redundancy (FCFS-R)

We first consider redundancy systems using the standard first-come first-served scheduling policy, which we call FCFS-R to distinguish this policy from FCFS scheduling in a system in which no jobs are redundant. Under FCFS-R, each server works on the jobs in its queue in FCFS order; a job may be in service at multiple servers at the same time. Throughout this section we assume that

$(2,3,6,5,4,2,1)$	$(2,3,6,5,2,4,1)$	$(2,3,6,5,2,1,4)$	$(2,3,6,2,5,4,1)$	$(2,3,6,2,5,1,4)$
$(2,3,6,2,1,5,4)$	$(2,3,2,6,5,4,1)$	$(2,3,2,6,5,1,4)$	$(2,3,2,6,1,5,4)$	$(2,3,2,1,6,5,4)$

Table 1: If we did not know the order in which the jobs arrived, we would not be able to distinguish between these ten states just from looking at the snapshot of the system shown in Figure 1a.

arrivals form a Poisson process.

4.1 Response Time Analysis

Our goal in this section is to derive the distribution of response time—defined as the time from when a job arrives to the system until its first copy completes service—in nested redundancy systems under FCFS-R scheduling. We do so using a Markov chain approach. Following [5], our state space tracks all of the jobs in the system in the order in which they arrived; this can be viewed as a single central queue, as shown in Figure 1(b). We denote the system state by $(c_m, c_{m-1}, \ldots, c_1)$, where there are m jobs in the system and c_i is the class of the ith job in the (central) queue; the head of the queue is at the right. In the example shown in Figure 1 the system state is $(2,3,6,2,5,1,4)$. Note that without knowing the arrival order of the jobs shown in Figure 1(a), we do not have enough information to distinguish between the states shown in Table 1. While we know that the oldest class-2 job arrived after the class-1 job, and the class-6 job arrived after the class-5 job, which in turn arrived after the class-4 job, we do not know how the class-1 and class-2 jobs are interleaved with the class-4, class-5, and class-6 jobs because the two sets of jobs have no servers in common. However, all of the states listed in Table 1 are equivalent in terms of the future evolution of the system.

Theorem 1 in [5] tells us that the limiting probability of being in state $(c_m, c_{m-1}, \ldots, c_1)$ is

$$\pi_{(c_m,\ldots,c_1)} = \mathcal{C} \prod_{i=1}^{m} \frac{\lambda_{c_i}}{\mu_{\mathcal{I}_{1,\ldots,i}}}, \tag{1}$$

where $\mathcal{I}_{1,\ldots,i} = \{s | s \in \bigcup_{j \le i} S_{c_j}\}$ is the set of all servers that can serve at least one job in positions 1 to i in the queue, and $\mathcal{C}$ is a normalizing constant. For example, the limiting probability of the state shown in Figure 1(b) is

$$\pi_{(2,3,6,2,5,1,4)} = \mathcal{C} \cdot \left(\frac{\lambda_2}{\sum_{i=1}^5 \mu_i}\right) \cdot \left(\frac{\lambda_3}{\sum_{i=1}^5 \mu_i}\right) \cdot \left(\frac{\lambda_6}{\sum_{i=1}^5 \mu_i}\right) \cdot \left(\frac{\lambda_2}{\sum_{i=1}^5 \mu_i}\right) \cdot \left(\frac{\lambda_5}{\mu_1+\mu_3+\mu_4+\mu_5}\right) \cdot \left(\frac{\lambda_1}{\mu_1+\mu_4}\right) \cdot \left(\frac{\lambda_4}{\mu_4}\right).$$

This form is quite unusual: the limiting probabilities cannot be written in general as a product of per-class terms or of per-server terms.

Unfortunately, knowing an explicit expression for the limiting distribution of our state space does not immediately yield results for the distribution of response time for each class; [5] only finds

per-class response time distributions for a few small systems. The problem is that the state space is very detailed, and it is not obvious how to aggregate the states to find the per-class distribution of the number in system, from which we can find the distribution of response time via Distributional Little's Law. This aggregation is performed in [6] for a system called the Redundancy-d system, in which each job replicates itself to d randomly chosen servers, but nested structures do not fit this model.

In this section, we derive the distribution of response time for each job class in any nested redundancy structure. We perform a state aggregation that allows us to express the limiting probabilities in terms of the limiting probabilities in the left and right subsystems. Our new aggregated states are still sufficiently detailed for us to determine the steady-state response time distribution for each job class.

Let class-R denote the most redundant job class in the system (i.e., class-R jobs replicate to all k servers). We define the aggregated state $\left(R, n_R, \substack{(\mathcal{A}) \\ (\mathcal{B})}\right)$ to be the state in which subsystems $\mathcal{A}$ and $\mathcal{B}$ have states $(\mathcal{A})$ and $(\mathcal{B})$ respectively (including only those jobs that are ahead of the first class-R job in the queue), the R denotes the first class-R job in the system, and n_R is the number of jobs in the queue behind the first class-R job (note that these jobs may be of any class). If there are no class-R jobs in the system, we set $n_R =$ '-', so the state is $\left(R, -, \substack{(\mathcal{A}) \\ (\mathcal{B})}\right)$. If the system is empty the state is defined to be $\emptyset$. Our new state aggregates over two things. First, we aggregate over all possible interleavings of the jobs in subsystems $\mathcal{A}$ and $\mathcal{B}$. Second, we track only the number of jobs in the queue behind the first class-R job rather than the specific classes of each of these jobs. We can recursively apply our aggregation so that in states $(\mathcal{A})$ and $(\mathcal{B})$ we only track the number of jobs behind the first job that is fully redundant in subsystems $\mathcal{A}$ and $\mathcal{B}$ respectively.

Example 2 *We rewrite the state in Figure 1 as* $(3, n_3 = 1, \substack{(6,5,4) \\ (2,1)})$*, where here* $(\mathcal{A}) = (6, 5, 4)$ *and* $(\mathcal{B}) = (2, 1)$*. Note that substates* $(\mathcal{A})$ *and* $(\mathcal{B})$ *only include the jobs that appear in the queue ahead of the first class-*3 *job, which is fully redundant. We then recursively aggregate substates* $(\mathcal{A})$ *and* $(\mathcal{B})$ *to get the state*

$$\left(3, n_3 = 1, \substack{\left(5, n_5=1, \substack{(4, n_4=0) \\ (6, n_6=-)}\right) \\ \left(2, n_2=0, \substack{(1, n_1=0) \\ (-)}\right)}\right).$$

In this new aggregated state, we know that there is a class-3 job in the system with one job in the queue behind it, but we do not know the class of this job (it could be any class). Similarly, we know that there is a class-5 job in the system with one job in the queue behind it but in front of the class-3 job. This job in the queue could be class-4, 5, or 6.

Effectively, aggregating states in this manner defers the point at which we determine a job's class. In our system description in Section 3, we assign each job's class upon arrival. However, because job classes are assigned independently according to fixed probabilities, there is no need to realize a job's class until a server becomes available and we need to know whether the server is capable of working on the job. We can interpret our aggregated states as being the state of an alternative (but equivalent) system in which we only discover information about a job's class at the moment when we must determine whether the job will enter service on an idle server. This state

description also aggregates states that are indistinguishable in terms of arrival order. For example, all of the states listed in Table 1 are included in the aggregated state in the above example.

Let $\mathcal{I}_j$ denote the subsystem in which class-j is fully redundant. That is, the classes in subsystem $\mathcal{I}_j$ are all classes i such that $S_i \subseteq S_j$, and the servers in subsystem $\mathcal{I}_j$ are the servers in S_j. In our example, $\mathcal{A} = \mathcal{I}_5$ and $\mathcal{B} = \mathcal{I}_2$. We use $\mu_{\mathcal{I}_j}$ to denote the total service rate of all servers in $\mathcal{I}_j$, and $\lambda_{\mathcal{I}_j}$ to denote the total arrival rate of all job classes in $\mathcal{I}_j$, and define $\mu_{\mathcal{A}} = \sum_{s \in S_{\mathcal{A}}} \mu_s$, $\mu_{\mathcal{B}} = \sum_{s \in S_{\mathcal{B}}} \mu_s$, $\lambda_{\mathcal{A}}$, and $\lambda_{\mathcal{B}}$ similarly.

Lemma 3 *The limiting probability of being in state* $\left(R, n_R, \binom{(\mathcal{A})}{(\mathcal{B})}\right)$*, for* $n_R \neq -$*, is*

$$\pi_{\left(R, n_R, \binom{(\mathcal{A})}{(\mathcal{B})}\right)} = \mathcal{C} \cdot P\left(R, n_R, \binom{(\mathcal{A})}{(\mathcal{B})}\right), \tag{2}$$

where $\mathcal{C}$ *is a normalizing constant and* P *satisfies*

$$P\left(R, n_R, \binom{(\mathcal{A})}{(\mathcal{B})}\right) = \left(\frac{\lambda_{\mathcal{A}} + \lambda_{\mathcal{B}} + \lambda_R}{\mu_{\mathcal{A}} + \mu_{\mathcal{B}}}\right)^{n_R} \left(\frac{\lambda_R}{\mu_{\mathcal{A}} + \mu_{\mathcal{B}}}\right) \cdot P(\mathcal{A}) \cdot P(\mathcal{B}) \tag{3}$$

and $P(\emptyset) = 1$*. The limiting probability of being in state* $\left(R, -, \binom{(\mathcal{A})}{(\mathcal{B})}\right)$ *is*

$$\pi_{\left(R, -, \binom{(\mathcal{A})}{(\mathcal{B})}\right)} = \mathcal{C} \cdot P\left(R, -, \binom{(\mathcal{A})}{(\mathcal{B})}\right),$$

where $P\left(R, -, \binom{(\mathcal{A})}{(\mathcal{B})}\right)$ *satisfies*

$$P\left(R, -, \binom{(\mathcal{A})}{(\mathcal{B})}\right) = P(\mathcal{A}) \cdot P(\mathcal{B}).$$

Proof. We defer the detailed proof to Appendix A and here give a proof sketch for an example of the $\mathbb{W}$ model. Consider state (A, R, B, A). From (1) we know that the limiting probability of being in this state is

$$\pi_{A,B,R,A} = \mathcal{C}\left(\frac{\lambda_A}{\mu} \cdot \frac{\lambda_R}{\mu} \cdot \frac{\lambda_B}{\mu} \cdot \frac{\lambda_A}{\mu_1}\right).$$

We first aggregate over the different arrival orders that are indistinguishable from looking at a snapshot of the system. In particular, we cannot distinguish between states (A, R, B, A) and (A, R, A, B). We have:

$$\begin{aligned}
\pi_{\left(A, R, \binom{A}{B}\right)} &= \pi_{(A,R,B,A)} + \pi_{(A,R,A,B)} \\
&= \mathcal{C}\left(\frac{\lambda_A}{\mu} \cdot \frac{\lambda_R}{\mu} \cdot \frac{\lambda_B}{\mu} \cdot \frac{\lambda_A}{\mu_1} + \frac{\lambda_A}{\mu} \cdot \frac{\lambda_R}{\mu} \cdot \frac{\lambda_A}{\mu} \cdot \frac{\lambda_B}{\mu_2}\right) \\
&= \mathcal{C}\left(\frac{\lambda_A}{\mu} \cdot \frac{\lambda_R}{\mu} \cdot \frac{\lambda_A}{\mu_1} \cdot \frac{\lambda_B}{\mu_2}\right).
\end{aligned}$$

We then aggregate over all job classes that could appear in the queue after the first class-R job. That is, we aggregate states $(A, R, \frac{A}{B})$, $(B, R, \frac{A}{B})$, and $(R, R, \frac{A}{B})$:

$$\begin{aligned}\pi_{\left(R,1,\frac{A}{B}\right)} &= \pi_{\left(A,R,\frac{A}{B}\right)} + \pi_{\left(B,R,\frac{A}{B}\right)} + \pi_{\left(R,R,\frac{A}{B}\right)} \\ &= \mathcal{C}\left(\frac{\lambda_A}{\mu}\cdot\frac{\lambda_R}{\mu}\cdot\frac{\lambda_A}{\mu_1}\cdot\frac{\lambda_B}{\mu_2} + \frac{\lambda_B}{\mu}\cdot\frac{\lambda_R}{\mu}\cdot\frac{\lambda_A}{\mu_1}\cdot\frac{\lambda_B}{\mu_2} + \frac{\lambda_R}{\mu}\cdot\frac{\lambda_R}{\mu}\cdot\frac{\lambda_A}{\mu_1}\cdot\frac{\lambda_B}{\mu_2}\right) \\ &= \mathcal{C}\left(\frac{\lambda}{\mu}\cdot\frac{\lambda_R}{\mu}\cdot\frac{\lambda_A}{\mu_1}\cdot\frac{\lambda_B}{\mu_2}\right),\end{aligned}$$

which is the form given in 3. The full proof, given in Appendix A, is a generalization of this example. □

Note that Lemma 3 can be applied recursively to the subsystems $\mathcal{A}$ and $\mathcal{B}$.

Let $\rho_i = \frac{\lambda_i}{\mu_{\mathcal{I}_i} - \lambda_{\mathcal{I}_i} + \lambda_i}$. We can interpret $\mu_{\mathcal{I}_i} - \lambda_{\mathcal{I}_i} + \lambda_i$ as being the service capacity in subsystem $\mathcal{I}_i$ that is available for class-i after capacity is allocated to the remaining classes in subsystem $\mathcal{I}_i$. Then ρ_i is the fraction of time that this available capacity is used to process class-i jobs. We will show later that ρ_i is also the probability that there is at least one class-i job in the system.

Lemma 4 *In a nested redundancy system with ℓ job classes, the normalizing constant $\mathcal{C}$ is*

$$\mathcal{C} = \prod_{i=1}^{\ell}(1-\rho_i) \tag{4}$$

Proof. Deferred to Appendix B. □

Example 3 *In the system shown in Figure 1, the normalizing constant is*

$$\begin{aligned}\mathcal{C} &= \prod_{i=1}^{6}(1-\rho_i) \\ &= \left(1-\frac{\lambda_1}{\mu_1}\right)\cdot\left(1-\frac{\lambda_2}{\mu_1+\mu_2-\lambda_1}\right)\cdot\left(1-\frac{\lambda_3}{\sum_{s=1}^{5}\mu_s - \sum_{i=1}^{5}\lambda_i + \lambda_3}\right) \\ &\quad\cdot\left(1-\frac{\lambda_4}{\mu_4}\right)\cdot\left(1-\frac{\lambda_5}{\mu_3+\mu_4+\mu_5-\lambda_4-\lambda_6}\right)\cdot\left(1-\frac{\lambda_6}{\mu_6}\right)\end{aligned}$$

Theorem 1 *In a nested redundancy system with ℓ job classes, the response time for class i, T_i, can be expressed as*

$$T_i = T(\lambda_{\mathcal{I}_i}, \mu_{\mathcal{I}_i}) + \sum_{j:S_i\subset S_j} T_Q(\lambda_{\mathcal{I}_j}, \mu_{\mathcal{I}_j}),$$

where $T(\lambda,\mu)$ and $T_Q(\lambda,\mu)$ represent the response time and queueing time respectively in an M/M/1 with arrival rate λ and service rate μ. That is, the Laplace transform of response time for class i is

$$\widetilde{T}^{(i)}(s) = \left(\frac{\mu_{\mathcal{I}_i}-\lambda_{\mathcal{I}_i}}{\mu_{\mathcal{I}_i}-\lambda_{\mathcal{I}_i}+s}\right)\cdot\prod_{j:S_i\subset S_j}\left(\frac{\mu_{\mathcal{I}_j}-\lambda_{\mathcal{I}_j}}{\mu_{\mathcal{I}_j}-\lambda_{\mathcal{I}_j}+s}\right)\left(\frac{\mu_{\mathcal{I}_j}-\lambda_{\mathcal{I}_j}+\lambda_j+s}{\mu_{\mathcal{I}_j}-\lambda_{\mathcal{I}_j}+\lambda_j}\right) \tag{5}$$

To interpret the form of the Laplace transform of response time given in (5), observe that the Laplace transform of time in queue in an M/M/1 with arrival rate λ and service rate μ is

$$\widetilde{T}_Q^{M/M/1} = \left(\frac{\mu - \lambda}{\mu - \lambda + s}\right) \cdot \left(\frac{\mu + s}{\mu}\right).$$

Hence we can interpret the class-i response time as the queueing time in $x - i$ successive M/M/1s, followed by the response time in a final M/M/1. Consider the example shown in Figure 1(a) for class $i = 4$. We imagine that when a class-4 job arrives to the system it behaves like a fully redundant class-3 job and joins the queue at all five servers. Now suppose that our class-4 job reaches the head of the queue at server 2, which cannot serve class-4 jobs. At this point the class-4 job has experienced the queueing time of a class-3 job, but it cannot yet enter service. We now imagine the class-4 job remaining in only the queues in subsystem $\mathcal{A}$, and repeat the above reasoning to see that the class-4 job will now experience the queueing time of a class-5 job (which is fully redundant in subsystem $\mathcal{A}$). Hence we can view class-i jobs as moving through an M/M/1 queue for each successive nested subsystem, until they are finally served in the "innermost" subsystem $\mathcal{I}_i$!

We now turn to the formal proof of Theorem 1.

Proof. Consider $\mathcal{I}_i$, the subsystem consisting only of those servers in S_i and the job classes r such that $S_r \subseteq S_i$. From Lemma 3 we know that the state of subsystem $\mathcal{I}_i$ can be written in the form $\left(i, n_i, \binom{(\mathcal{A}_i)}{(\mathcal{B}_i)}\right)$, where $(\mathcal{A}_i)$ and $(\mathcal{B}_i)$ represent the substates of the smaller subsystems into which subsystem $\mathcal{I}$ decomposes, as in Lemma 2. Note that we exclude jobs that appear in the queue behind the first class-i' job, where i' is the class such that $S_{i'}$ is the smallest set that strictly contains S_i.

Now consider some class j such that $S_j \supseteq S_{i'}$. We can write the state of subsystem $\mathcal{I}_j$ in the form $\left(j, n_j, \binom{(\mathcal{A}_j)}{(\mathcal{B}_j)}\right)$, where $(\mathcal{A}_j)$ and $(\mathcal{B}_j)$ represent the states of the smaller subsystems into which subsystem $\mathcal{I}_j$ decomposes. It must be the case that either $\mathcal{I}_i \subset \mathcal{A}_j$ or $\mathcal{I}_i \subset \mathcal{B}_j$; without loss of generality assume that $\mathcal{I}_i \subset \mathcal{A}_j$.

When deriving the distribution of the number of class-i jobs in the system, we can immediately aggregate over all possible $(\mathcal{B})$ states because none of the $\mathcal{B}$ subsystems contain any class-i jobs. Hence the overall system states we care about are of the form $\left(R, n_R, \binom{(\mathcal{A}_R)}{(*)}\right)$. Expanding all $(\mathcal{A})$ substates until we reach subsystem $\mathcal{I}_i$ and dropping the $(*)$ substates from our notation, we obtain states of the form $\left(C_x, n_x, C_{x-1}, n_{x-1}, \ldots, i', n_{i'}, i, n_i, \binom{(\mathcal{A}_i)}{(\mathcal{B}_i)}\right)$, where job classes $C_{i+1=i'}, \ldots, C_x = R$ are all the job classes j with $S_j \supset S_i$. Note that due to the nested structure, there is only one order in which these classes can appear in this description of the system state. In our state, if there is not a class-j job in the system we set $n_j =$ ' $-$ '; otherwise n_j is the number of jobs behind the first class-j job in the queue and in front of the first class-$(j+1)$ job. In the example shown in Figure 1, we can write the system state for class $i = 4$ as $\left(3, n_3 = 1, 5, n_5 = 2, 4, n_4 = 1, \binom{(-)}{(-)}\right)$.

From Lemma 3, the limiting probability of being in state $\left(C_x, n_x, C_{x-1}, n_{x-1}, \ldots, i, n_i, \substack{(*)\\(*)}\right)$ is

$$\pi_{\left(C_x,n_x,C_{x-1},n_{x-1},\ldots,i,n_i,\substack{(*)\\(*)}\right)} = \prod_{\substack{j=i\\ n_j\neq -}}^{x} \left(\frac{\lambda_j}{\mu_{\mathcal{I}_j}}\right) \cdot \left(\frac{\lambda_{\mathcal{I}_j}}{\mu_{\mathcal{I}_j}}\right)^{n_j} \cdot (1-\rho_j)\,.$$

Now we are ready to find the z-transform of the number of class-i jobs in the system. We will do this by conditioning on there being a class-a job in the system, where $S_a \supset S_i$. Aggregating over all possible values of n_j (including $n_j = -$) for all $j \neq a$, we find

$$\Pr\{\text{at least one class-}a\text{ job in system}\} = \frac{\lambda_a}{\mu_{\mathcal{I}_a} - \lambda_{\mathcal{I}_a} + \lambda_a} = \rho_a.$$

Conditioned on there being a class-a job in the system, we have:

$$\pi_{\left(C_x,n_x,C_{x-1},n_{x-1},\ldots,i,n_i,\substack{(*)\\(*)}\right)|n_a\neq -} = \left(\frac{\lambda_a}{\mu_{\mathcal{I}_a}}\right) \cdot \left(\frac{\lambda_{\mathcal{I}_a}}{\mu_{\mathcal{I}_a}}\right)^{n_a} \cdot \left(1-\frac{\lambda_{\mathcal{I}_a}}{\mu_{\mathcal{I}_a}}\right) \cdot \prod_{\substack{j=i\\ j\neq a\\ n_j\neq -}}^{x} \left(\frac{\lambda_j}{\mu_{\mathcal{I}_j}}\right) \cdot \left(\frac{\lambda_{\mathcal{I}_j}}{\mu_{\mathcal{I}_j}}\right)^{n_j} \cdot (1-\rho_j)\,.$$

Observe that the number of jobs in the queue behind the class-a job (but in front of the first class-$(a+1)$ job), $N^{(a)}$, is geometrically distributed with parameter $1-\frac{\lambda_{\mathcal{I}_a}}{\mu_{\mathcal{I}_a}}$. Each of these jobs is a class-i job with probability $\frac{\lambda_i}{\lambda_{\mathcal{I}_a}}$. Let $N_{i|a}$ be the number of class-i jobs behind the first class-a job, given that there is a class-a job. Conditioning on the value of $N^{(a)}$, we find that the z-transform of $N_{i|a}$ is

$$\begin{aligned}
\widehat{N}_{i|a}(z) &= \sum_{m=0}^{\infty} \left(\frac{\lambda_{\mathcal{I}_a}}{\mu_{\mathcal{I}_a}}\right)^m \left(1-\frac{\lambda_{\mathcal{I}_a}}{\mu_{\mathcal{I}_a}}\right) \sum_{n=0}^{m} \binom{m}{n} z^n \left(\frac{\lambda_i}{\lambda_{\mathcal{I}_a}}\right)^n \left(1-\frac{\lambda_i}{\lambda_{\mathcal{I}_a}}\right) \\
&= \sum_{m=0}^{\infty} \left(\frac{\lambda_{\mathcal{I}_a}}{\mu_{\mathcal{I}_a}}\right)^m \left(1-\frac{\lambda_{\mathcal{I}_a}}{\mu_{\mathcal{I}_a}}\right) \left(1-\frac{\lambda_i}{\lambda_{\mathcal{I}_a}}(1-z)\right)^m \\
&= \frac{1-\frac{\lambda_{\mathcal{I}_a}}{\mu_{\mathcal{I}_a}}}{1-\frac{\lambda_{\mathcal{I}_a}}{\mu_{\mathcal{I}_a}}\left(1-\frac{\lambda_i}{\lambda_{\mathcal{I}_a}}(1-z)\right)} \\
&= \frac{\mu_{\mathcal{I}_a} - \lambda_{\mathcal{I}_a}}{\mu_{\mathcal{I}_a} - \lambda_{\mathcal{I}_a} + \lambda_i - \lambda_i z} \\
&= \frac{\frac{\mu_{\mathcal{I}_a}-\lambda_{\mathcal{I}_a}}{\mu_{\mathcal{I}_a}-\lambda_{\mathcal{I}_a}+\lambda_i}}{1-\frac{\lambda_i}{\mu_{\mathcal{I}_a}-\lambda_{\mathcal{I}_a}+\lambda_i}z}.
\end{aligned}$$

Note that this is the transform of a geometric random variable with parameter

$$p_{i|a} = 1 - \frac{\lambda_i}{\mu_{\mathcal{I}_a} - \lambda_{\mathcal{I}_a} + \lambda_i}.$$

We now condition on whether there is a class-a job in the system to find $\widehat{N}_{i,a}(z)$, the z-transform of the number of class-i jobs in the queue immediately behind the class-a job (if there is one):

$$\begin{aligned}\widehat{N}_{i,a}(z) &= \Pr\{\text{at least one class-}a\text{ job in system}\}\widehat{N}_{i|a}(z) + \Pr\{\text{no class-}a\text{ job in system}\}z^0 \\ &= \rho_a \cdot \left(\frac{\mu_{\mathcal{I}_a} - \lambda_{\mathcal{I}_a}}{\mu_{\mathcal{I}_a} - \lambda_{\mathcal{I}_a} + \lambda_i - \lambda_i z}\right) + (1 - \rho_a) \\ &= \left(\frac{\mu_{\mathcal{I}_a} - \lambda_{\mathcal{I}_a}}{\mu_{\mathcal{I}_a} - \lambda_{\mathcal{I}_a} + \lambda_a}\right) \cdot \left(1 + \frac{\lambda_a}{\mu_{\mathcal{I}_a} - \lambda_{\mathcal{I}_a} + \lambda_i - \lambda_i z}\right) \\ &= \left(\frac{\mu_{\mathcal{I}_a} - \lambda_{\mathcal{I}_a}}{\mu_{\mathcal{I}_a} - \lambda_{\mathcal{I}_a} + \lambda_a}\right) \cdot \left(\frac{\mu_{\mathcal{I}_a} - \lambda_{\mathcal{I}_a} + \lambda_i + \lambda_a - \lambda_i z}{\mu_{\mathcal{I}_a} - \lambda_{\mathcal{I}_a} + \lambda_i - \lambda_i z}\right).\end{aligned}$$

Because we can view the queue as $x - i$ independent pieces, each with a geometrically distributed number of class-i jobs, we can write the z-transform of the number of class-i jobs in the system as:

$$\begin{aligned}\widehat{N}_i(z) = &\left(\frac{\mu_{\mathcal{I}_i} - \lambda_{\mathcal{I}_i}}{\mu_{\mathcal{I}_i} - \lambda_{\mathcal{I}_i} + \lambda_i}\right) \cdot \left(\frac{\mu_{\mathcal{I}_i} - \lambda_{\mathcal{I}_i} + \lambda_i}{\mu_{\mathcal{I}_i} - \lambda_{\mathcal{I}_i} + \lambda_i - \lambda_i z}\right) \\ &\cdot \prod_{a=i+1}^{x} \left(\frac{\mu_{\mathcal{I}_a} - \lambda_{\mathcal{I}_a}}{\mu_{\mathcal{I}_a} - \lambda_{\mathcal{I}_a} + \lambda_a}\right) \cdot \left(\frac{\mu_{\mathcal{I}_a} - \lambda_{\mathcal{I}_a} + \lambda_i + \lambda_a - \lambda_i z}{\mu_{\mathcal{I}_a} - \lambda_{\mathcal{I}_a} + \lambda_i - \lambda_i z}\right),\end{aligned}$$

where the first term is different because if $n_i \neq -$ we have one additional class-i job in the system.

Finally, to find the Laplace transform of response time for class-i jobs, note that class-i jobs depart the system in the order in which they arrived, so we can apply Distributional Little's Law. Let $z = \frac{\lambda_i + s}{\lambda_i}$. Then we have

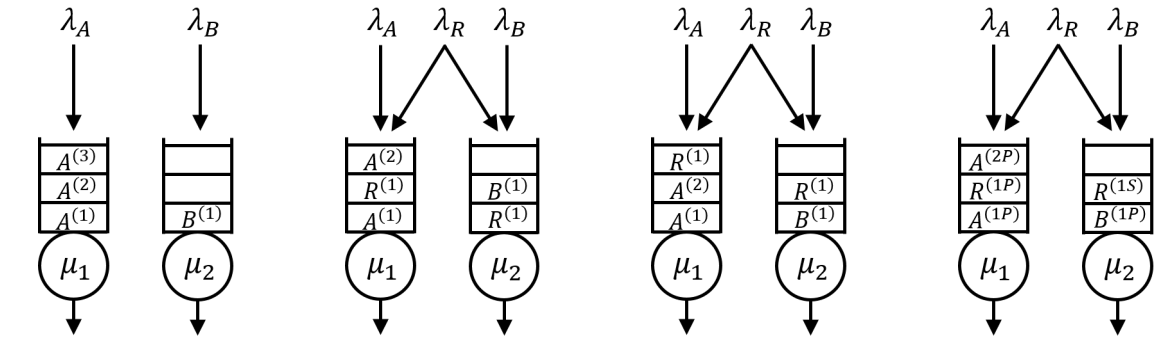
$$\widetilde{T}^{(i)}(s) = \left(\frac{\mu_{\mathcal{I}_i} - \lambda_{\mathcal{I}_i}}{\mu_{\mathcal{I}_i} - \lambda_{\mathcal{I}_i} + s}\right) \cdot \prod_{j:S_i \subset S_j} \left(\frac{\mu_{\mathcal{I}_j} - \lambda_{\mathcal{I}_j}}{\mu_{\mathcal{I}_j} - \lambda_{\mathcal{I}_j} + s}\right)\left(\frac{\mu_{\mathcal{I}_j} - \lambda_{\mathcal{I}_j} + \lambda_j + s}{\mu_{\mathcal{I}_j} - \lambda_{\mathcal{I}_j} + \lambda_j}\right)$$

as desired. □

4.2 Performance

Recall that our two goals are (1) to achieve low overall system mean response time, and (2) to preserve fairness across job classes, meaning that no class of jobs should have a higher mean response time in the redundancy system than in the original system with no redundancy. In this section we evaluate the extent to which FCFS-R meets these goals. We look specifically at the $\mathbb{W}$ model; Figure 3(a) shows the non-redundant system, and Figure 3(b) shows the redundancy system with FCFS scheduling. For our performance analysis, we assume the service rate is the same at both servers ($\mu_1 = \mu_2 = 1$). Figures 3(c) and (d) show the scheduling policies that we study in Section 5 and 6.

As described in Section 3, we consider two different settings. In the first setting, we start with an *asymmetric* system in which $p_A = 0.7$ and $p_B = 0.3$; we then look at the effect of having some jobs switch from being class-A to class-R by decreasing p_A while increasing p_R (Figure 2(b)). In the second setting, we start with a *symmetric* system (i.e., $p_A = p_B = 0.5$) and look at the effect of having some become redundant while the system remains symmetric. That is, we hold $\lambda_A = \lambda_B$ while increasing p_R (Figure 2(c)).



(a) non-redundant FCFS (b) FCFS-R (c) LRF (d) PF

Figure 3: The $\mathbb{W}$ model with four different scheduling policies: (a) FCFS in a system with no redundancy, (b) FCFS in a system where class-R jobs are redundant (FCFS-R, Section 4), (c) Least Redundant First (Section 5), and (d) Primaries First, where P and S denote a job's primary and secondary copies respectively (Section 6).

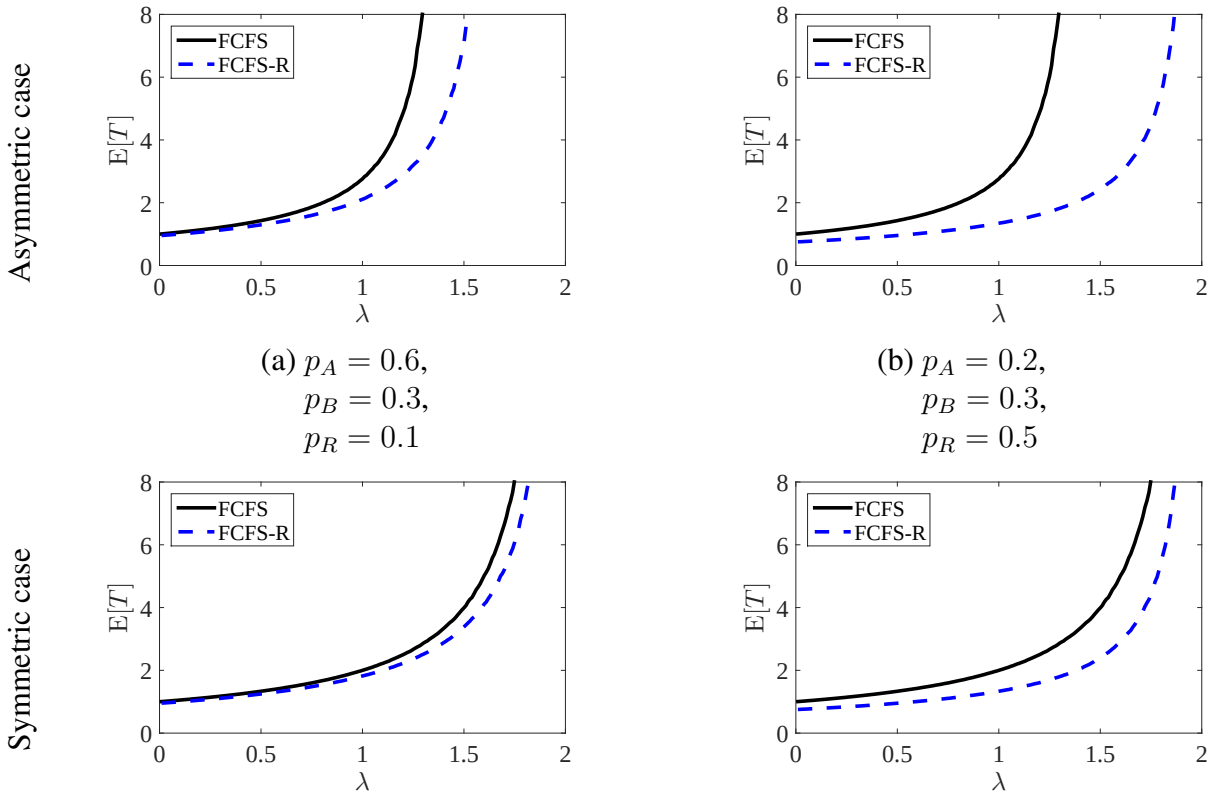
4.2.1 Objective 1: Low Overall Mean Response Time

We first evaluate the response time benefit achieved by the overall system from using redundancy. In the asymmetric case ($p_A = 0.7$, $p_B = 0.3$), allowing some class-A jobs to become redundant (i.e., increasing p_R and decreasing p_A) dramatically reduces mean response time for the overall system, and can even increase the system's stability region. The improvement is larger as the fraction of jobs that are redundant increases (see Figure 4(a),(b)). When the system initially is symmetric ($p_A = p_B = 0.5$), mean response time can be as much as 30% lower under FCFS-R than under non-redundant FCFS, but the advantage is not as great as in the asymmetric case (see Figure 4(c),(d)).

The difference in the magnitude of improvement in the two system configurations occurs because one of the primary benefits of redundancy is that it helps with load balancing. In the asymmetric system, allowing some class-A jobs to become redundant shifts some of the class-A load away from server 1 and onto server 2, which helps alleviate long queueing times experienced by the jobs that are served at server 1. In contrast, the symmetric system does not have an initial load imbalance, so there is not as much room for improvement in queueing time. While waiting in both queues can help the redundant jobs to experience shorter queueing times, in the symmetric case the service time reduction becomes a more significant component of the benefit offered by redundancy. When both class-A and class-B jobs become redundant and the system remains symmetric as p_R increases, an increasing number of jobs end up in service on both servers simultaneously and thus get to experience the minimum service time across the two servers.

4.2.2 Objective 2: Fairness Across Classes

Here we consider the per-class mean response times; our goal is for each class to perform at least as well in the redundancy system as under the initial non-redundant FCFS.



(a) $p_A = 0.6$, $p_B = 0.3$, $p_R = 0.1$

(b) $p_A = 0.2$, $p_B = 0.3$, $p_R = 0.5$

(c) $p_A = 0.45$, $p_B = 0.45$, $p_R = 0.1$

(d) $p_A = 0.25$, $p_B = 0.25$, $p_R = 0.5$

Figure 4: Comparing mean response time for the overall system under FCFS-R (dashed blue line) and under non-redundant FCFS (solid black line) in the asymmetric (top row) and symmetric (bottom row) cases.

We begin by considering the asymmetric case, in which in the nonredundant system 70% of the jobs are class-A and 30% are class-B, and in the redundant system some of the class-A jobs become redundant. It is easy to see by a sample-path argument that jobs that switch from class-A to class-R will benefit from the switch (Figures 5 and 6). Under FCFS-R, each class-R job retains the spot in first-come first-served order in the queue at server 1 that it held before it became redundant, while adding an opportunity to complete earlier on server 2. Note that there is a kink in the curve for class-R jobs under FCFS-R at $\lambda = 1.67$. This is because at $\lambda = 1.67$, the class-A jobs become overloaded ($\lambda \cdot p_A = 1 = \mu$), so none of the class-$R$ jobs end up in service on server 1. Instead, all class-R jobs effectively only get to join the queue at server 2, which becomes an M/M/1 with arrival rate $\lambda_B + \lambda_R$ and service rate μ. Thus at $\lambda = 1.67$ the mean response time for class-R jobs shifts from $\frac{1}{2\mu - \lambda_A - \lambda_B - \lambda_R}$ to $\frac{1}{\mu - \lambda_B - \lambda_R}$.

Class-A jobs also benefit from allowing some jobs to become redundant class-R jobs because the class-R jobs may leave the queue at server 1 without ever entering service. Unfortunately, the

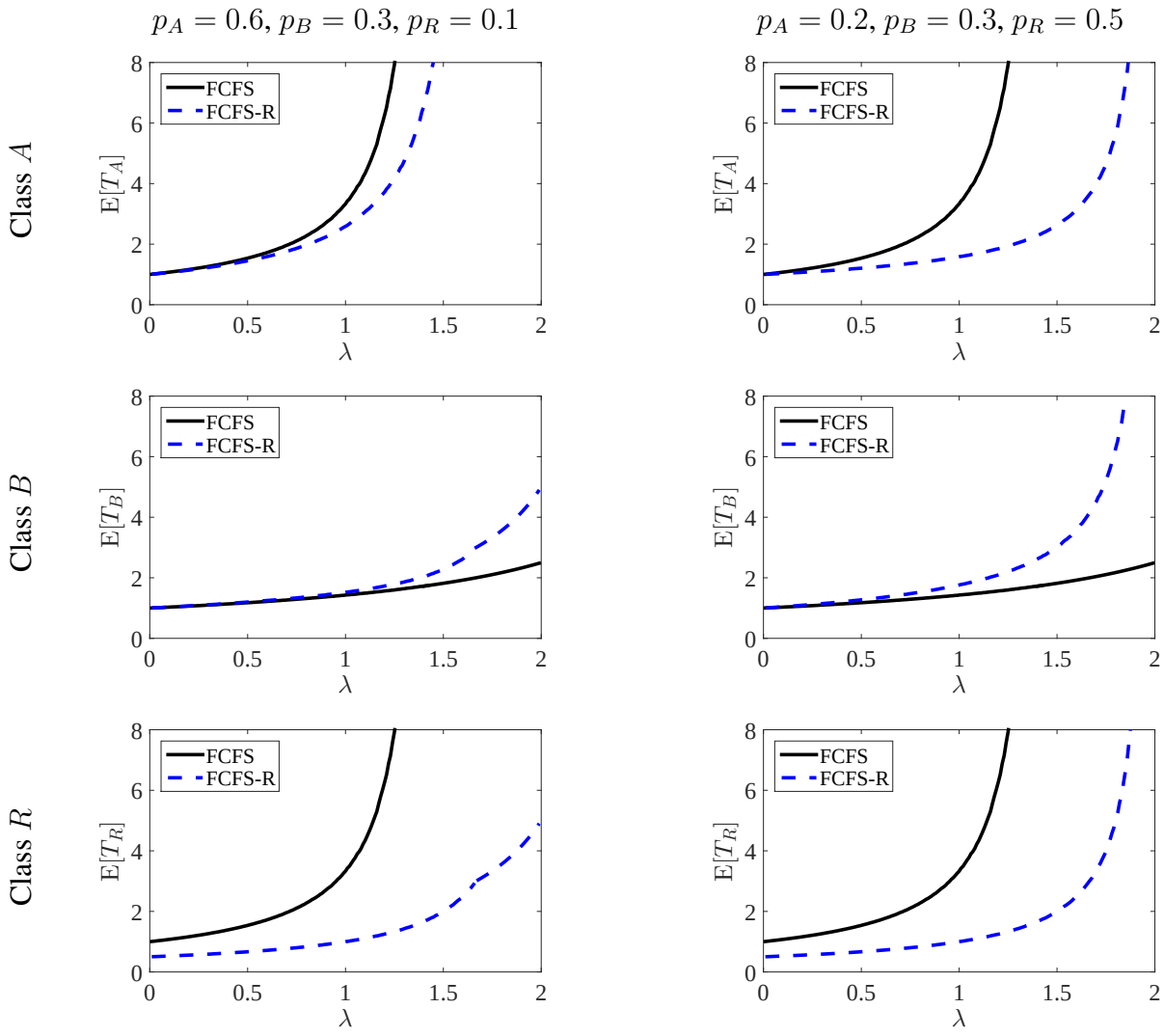


Figure 5: Comparing mean response time under FCFS-R (dashed blue line) and FCFS (solid black line) in the asymmetric case for class-A (top row), class-B (middle row), and class-R (bottom row) jobs. Note the kink at $\lambda = 1.67$ for class-B and class-R jobs when $p_A = 0.6$; this occurs because at $\lambda = 1.67$ the class-A jobs become unstable.

story is the opposite for the class-B jobs. The class-B jobs can be hurt by redundancy because the redundant jobs, which initially were class-A jobs, now join the queue at server 2 and take away some capacity from the class-B jobs. For example, when $p_R = 0.5$ and $\lambda = 1.4$ mean response time for the class-B jobs is 32% higher under FCFS-R than under non-redundant FCFS.

In the symmetric case, in which in the nonredundant system $p_A = p_B = 0.5$, and in the redundant system both class-A jobs and class-B jobs become redundant, the effects of redundancy are much less pronounced (see Figure 6). Here all three classes experience lower mean response time under FCFS-R than under non-redundant FCFS. But for the class-A and class-R jobs, the

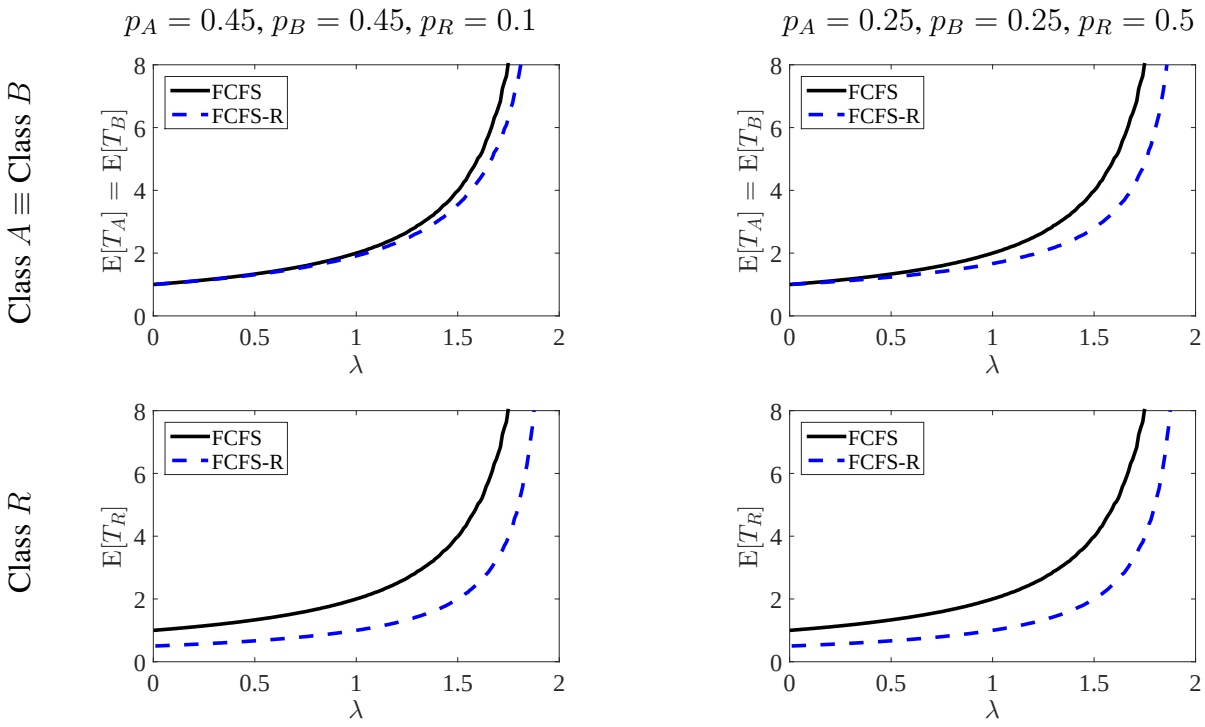


Figure 6: Comparing mean response time under FCFS-R (dashed blue line) and FCFS (solid black line) in the symmetric case for class-A and class-B (top row, classes A and B experience the same performance in the symmetric case), and class-R (bottom row).

improvement is far greater in the asymmetric case. Unlike in the asymmetric case, the class-B jobs are not hurt by redundancy in the symmetric case because here some class-B jobs get to become redundant, thereby benefiting from waiting in both queues and potentially running on both servers.

We see, from the pain experienced by the class-B customers in the asymmetric case, that, in general, FCFS-R *fails to achieve our second objective* of ensuring that all classes perform at least as well in the redundant system as in the non-redundant system.

5 Least Redundant First (LRF)

Our study of the FCFS-R policy reveals that using redundancy can lead to significant response time reductions relative to a system in which no jobs are redundant. In this section, we ask what is the *most* benefit we can achieve with respect to efficiency, i.e., reduction in overall system response time. Is FCFS-R the best we can do? Or is there some other policy that allows us to achieve even lower response times?

Intuitively, one important reason why redundancy helps is that it reduces the likelihood that servers are idle while there is still work in the system. While FCFS-R helps with load balancing, it

is possible for all of the redundant jobs to complete service, leaving behind a potentially unbalanced number of non-redundant jobs at each server. If this imbalance is large enough, one server can idle while others still have many jobs in the queue. We can alleviate this problem by preemptively prioritizing less-redundant jobs so that we preserve redundancy to prevent idling in the future. We call this idea the Least Redundant First (LRF) scheduling policy. We will show that LRF minimizes overall response time for any general exogenous arrival process.

5.1 Policy and Optimality

Under LRF, at all times each server works on the job in its queue that is redundant at the fewest queues. Formally, for an arbitrary server s, let $J_s = \{i : \ s \in S_i\}$ be the set of job classes that replicate to server s, as defined in Section 3. Under LRF, server s gives priority to jobs of class i over jobs of class $j \neq i$ if $i, j \in J_s$ and $|S_i| < |S_j|$. Note that this policy is well defined due to the nested structure: it is not possible to have $i, j \in J_s$ and $|S_i| = |S_j|$ unless $i = j$.

Intuitively, from the overall system's perspective LRF achieves low mean response time by preserving redundant jobs as long as possible (without idling) to maximize system efficiency. Under LRF, a redundant job only enters service at a particular server if there are no less-redundant jobs in the queue at that server, meaning that the jobs that stay in the system the longest are those that have the most options. This makes it much less likely under LRF than under non-redundant FCFS or even under FCFS-R that servers will idle while there is still work in the system. Indeed, we prove that LRF is optimal with respect to overall mean response time among all preemptive policies (Theorem 2).

Our proof relies on tracking the number of jobs of each class in the system at all times t for a given sample path. We show that for all job classes i and at all times t, if we count up the number of jobs that are class i or that share servers with but are less redundant than class i, this number is smaller under LRF than under any other policy. This tells us that LRF stochastically maximizes the number of departures from the system by time t, for all t, and so also minimizes mean response time (Corollary 1). Let $N_i(t)$ be the number of class-i jobs in the system at time t, and let $N^i(t) = \sum_{j:S_j \subseteq S_i} N_j(t)$ be the total number of class-i jobs plus those jobs that have priority over class-i jobs at any server (i.e., the jobs that are less redundant than class-i jobs and share a server with class-i jobs). Finally, let $\vec{N}(t) = (N^1(t), N^2(t), \ldots, N^\ell(t))$, where ℓ is the number of job classes. Unlike our analysis of FCFS-R, the proof of optimality for LRF does not require Poisson arrivals; we simply assume that arrivals are an arbitrary exogenous process.

Theorem 2 *The preemptive non-idling LRF policy stochastically minimizes $\{\vec{N}(t)\}_{t=0}^{\infty}$, among all preemptive, possibly idling, policies when service times are exponential, the arrivals form a general exogenous process, and the redundancy structure is nested.*

Proof. We first show that idling is non-optimal. Suppose that some arbitrary scheduling policy π idles server s at time 0 when there is a job of class i in the system such that $s \in S(i)$. Let an alternative scheduling policy π' serve the job of class i on server s and otherwise agree with π at time 0. Let $\delta > 0$ be the first time an event (arrival, service completion, or preemption) occurs

under π, and let $Y \sim \text{Exp}(\mu_s)$ be the next completion time on server s under π' (at which time our class-i job will depart).

Case 1: $Y > \delta$**.** Let π' agree with π for all scheduling decisions beginning at time δ, and couple all random variables under π and π'. Then $\{\vec{N}'(t)\}_{t=0}^{\infty} = \{\vec{N}(t)\}_{t=0}^{\infty}$ almost surely.

Case 2: $Y \leq \delta$**.** From time Y on, whenever any server serves our class-i job under π, let the corresponding server idle under π', and let π' otherwise agree with π, again coupling all random variables under π and π'. Then our class-i job completes earlier under π' than under π, while all other jobs complete at the same time, so $\{\vec{N}'(t)\}_{t=0}^{\infty} \leq \{\vec{N}(t)\}_{t=0}^{\infty}$ almost surely.

We can repeat this argument at each time at which a server idles.

We next show that LRF is optimal among all non-idling policies. Suppose that some arbitrary scheduling policy π serves a class-j job on server s at time 0 when there is a class-i job in the system such that $s \in S(i) \subset S(j)$. Let an alternative scheduling policy π' serve the class-i job on server s and otherwise agree with π at time 0. As before, let δ be the first time an event occurs under π, and let Y be the next completion time on server s (note that under π this completion is of a class-j job, whereas under π' it is a class-i job).

Case 1: $Y > \delta$**.** Let π' agree with π for all scheduling decisions beginning at time δ, and couple all random variables under π and π'. Then $\{\vec{N}'(t)\}_{t=0}^{\infty} = \{\vec{N}(t)\}_{t=0}^{\infty}$ almost surely.

Case 2: $Y = \delta$**.** From time Y on, whenever any server serves our class-i job under π (note that this job has departed under π'), let the corresponding server serve our class-j job under π' (which is possible because $S(i) \subset S(j)$, and let π' otherwise agree with π. Then $N'^i(Y) < N^i(Y)$ and $N'^j(Y) = N^j(Y)$. More generally, repeating the argument at each time at which a server serves a more-redundant job instead of a less-redundant job that it could serve, we find that $\{\vec{N}'(t)\}_{t=0}^{\infty} \leq \{\vec{N}(t)\}_{t=0}^{\infty}$ almost surely, and hence it is optimal for each server to always serve the least redundant job among those it can serve. This policy is exactly LRF. □

Corollary 1 *LRF stochastically maximizes the number of departures (job completions) by time t, for all t, among all preemptive, possibly idling, policies, and therefore also minimizes mean response time.*

While our focus is on nested systems, it is worth noting that our optimality proof for LRF holds for any nested substructure, even in general, non-nested systems. For example, non-redundant jobs are always nested relative to all other job classes, so they should be given highest priority.

Corollary 2 *For* any *redundancy system (not just nested systems), any scheduling policy that minimizes mean response time must give highest preemptive priority to job classes i with $|S_i| = 1$, i.e., non-redundant job classes.*

5.2 Analysis

In general, analyzing per-class and overall system response time under LRF seems to be intractable. For non-redundant classes, analysis is possible because these classes are isolated from all other jobs in the system; we provide exact expressions for the response time distribution for all non-redundant classes under LRF scheduling.

Lemma 5 *Under LRF, any non-redundant class i that runs only on server j has response time $T_i \sim \text{Exp}(\mu_j - \lambda_i)$.*

Proof. The non-redundant class-i jobs have highest preemptive priority among all jobs that share server j. Hence these jobs see an M/M/1 with arrival rate λ_i and service rate μ_j. □

Redundant classes and the overall system are more difficult to analyze exactly because a redundant job's response time is the minimum of the time it would experience across multiple queues, and these queues are not independent. Instead, we derive bounds on mean response time for redundant jobs and the overall system. Here we consider the $\mathbb{W}$ model, but our approach extends to other nested systems.

Theorem 3 *Under LRF scheduling in the $\mathbb{W}$ model, the overall system mean response time, $\mathbf{E}\left[T^{\text{LRF}}\right]$, is bounded by:*

$$\max\left\{\frac{1}{\mu-\lambda}, \frac{1}{\lambda}\left(\frac{\lambda_R}{\mu-\lambda} + \frac{\lambda_A}{\mu_1-\lambda_A} + \frac{\lambda_B}{\mu_2-\lambda_B}\right)\right\}$$
$$\leq \mathbf{E}\left[T^{\text{LRF}}\right] \leq \frac{1}{\lambda}\left(\frac{\lambda}{\mu-\lambda} + \frac{\lambda_A}{\mu_1-\lambda_A} + \frac{\lambda_B}{\mu_2-\lambda_B} - \frac{\lambda_A+\lambda_B}{\mu-\lambda_A-\lambda_B}\right),$$

where $\mu = \mu_1 + \mu_2$ and $\lambda = \lambda_A + \lambda_B + \lambda_R$.

Proof. Mean response time is minimized when all jobs are fully redundant (see, e.g., [8]). If all jobs are fully redundant, the system is equivalent to a single M/M/1 with arrival rate λ and service rate $\mu_1 + \mu_2$. This gives us the first term in our lower bound. The second term results from individual per-class bounds: the response time for class-i must be at least that in an M/M/1 with arrival rate λ_i and service rate $\sum_{s\in S_i} \mu_s$.

The upper bound is the overall system mean response time under FCFS-R (derived in Theorem 1); that it is an upper bound follows from the optimality of LRF (Theorem 2). □

Combining Lemma 5 and Theorem 3 gives us Theorem 4, which provides bounds on the class-R mean response time under LRF.

Theorem 4 *Under LRF in the $\mathbb{W}$ model, the mean response time for class-R, $\mathbf{E}\left[T_R^{\text{LRF}}\right]$, is bounded by*

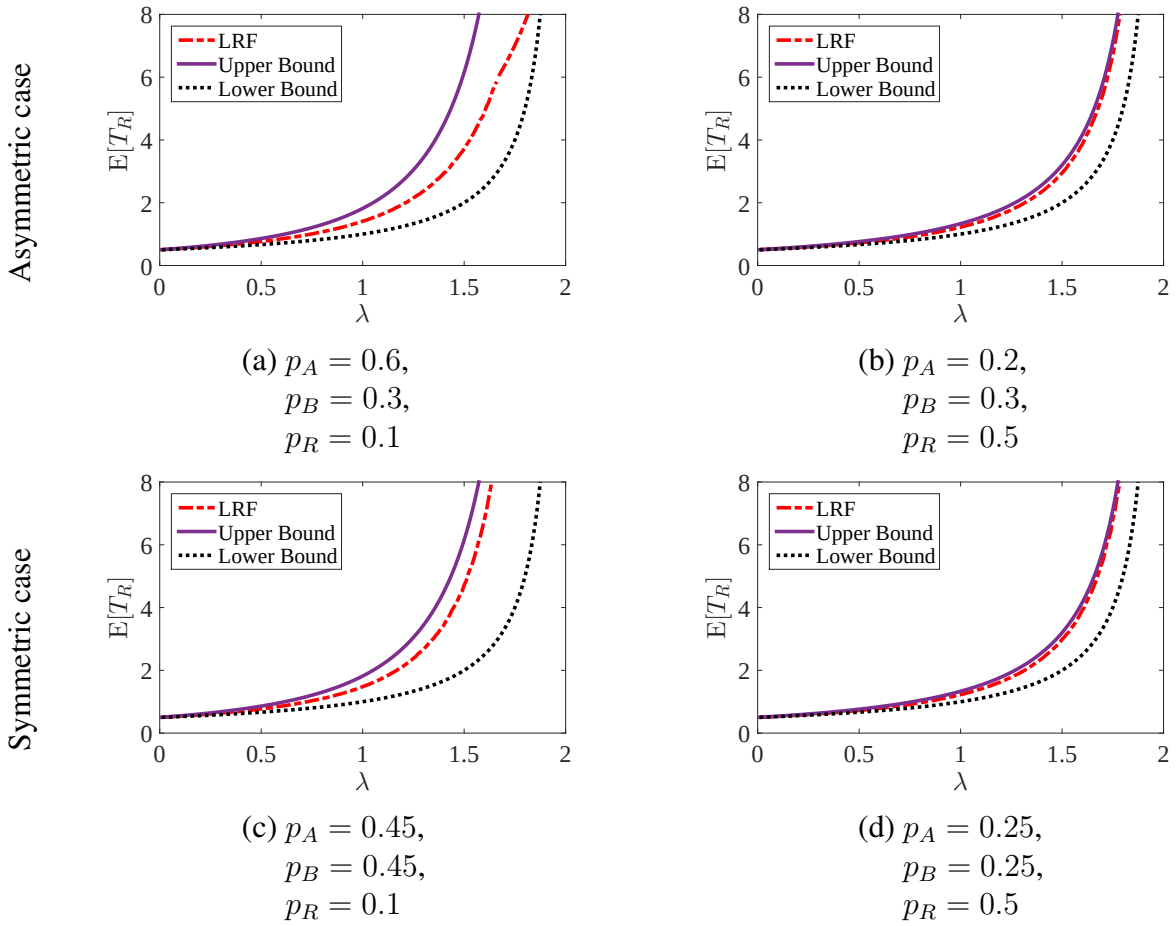
$$\frac{1}{\mu-\lambda} \leq \mathbf{E}\left[T_R^{\text{LRF}}\right] \leq \frac{1}{\lambda_R}\left(\frac{\lambda}{\mu-\lambda} - \frac{\lambda_A+\lambda_B}{\mu-\lambda_A-\lambda_B}\right).$$

Proof. We know that

$$\mathbf{E}\left[T^{\text{LRF}}\right] = \frac{\lambda_A}{\lambda}\mathbf{E}\left[T_A^{\text{LRF}}\right] + \frac{\lambda_B}{\lambda}\mathbf{E}\left[T_B^{\text{LRF}}\right] + \frac{\lambda_R}{\lambda}\mathbf{E}\left[T_R^{\text{LRF}}\right].$$

Combining this with the exact forms for $\mathbf{E}\left[T_A^{\text{LRF}}\right]$ and $\mathbf{E}\left[T_B^{\text{LRF}}\right]$ given in Lemma 5 and the upper bound for $\mathbf{E}\left[T^{\text{LRF}}\right]$ given in Theorem 3 immediately yields the upper bound on $\mathbf{E}\left[T_R^{\text{LRF}}\right]$. The lower bound is the response time for class-R jobs under FCFS-R, and it is clear that giving class-R jobs lowest priority can only increase their response time. □

Figure 7 shows the quality of the bounds on $\mathbf{E}\left[T_R\right]^{\text{LRF}}$ in both the asymmetric and symmetric cases described in Section 3. The upper bound becomes increasingly tight as a larger fraction of jobs become redundant.



(a) $p_A = 0.6$, $p_B = 0.3$, $p_R = 0.1$

(b) $p_A = 0.2$, $p_B = 0.3$, $p_R = 0.5$

(c) $p_A = 0.45$, $p_B = 0.45$, $p_R = 0.1$

(d) $p_A = 0.25$, $p_B = 0.25$, $p_R = 0.5$

Figure 7: Bounds on class-R mean response time under LRF in the asymmetric (top row) and symmetric (bottom row) cases. As the fraction of class-R jobs increases, the upper bound becomes increasingly tight.

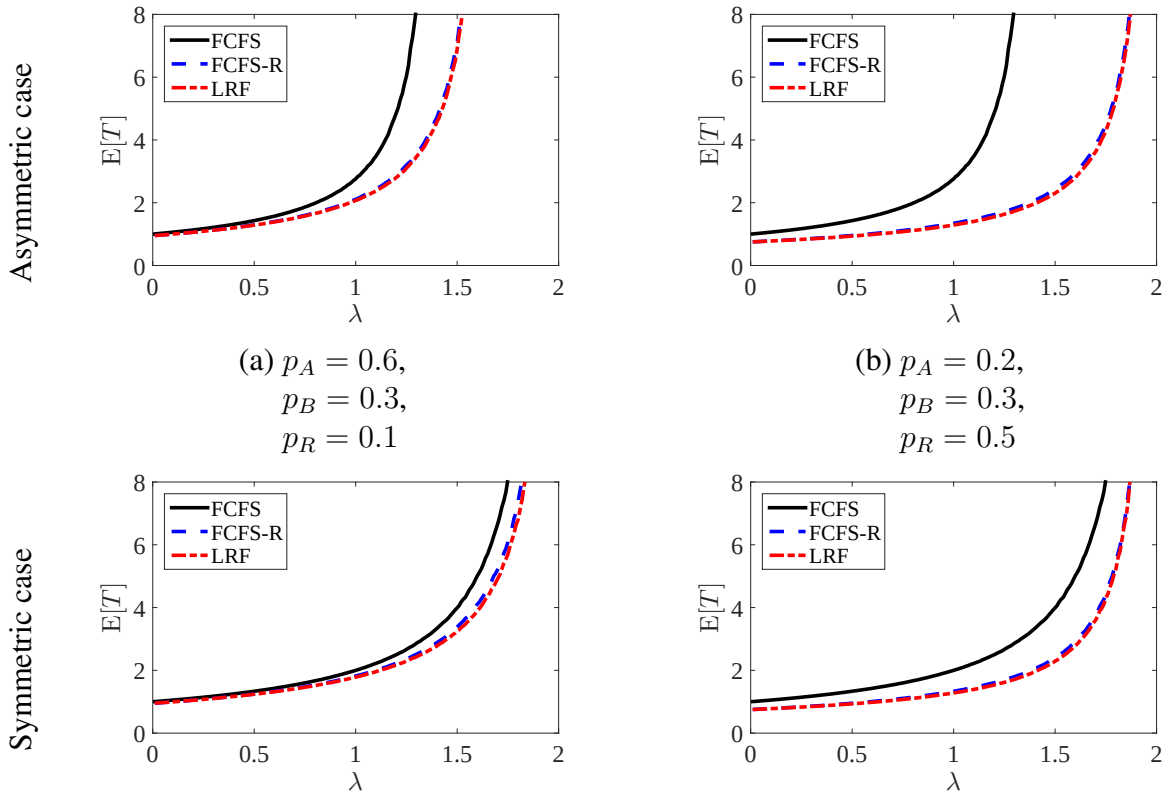
5.3 Performance

To evaluate the performance of LRF, we again turn to the $\mathbb{W}$ model. As before, we consider two system configurations: the asymmetric system where only class-A jobs become redundant, and the symmetric system where both class-A and class-B jobs become redundant.

5.3.1 Objective 1: Low Overall Mean Response Time

Given that LRF is optimal with respect to overall mean response time (Theorem 2), it is unsurprising that Figure 8(d) shows that LRF outperforms both non-redundant FCFS and FCFS-R. What is surprising is that the gap is so small between FCFS-R and LRF for all of the system configurations.

To understand why, we turn to the bounds on mean response time under LRF (Theorem 3). Figure 9 shows that for nearly all parameter settings, the upper bound (which is equivalent to FCFS-R)



(a) $p_A = 0.6$, $p_B = 0.3$, $p_R = 0.1$

(b) $p_A = 0.2$, $p_B = 0.3$, $p_R = 0.5$

(c) $p_A = 0.45$, $p_B = 0.45$, $p_R = 0.1$

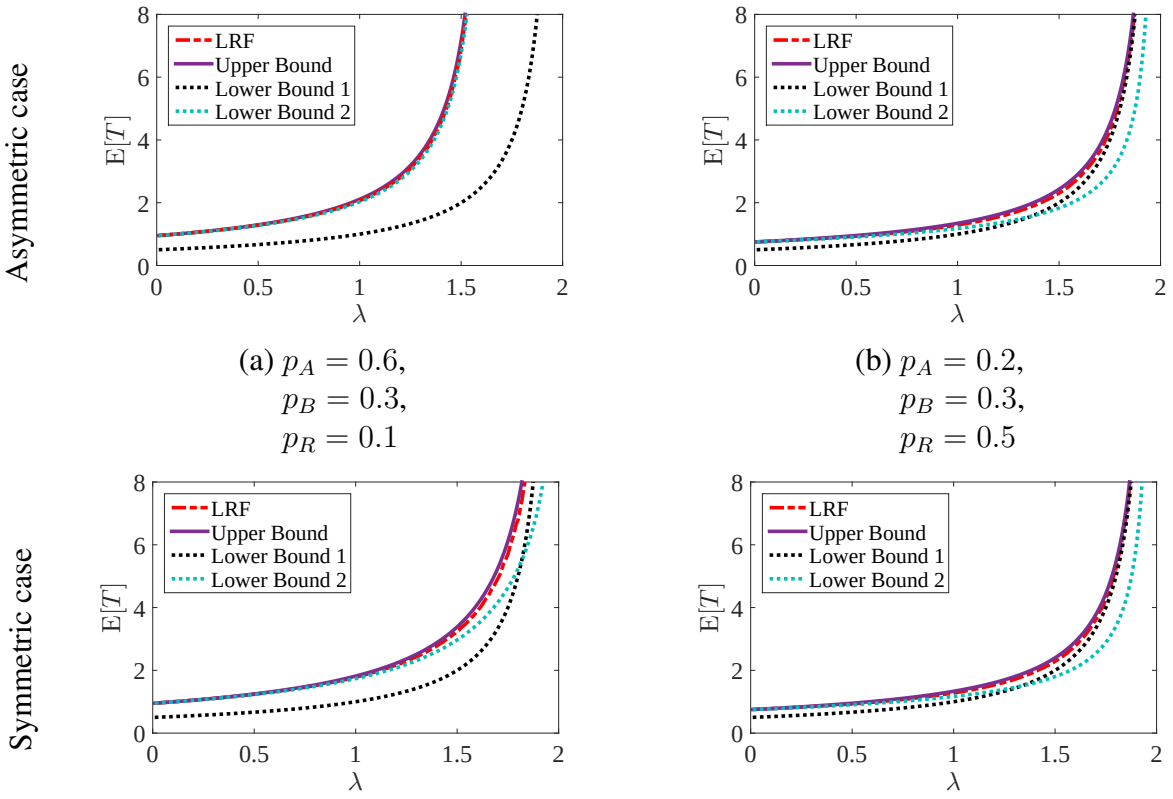
(d) $p_A = 0.25$, $p_B = 0.25$, $p_R = 0.5$

Figure 8: Comparing mean response time for the overall system under LRF (dot-dashed red line) to that under FCFS-R (dashed blue line) and non-redundant FCFS (solid black line) in the asymmetric (top row) and symmetric (bottom row) cases.

is extremely close to the lower bound. We see very little difference in overall mean response time from using different redundancy-based scheduling policies because *redundancy itself* is extremely powerful. The benefit of simply allowing some jobs to be redundant, thereby preventing servers from idling, is much greater than any additional benefit that can be achieved via sophisticated scheduling policies.

5.3.2 Objective 2: Fairness Across Classes

Under LRF, in the asymmetric case all three classes of jobs are better off than they were under FCFS-R (see Figure 10). The class-A jobs see a much more pronounced improvement relative to the non-redundant system under LRF than under FCFS-R, particularly as the fraction of class-R jobs increases. Under FCFS-R the class-A jobs approach instability as λ approaches 2, whereas the class-A mean response time remains very low under LRF even as λ gets high because class-A



(a) $p_A = 0.6$, $p_B = 0.3$, $p_R = 0.1$

(b) $p_A = 0.2$, $p_B = 0.3$, $p_R = 0.5$

(c) $p_A = 0.45$, $p_B = 0.45$, $p_R = 0.1$

(d) $p_A = 0.25$, $p_B = 0.25$, $p_R = 0.5$

Figure 9: Upper and lower bounds for overall system mean response time under LRF in the asymmetric (top row) and symmetric (bottom row) cases. Here we show both terms in the lower bound given in Theorem 4.

jobs have preemptive priority over class-R jobs. Similarly, under LRF the class-B jobs are not hurt by the class-R jobs. Class-B jobs experience exactly the same mean response time under LRF as under non-redundant FCFS because under both policies, there are no class-R jobs competing with the class-B jobs, which have preemptive priority over class-R jobs.

The class-R jobs, too, benefit significantly from getting to wait in both queues. Even though the class-R jobs have lowest priority in both queues, they are likely to be preempted by fewer class-B jobs at server 2 than class-A jobs at server 1. When λ_B is sufficiently low, the class-R jobs are better off having low priority behind the class-B jobs than they would have been waiting in FCFS order among the class-A jobs. Note that, as was the case for FCFS-R, there is a kink in the curve for class-R jobs at $\lambda = 1.67$. Again, this is because at $\lambda = 1.67$ the class-A jobs become unstable, so the class-R jobs effectively only join the queue at server 2. Server 2 then behaves like a priority queue in which class-B jobs have preemptive priority over class-R jobs. Mean response

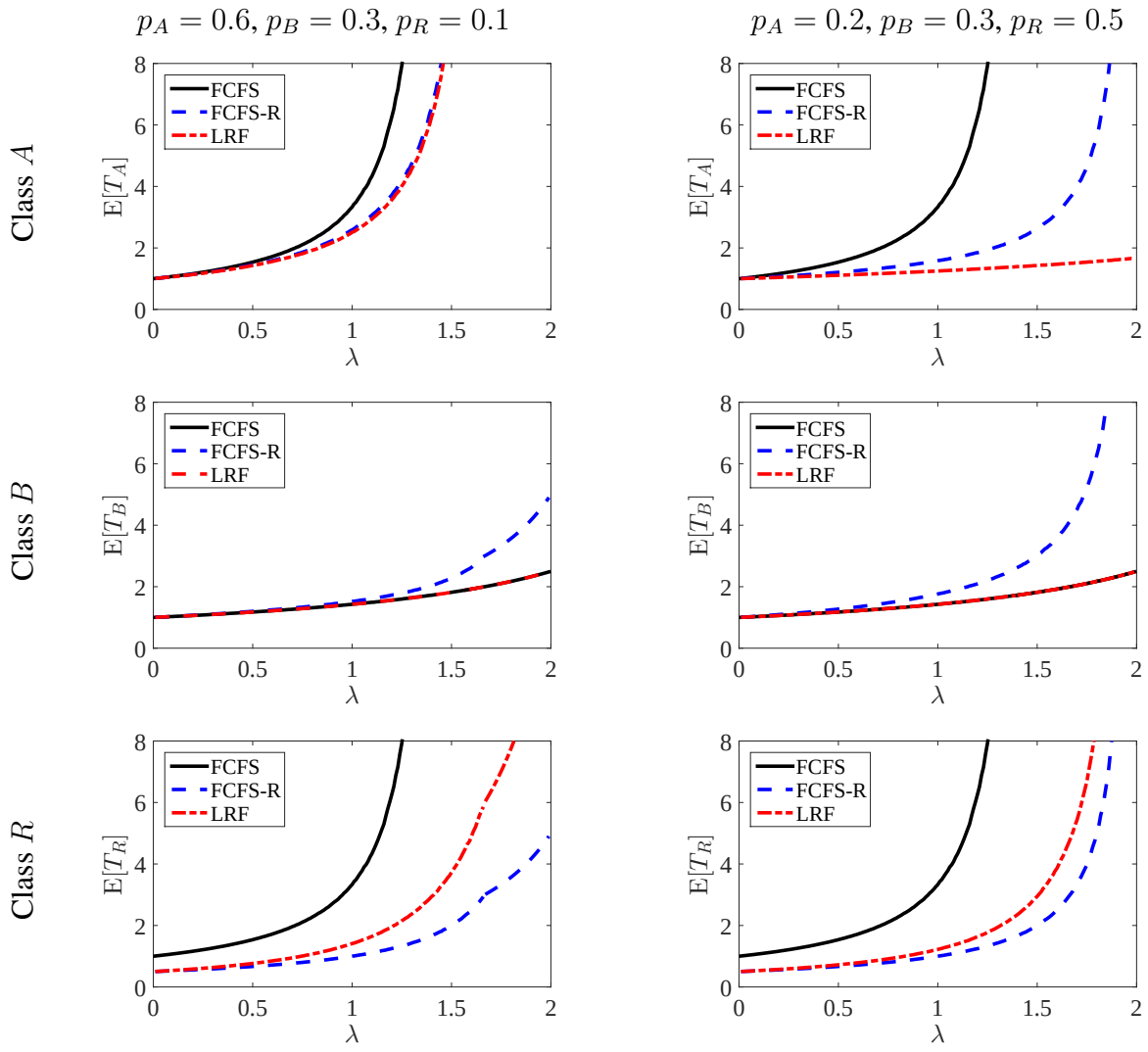


Figure 10: Comparing mean response time under LRF (dot-dashed red line) to that under FCFS-R (dashed blue line) and FCFS (solid black line) in the asymmetric case for class-A (top row), class-B (middle row), and class-R (bottom row) jobs. Note that there is a kink at $\lambda = 1.67$ for the class-R jobs when $p_A = 0.6$ because at this point the class-A jobs become unstable.

time for the low-priority class-R jobs in this system is

$$\mathbf{E}\left[T_R\right] = \frac{1}{\mu - \lambda_B} + \frac{\lambda_B + \lambda_R}{(\mu - \lambda_B)(\mu - \lambda_B - \lambda_R)}.$$

Unfortunately, the symmetric case reveals LRF's inability to satisfy our fairness objective in general (see Figure 11). LRF is able to achieve optimal overall mean response time and avoid hurting the non-redundant jobs by forcing the redundant jobs to have lowest preemptive priority. This means that redundant jobs only receive service when there are no non-redundant jobs at the

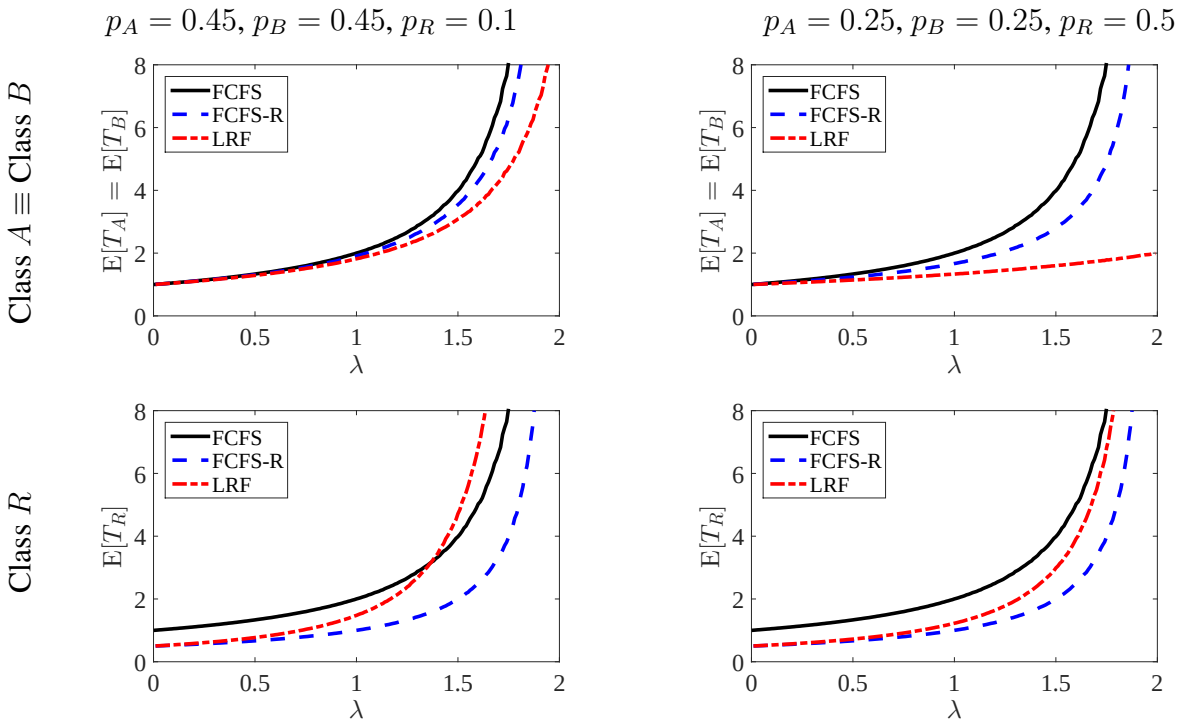


Figure 11: Comparing mean response time under LRF (dot-dashed red line) to that under FCFS-R (dashed blue line) and FCFS (solid black line) in the symmetric case for class-A and class-B (top row, classes A and B experience the same performance in the symmetric case), and class-R (bottom row).

server. When the system load is sufficiently high, both the class-A and class-B busy periods can be very long. Hence the redundant jobs can be starved and experience higher mean response times than they would have if they had remained class-A jobs under non-redundant FCFS. For example, in the symmetric case, when $p_R = 0.1$ and $\lambda = 1.6$, mean response time for the class-R jobs is 36% higher under LRF than under non-redundant FCFS. Thus like FCFS-R, LRF is unable to achieve our goal of fairness across job classes for all parameter values.

6 Primaries First (PF)

In Sections 4 and 5, we saw that although both FCFS-R and LRF achieve near-optimal and optimal performance with respect to overall mean response time, they both fail to achieve our second objective of maintaining fairness across classes. Specifically, FCFS-R may penalize the class-B jobs by forcing them to wait behind the class-R jobs that become redundant. LRF tries to compensate for this by prioritizing the non-redundant jobs. However, this can cause the class-R jobs to experience higher mean response times than if they were not redundant.

The Primaries First (PF) policy is designed to balance the strengths of both LRF and FCFS.

Under PF, each arriving job designates a single *primary* copy, and any additional replicas are designated *secondary* copies. At each server, primaries have preemptive priority over secondaries, regardless of job class. Within the primaries (respectively, secondaries) jobs are served in FCFS order. When comparing PF with a system in which no jobs are redundant, we assume that jobs that become redundant designate their primary to be the copy that joins the queue at the server that the job would have gone to in the corresponding non-redundant system (e.g., in the asymmetric $\mathbb{W}$ model, class-R jobs have their primary copy on server 1).

Intuitively, PF is successful at balancing the dual goals of achieving good overall performance and preserving fairness among classes because PF represents a compromise between LRF, which achieves optimal overall performance at the expense of the redundant jobs, and FCFS-R, which tries to be fair to the redundant jobs but consequently can hurt the non-redundant jobs and the overall system. Like LRF, PF only allows extra copies to run when there is spare server capacity. This prevents PF from hurting non-redundant jobs. Like FCFS, PF allows one copy of each redundant job to hold its place in the system-wide FCFS ordering. This prevents PF from hurting redundant jobs. In Theorem 5 we formalize this intuition by proving that under PF, every job class has response time at least as low as its response time under (non-redundant) FCFS. Unlike our results for FCFS-R and LRF, the proof of Theorem 5 is quite general: service times can be generally distributed, service times for the same job can be correlated across servers, the arrival process can be any exogenous process, and the redundancy structure need not be nested.

Theorem 5 *For all job classes i, $T^{(i)}_{\mathrm{PF}} \leq_{st} T^{(i)}_{\mathrm{FCFS}}$ for any general service times and any exogenous arrival process.*

Proof. On any sample path, if PF never schedules any secondaries, then all jobs complete at the same time under PF as in the system with no redundancy. Suppose a secondary copy of job i is scheduled at some time under PF, say on server s. This means that server s is empty of primaries. If a different copy of job i completes on a different server before completing on server s, then job i and all other jobs continue to have the same completion times under PF as in the system with no redundancy. If job i completes on server s, then all of its copies disappear immediately from the system (including, in particular, its primary copy, which we will say disappeared from server s'). Then server s' now has one fewer job under PF than under non-redundant FCFS, while all other queues are the same length under both policies (ignoring all other jobs' secondaries). Hence the jobs at server s' will all complete earlier under PF than under non-redundant FCFS.

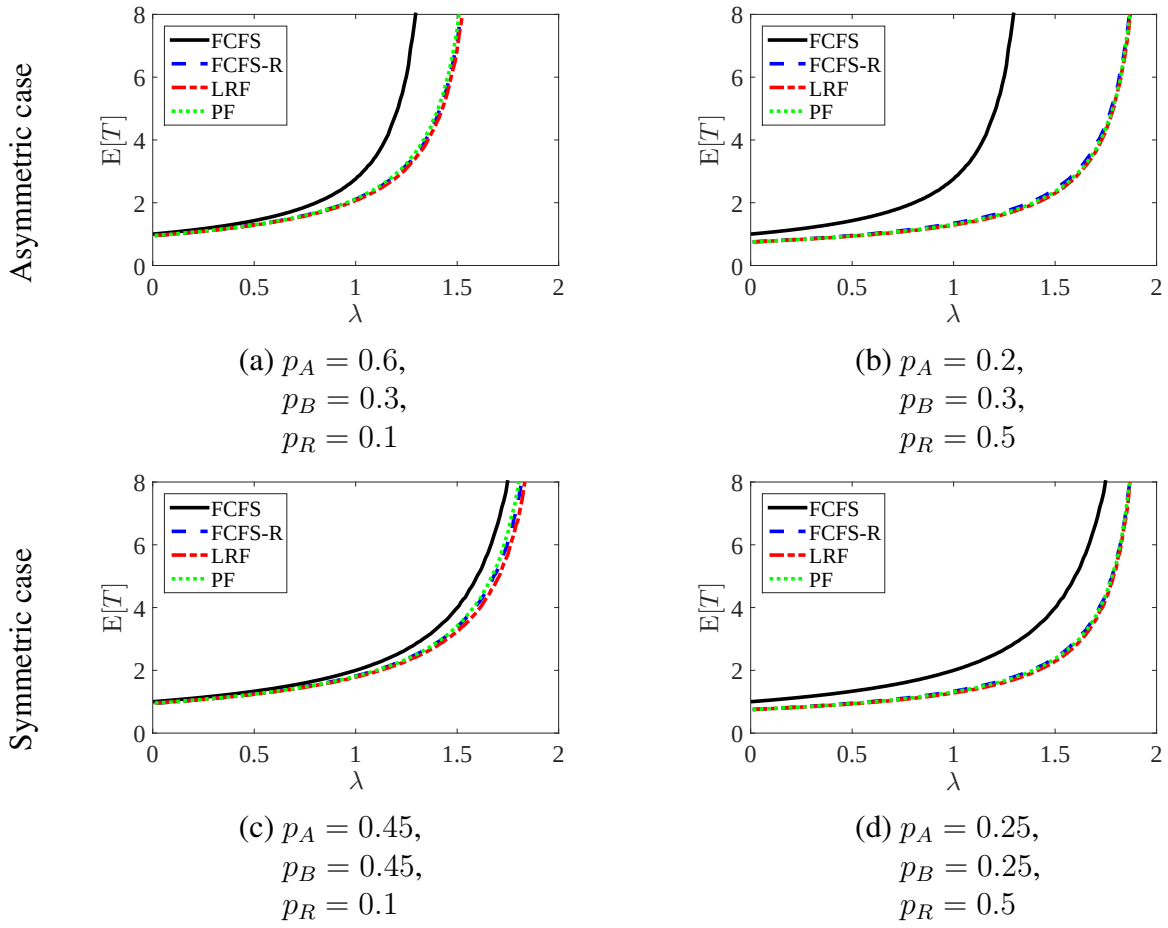
The result follows from repeating this argument each time PF schedules a secondary copy. □

6.1 Performance

We again consider the $\mathbb{W}$ model, shown in Figure 3(d) with redundancy and PF scheduling.

6.1.1 Objective 1: Low Overall Mean Response Time

PF successfully meets our first objective of achieving low overall mean response time (Figure 12): like FCFS-R and LRF, PF always outperforms non-redundant FCFS. One might think that since



(a) $p_A = 0.6$, $p_B = 0.3$, $p_R = 0.1$

(b) $p_A = 0.2$, $p_B = 0.3$, $p_R = 0.5$

(c) $p_A = 0.45$, $p_B = 0.45$, $p_R = 0.1$

(d) $p_A = 0.25$, $p_B = 0.25$, $p_R = 0.5$

Figure 12: Comparing mean response time for the overall system under PF (dotted green line) to that under LRF (dot-dashed red line), FCFS-R (dashed blue line) and FCFS (solid black line) for both the asymmetric (top row) and symmetric (bottom row) cases.

PF uses FCFS for primary copies, yet gives secondary copies lowest priority as in LRF, its mean response time should lie between that under FCFS-R and LRF. Unfortunately, this is not the case, particularly when load is high and the fraction of redundant jobs is relatively small (see Figure 12(c)). PF load balances by dispatching each job to a single server, and then adding extra low-priority secondary copies of the redundant jobs at other servers. When the arrival rate is high, it is fairly unlikely that the secondary copies get to run, so most redundant jobs experience the same performance as they did as class-A jobs in the original non-redundant system. In contrast, under FCFS-R a redundant job enters service at whichever server has less work in the queue when the job arrives, which reduces queueing time relative to the original system. As we increase the proportion of redundant jobs or decrease load, we increase the likelihood that the secondary copies actually enter service under PF, and then PF starts doing better than FCFS-R.

We observe empirically that mean response time under PF is quite similar to that under the

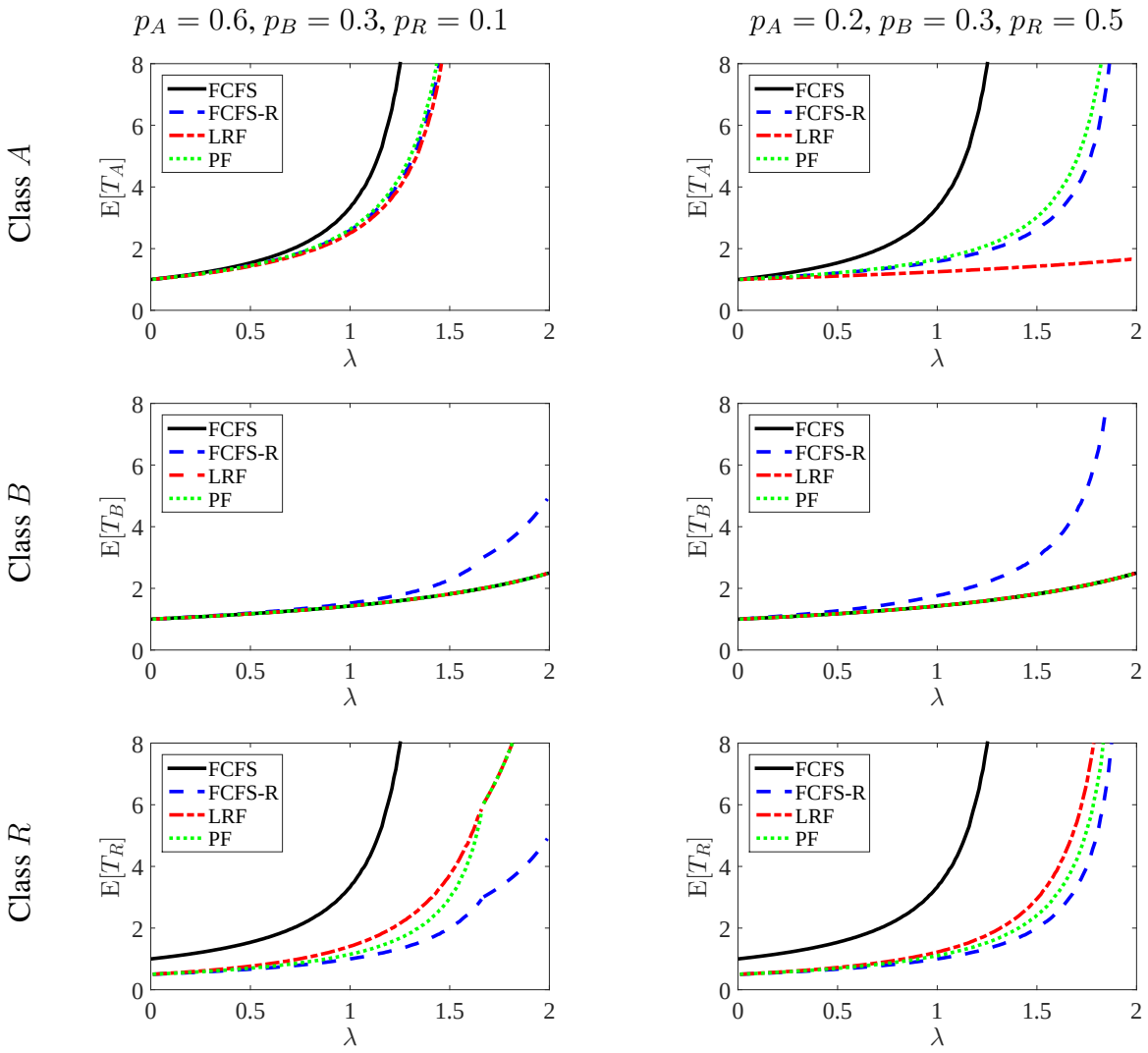


Figure 13: Comparing mean response time under PF (dotted green line) to that under LRF (dot-dashed red line), FCFS-R (dashed blue line) and FCFS (solid black line) in the asymmetric case for class-A (top row), class-B (middle row), and class-R (bottom row) jobs.

optimal LRF policy. Again, this is because redundancy inherently is so powerful at keeping servers from idling that the particular scheduling policy does not play a significant role in further reducing mean response time.

6.1.2 Objective 2: Fairness Across Classes

PF successfully meets our second objective of preserving fairness across job classes in both the asymmetric case (Figure 13) and the symmetric case (Figure 14): as we saw in Theorem 5, all classes experience mean response times that are no worse than that under non-redundant FCFS.

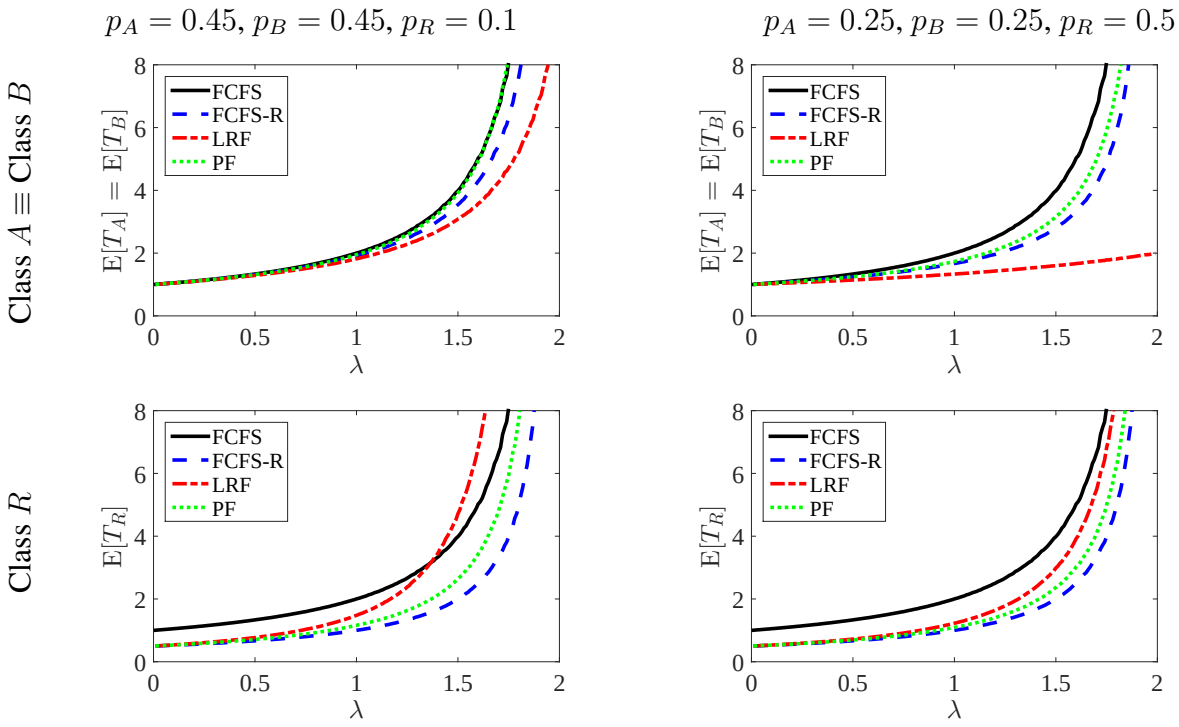


Figure 14: Comparing mean response time under PF (dotted green line) to that under LRF (dot-dashed red line), FCFS-R (dashed blue line) and FCFS (solid black line) in the symmetric case for class-A and class-B (top row, classes A and B experience the same performance in the symmetric case), and class-R (bottom row).

7 Conclusions

In this paper we study scheduling policies in systems with redundant requests. We consider class-based systems, in which each job has a class that determines the set of servers to which the job replicates itself. Our goal is to design policies that simultaneously satisfy the two objectives of (1) *efficiency*, i.e., low overall mean response time and (2) *fairness* across classes, i.e., each job class should perform at least as well under our redundancy-based scheduling policies as in a system in which no jobs are redundant. The second objective is motivated by systems such as organ transplant waitlists, in which a job's ability to become redundant correlates with the job's socioeconomic status; here it is important to ensure that we do not improve response time for the system as a whole at the expense of jobs that cannot become redundant. Our first scheduling policy is First-Come, First-Served with redundancy (FCFS-R). We derive exact, closed-form expressions for the per-class distribution of response time, which we use to show that FCFS-R is highly effective at reducing the overall system mean response time. But it is possible to do even better: we introduce the Least Redundant First (LRF) policy, which we prove is optimal with respect to minimizing overall mean response time. LRF prevents servers from idling while there is still work in the system, thereby maximizing the system efficiency. But surprisingly, FCFS-R achieves nearly the

optimal overall mean response time. This suggests that the benefits offered by redundancy itself are so substantial that there is limited room for the particular scheduling policy to greatly affect mean response time.

On the other hand, scheduling plays a crucial role in achieving our second objective of fairness across classes. Under FCFS-R non-redundant jobs can be hurt when some jobs become redundant due to the extra load added by the redundant jobs. LRF prevents the non-redundant jobs from being hurt by giving them full preemptive priority, but this can cause the redundant jobs to suffer. To overcome these weaknesses, we introduce the Primaries First (PF) policy, under which each job designates one copy as its primary and all other copies as secondaries; primaries have preemptive priority over secondaries on all servers. We prove that PF successfully maintains fairness across all job classes. Furthermore, PF accomplishes this while still providing low overall mean response time, thereby achieving our first goal as well.

Our analyses of FCFS-R and LRF rely on several assumptions, including that service times are exponentially distributed and independent across servers, and that the system has a nested structure; these assumptions may not hold in practice. Nonetheless, we believe the results presented here represent an important step towards understanding the role scheduling plays in systems with redundancy. We leave incorporating features such as general service times and correlated service times, as well as designing optimal policies in non-nested systems, for future work. Importantly, however, our fairness results under PF do *not* require these assumptions. PF continues to satisfy our desired fairness property when service times are generally distributed and correlated across servers, when the arrival process is non-Poisson, and when the system is not nested.

8 Acknowledgments

This work was supported by the NSF under grants NSF-XPS-1629444, NSF-CMMI-1538204, and NSF-CMMI-1334194; by a Google Faculty Research Award; by a Facebook Faculty Gift; by a Google Anita Borg Memorial Scholarship; and by the Siebel Scholars Program.

References

[1] Ivo Adan and Gideon Weiss. A skill based parallel service system under FCFS-ALIS - steady state, overloads, and abandonments. *Stochastic Systems*, 4(1):250–299, 2014.

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

[3] Eric J Friedman and Shane G Henderson. Fairness and efficiency in web server protocols. In *ACM SIGMETRICS Performance Evaluation Review*, volume 31, pages 229–237. ACM, 2003.

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

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

[6] Kristen Gardner, Samuel Zbarsky, Mor Harchol-Balter, and Alan Scheller-Wolf. Analyzing response time in the redundancy-d system. Technical report, Technical Report CMU-CS-15-141, 2015.

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

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

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

[10] William Stallings and Goutam Kumar Paul. *Operating systems: internals and design principles*, volume 3. prentice hall Upper Saddle River, NJ, 1998.

[11] Yin Sun, Zizhan Zheng, C Emre Koksal, Kyu-Han Kim, and Ness B Shroff. Provably delay efficient data retrieving in storage clouds. In *2015 IEEE Conference on Computer Communications (INFOCOM)*, pages 585–593. IEEE, 2015.

[12] Andrew S Tanenbaum and Albert S Woodhull. *Operating systems: design and implementation*, volume 2. Prentice-Hall Englewood Cliffs, NJ, 1987.

[13] Jeremy Visschers, Ivo Adan, and Gideon Weiss. A product form solution to a system with multi-type jobs and multi-type servers. *Queueing Systems*, 70:269–298, 2012.

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

A Proof of Lemma 3

Proof. Our proof is by induction on the number of jobs in subsystems $\mathcal{A}$ and $\mathcal{B}$. As our base case when $n_R \neq -$, assume there is one job in each subsystem, and call these jobs a_1 and b_1. Then there are two possible ways to interleave jobs a_1 and b_1, so we have

$$\begin{aligned} P(R, n_R, \binom{a_1}{b_1}) &= \sum_{c_1,\ldots,c_{n_R}} \left(P(c_{n_R}, \ldots, c_1, R, a_1, b_1) + P(c_{n_R}, \ldots, c_1, R, b_1, a_1)\right) \\ &= \sum_{c_1,\ldots,c_{n_R}} \left[\left(\frac{\lambda_{c_1}}{\mu}\right) \cdots \left(\frac{\lambda_{c_{n_R}}}{\mu}\right) \cdot \left(\frac{\lambda_R}{\mu}\right) \cdot \left(\frac{\lambda_{a_1}}{\mu_{\mathcal{I}_{a_1}} + \mu_{\mathcal{I}_{b_1}}}\right) \cdot \left(\frac{\lambda_{b_1}}{\mu_{\mathcal{I}_{b_1}}}\right) \right. \end{aligned}$$

$$
\begin{aligned}
&\qquad + \left(\frac{\lambda_{c_1}}{\mu}\right) \cdots \left(\frac{\lambda_{c_{n_R}}}{\mu}\right) \cdot \left(\frac{\lambda_R}{\mu}\right) \cdot \left(\frac{\lambda_{b_1}}{\mu_{\mathcal{I}_1} + \mu_{\mathcal{I}_{b_1}}}\right) \cdot \left(\frac{\lambda_{a_1}}{\mu_{\mathcal{I}_{a_1}}}\right)\Bigg] \\
&= \left(\frac{\lambda}{\mu}\right)^{n_R} \cdot \left(\frac{\lambda_R}{\mu}\right) \cdot \frac{\lambda_{a_1}\lambda_{b_1}(\mu_{\mathcal{I}_{a_1}} + \mu_{\mathcal{I}_{b_1}})}{(\mu_{\mathcal{I}_{a_1}} + \mu_{\mathcal{I}_{b_1}})\mu_{\mathcal{I}_{a_1}}\mu_{\mathcal{I}_{b_1}}} \\
&= \left(\frac{\lambda}{\mu}\right)^{n_R} \cdot \left(\frac{\lambda_R}{\mu}\right) \cdot \left(\frac{\lambda_{a_1}}{\mu_{\mathcal{I}_{a_1}}}\right) \cdot \left(\frac{\lambda_{b_1}}{\mu_{\mathcal{I}_{b_1}}}\right) \\
&= \left(\frac{\lambda}{\mu}\right)^{n_R} \cdot \left(\frac{\lambda_R}{\mu}\right) \cdot P(\mathcal{A}) \cdot P(\mathcal{B}),
\end{aligned}
$$

where $P(\mathcal{A}) = \frac{\lambda_{a_1}}{\mu_{\mathcal{I}_{a_1}}}$ (respectively, $P(\mathcal{B}) = \frac{\lambda_{b_1}}{\mu_{\mathcal{I}_{b_1}}}$) gives the (unnormalized) limiting probability of being in state (a_1) (respectively, (b_1)) in a system consisting of only the job classes and servers in subsystem $\mathcal{I}_{a_1}$ (respectively, $\mathcal{I}_{b_1}$).

Now suppose inductively that (3) holds for all $i < \ell_{\mathcal{A}}$, $j < \ell_{\mathcal{B}}$ jobs in subsystems $\mathcal{A}$ and $\mathcal{B}$. Then with $\ell_{\mathcal{A}} + 1$ jobs in subsystem $\mathcal{A}$, we have:

$$
\begin{aligned}
P\left(R, n_R, \substack{(a_{\ell_{\mathcal{A}}},\ldots,a_1)\\(b_{\ell_{\mathcal{B}}},\ldots,b_1)}\right) &= \sum_{c_1,\ldots,c_{n_R}} \Bigg[P\left(c_{n_R},\ldots,c_1,R,a_{\ell_{\mathrm{A}}+1}, \substack{(a_{\ell_{\mathcal{A}}},\ldots,a_1)\\(b_{\ell_{\mathcal{B}}},\ldots,b_1)}\right) \\
&\qquad + P\left(c_{n_R},\ldots,c_1,R,b_{\ell_{\mathrm{B}}},a_{\ell_{\mathrm{A}}+1}, \substack{(a_{\ell_{\mathcal{A}}},\ldots,a_1)\\(b_{\ell_{\mathcal{B}}-1},\ldots,b_1)}\right) \\
&\qquad + \cdots + P\left(c_{n_R},\ldots,c_1,R,b_{\ell_{\mathrm{B}}},b_{\ell_{\mathcal{B}}-1},\ldots,b_1,a_{\ell_{\mathrm{A}}+1}, \substack{(a_{\ell_{\mathcal{A}}},\ldots,a_1)\\(-)}\right)\Bigg] \\
&= \left(\frac{\lambda}{\mu}\right)^{n_R} \cdot \left(\frac{\lambda_R}{\mu}\right) \cdot \Bigg[\left(\frac{\lambda_{a_{\ell_{\mathcal{A}}+1}}}{\mu_{\bigcup_{i=1}^{\ell_{\mathcal{A}}+1}\mathcal{I}_{a_i}} + \mu_{\bigcup_{i=1}^{\ell_{\mathcal{B}}}\mathcal{I}_{b_i}}}\right) \cdot \left(\prod_{j=1}^{\ell_{\mathcal{A}}} \frac{\lambda_{a_j}}{\mu_{\bigcup_{i=1}^{j}\mathcal{I}_{a_i}}}\right) \cdot \left(\prod_{j=1}^{\ell_{\mathcal{B}}} \frac{\lambda_{b_j}}{\mu_{\bigcup_{i=1}^{j}\mathcal{I}_{b_i}}}\right) \\
&\quad + \left(\frac{\lambda_{b_{\ell_{\mathcal{B}}}}}{\mu_{\bigcup_{i=1}^{\ell_{\mathcal{A}}+1}\mathcal{I}_{a_i}} + \mu_{\bigcup_{i=1}^{\ell_{\mathcal{B}}}\mathcal{I}_{b_i}}}\right) \cdot \left(\frac{\lambda_{a_{\ell_{\mathcal{A}}+1}}}{\mu_{\bigcup_{i=1}^{\ell_{\mathcal{A}}+1}\mathcal{I}_{a_i}} + \mu_{\bigcup_{i=1}^{\ell_{\mathcal{B}}-1}\mathcal{I}_{b_i}}}\right) \cdot \left(\prod_{j=1}^{\ell_{\mathcal{A}}} \frac{\lambda_{a_j}}{\mu_{\bigcup_{i=1}^{j}\mathcal{I}_{a_i}}}\right) \cdot \left(\prod_{j=1}^{\ell_{\mathcal{B}}-1} \frac{\lambda_{b_j}}{\mu_{\bigcup_{i=1}^{j}\mathcal{I}_{b_i}}}\right) \\
&\quad + \cdots + \left(\frac{\lambda_{b_{\ell_{\mathcal{B}}}}}{\mu_{\bigcup_{i=1}^{\ell_{\mathcal{A}}+1}\mathcal{I}_{a_i}} + \mu_{\bigcup_{i=1}^{\ell_{\mathcal{B}}}\mathcal{I}_{b_i}}}\right) \cdots \left(\frac{\lambda_{b_1}}{\mu_{\bigcup_{i=1}^{\ell_{\mathcal{A}}+1}\mathcal{I}_{a_i}} + \mu_{\mathcal{I}_{b_1}}}\right) \cdot \left(\frac{\lambda_{a_{\ell_{\mathcal{A}}+1}}}{\mu_{\bigcup_{i=1}^{\ell_{\mathcal{A}}+1}\mathcal{I}_{a_i}}}\right) \cdot \left(\prod_{j=1}^{\ell_{\mathcal{A}}} \frac{\lambda_{a_j}}{\mu_{\bigcup_{i=1}^{j}\mathcal{I}_{a_i}}}\right)\Bigg] \\
&= \left(\frac{\lambda}{\mu}\right)^{n_R} \cdot \left(\frac{\lambda_R}{\mu}\right) \cdot \left(\prod_{j=1}^{\ell_{\mathcal{A}}+1} \frac{\lambda_{a_j}}{\mu_{\bigcup_{i=1}^{j}\mathcal{I}_{a_i}}}\right) \cdot \left(\prod_{j=1}^{\ell_{\mathcal{B}}} \frac{\lambda_{b_j}}{\mu_{\bigcup_{i=1}^{j}\mathcal{I}_{b_i}}}\right),
\end{aligned}
$$

where the first equality results from the inductive hypothesis and the form of the limiting probabilities of the detailed state space given in (1), and the second equality results from combining the summed terms using a common denominator, and then simplifying. We thus have exactly the form given in (3).

A similar argument follows for the case when $n_R = -$. □ □

B Proof of Lemma 4

Proof. We find the normalizing constant via induction on the number of job classes. Our base case is the $\mathbb{W}$ model, which has three classes and two servers (note that we can obtain systems with two or one classes by simply setting the arrival rates for unwanted classes to be 0). From [5], we know that the normalizing constant in the $\mathbb{W}$ model is

$$\mathcal{C}_{\mathbb{W}} = (1-\rho_A)(1-\rho_B)(1-\rho_R).$$

Now assume that the normalizing constant follows the form given in (4) for all systems with at most m job classes. Consider a system with $m+1$ job classes, and let class R be the most redundant class (i.e., class-R jobs replicate to all servers). By Lemma 3, we know that the limiting probability of being in state $\left(R, n_R, \substack{(\mathcal{A})\\(\mathcal{B})}\right)$ is

$$\pi_{\left(R,n_R,\substack{(\mathcal{A})\\(\mathcal{B})}\right)} = \left(\frac{\lambda}{\mu_{\mathcal{A}}+\mu_{\mathcal{B}}}\right)^{n_R}\left(\frac{\lambda_R}{\mu_{\mathcal{A}}+\mu_{\mathcal{B}}}\right)\cdot\pi_{\mathcal{A}}\cdot\pi_{\mathcal{B}},$$

up to the normalizing constant, where there are $i \leq m$ job classes in the left subsystem and $m-i$ job classes in the right subsystem.

Aggregating over all possible states for the left and the right subsystems, we find, for a constant ξ to be determined below,

$$\pi_{\left(R,n_R,\substack{(*)\\(*)}\right)} = \xi\left(\frac{\lambda}{\mu_{\mathcal{A}}+\mu_{\mathcal{B}}}\right)^{n_R}\left(\frac{\lambda_R}{\mu_{\mathcal{A}}+\mu_{\mathcal{B}}}\right),$$

where, letting $A_1,\ldots,A_i$ and $B_1,\ldots,B_{n-i}$ denote the job classes in subsystems $\mathcal{A}$ and $\mathcal{B}$ respectively,

$$\mathcal{C}_{\mathcal{A}} = \prod_{y=1}^{i}(1-\rho_{A_y})$$

and

$$\mathcal{C}_{\mathcal{B}} = \prod_{y=1}^{m-i}(1-\rho_{B_y})$$

follow from the inductive hypothesis, and $\mathcal{C} = \xi\cdot\mathcal{C}_{\mathcal{A}}\cdot\mathcal{C}_{\mathcal{B}}$.

Finally, we sum over all values of n, as well as the state in which there are no class-R jobs in the system; this sum must be equal to 1:

$$1 = \pi_{\left(R,-,\substack{(*)\\(*)}\right)} + \sum_{n=0}^{\infty}\pi_{\left(R,n_R,\substack{(*)\\(*)}\right)}$$
$$1 = \xi + \sum_{n=0}^{\infty}\xi\left(\frac{\lambda}{\mu_{\mathcal{A}}+\mu_{\mathcal{B}}}\right)^{n_R}\left(\frac{\lambda_R}{\mu_{\mathcal{A}}+\mu_{\mathcal{B}}}\right)$$

$$1 = \xi \left(1 + \left(\frac{\lambda_R}{\mu_{\mathcal{A}} + \mu_{\mathcal{B}}} \right) \left(\frac{1}{1 - \frac{\lambda}{\mu_{\mathcal{A}} + \mu_{\mathcal{B}}}} \right) \right)$$
$$1 = \xi \left(\frac{\mu_{\mathcal{A}} + \mu_{\mathcal{B}} - \lambda + \lambda_R}{\mu_{\mathcal{A}} + \mu_{\mathcal{B}} - \lambda} \right),$$

so we have

$$\begin{aligned} \xi = \pi_{\binom{(*)}{(*)}_{R,-,}} &= \frac{\mu_{\mathcal{A}} + \mu_{\mathcal{B}} - \lambda}{\mu_{\mathcal{A}} + \mu_{\mathcal{B}} - \lambda + \lambda_R} \\ &= 1 - \rho_R \end{aligned}$$

and hence

$$\begin{aligned} \mathcal{C} &= \xi \cdot \mathcal{C}_{\mathcal{A}} \cdot \mathcal{C}_{\mathcal{B}} \\ &= \prod_{i=1}^{\ell} (1 - \rho_i) \end{aligned}$$

as desired. □