Efficient Redundancy Techniques for Latency Reduction in Cloud Systems

GAURI JOSHI, Carnegie Mellon University
EMINA SOLJANIN, Rutgers University
GREGORY WORNELL, Massachusetts Institute of Technology

In cloud computing systems, assigning a task to multiple servers and waiting for the earliest copy to finish is an effective method to combat the variability in response time of individual servers and reduce latency. But adding redundancy may result in higher cost of computing resources, as well as an increase in queueing delay due to higher traffic load. This work helps in understanding when and how redundancy gives a cost-efficient reduction in latency. For a general task service time distribution, we compare different redundancy strategies in terms of the number of redundant tasks and the time when they are issued and canceled. We get the insight that the log-concavity of the task service time creates a dichotomy of when adding redundancy helps. If the service time distribution is log-convex (i.e., log of the tail probability is convex), then adding maximum redundancy reduces both latency and cost. And if it is log-concave (i.e., log of the tail probability is concave), then less redundancy, and early cancellation of redundant tasks is more effective. Using these insights, we design a general redundancy strategy that achieves a good latency-cost trade-off for an arbitrary service time distribution. This work also generalizes and extends some results in the analysis of fork-join queues.

Categories and Subject Descriptors: C.4 [**Performance of Systems**]: Modeling Techniques, Reliability, Availability, and Serviceability; G.3 [**Probability and Statistics**]: Queueing Theory

General Terms: Task Replication, Fork-Join Queues

Additional Key Words and Phrases: Performance modeling, latency-cost analysis

ACM Reference Format:
Gauri Joshi, Emina Soljanin, and Gregory Wornell. 2017. Efficient redundancy techniques for latency reduction in cloud systems. ACM Trans. Model. Perform. Eval. Comput. Syst. 2, 2, Article 12 (April 2017), 30 pages.
DOI: http://dx.doi.org/10.1145/3055281

1. INTRODUCTION

1.1. Motivation

An increasing number of applications are now hosted on the cloud. Some examples are streaming (Netflix, YouTube), storage (Dropbox, Google Drive), and computing (Amazon EC2, Microsoft Azure) services. A major advantage of cloud computing and storage is that the large-scale sharing of resources provides scalability and flexibility. However,

This work was supported in part by NSF under grant CCF-1319828, AFOSR under grant FA9550-11-1-0183, and a Schlumberger Faculty for the Future Fellowship.
Our work was presented in part at the 2015 Allerton Conference on Communication, Control, and Computing and the 2015 ACM Sigmetrics Mathematical Modeling and Analysis Workshop. G. Joshi was at MIT at the time of this work.
Authors' addresses: G. Joshi, Carnegie Mellon University, 5000 Forbes Avenue, Pittsburgh PA 15213; email: gaurij@andrew.cmu.edu; E. Soljanin, Rutgers University, 94 Brett Road, Piscataway NJ 08854; email: emina.soljanin@rutgers.edu; G. Wornell, Massachusetts Institute of Technology, 77 Massachusetts Ave, Cambridge MA 02139; email: gww@mit.edu.
Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies show this notice on the first page or initial screen of a display along with the full citation. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, to republish, to post on servers, to redistribute to lists, or to use any component of this work in other works requires prior specific permission and/or a fee. Permissions may be requested from Publications Dept., ACM, Inc., 2 Penn Plaza, Suite 701, New York, NY 10121-0701 USA, fax +1 (212) 869-0481, or permissions@acm.org.
© 2017 ACM 2376-3639/2017/04-ART12 $15.00
DOI: http://dx.doi.org/10.1145/3055281

Table I. Organization of Main Latency-Cost Analysis Results Presented in the Rest of the Article

	$k = 1$ (Replicated) Case	**General k**
Full Forking to All n Servers	Section 5 Comparison of strategies with and without early task cancellation	Section 7 Bounds on latency and cost, and the diversity-parallelism trade-off
Partial Forking to r Out of n Servers	Section 6 Effect of r and the choice of servers on latency and cost	Section 8 General redundancy strategy for cost-efficient latency reduction

Note: We fork each job into tasks at all n servers (full forking) or to some subset r out of n servers (partial forking). A job is complete when any k of its tasks are served.

an adverse effect of the sharing of resources is the variability in the latency experienced by the user due to queueing, virtualization, server outages, and so forth. The problem becomes further aggravated when the computing job has several parallel tasks, because the slowest task becomes the bottleneck in job completion. Thus, ensuring fast and seamless service is a challenging problem in cloud systems.

One method to reduce latency that has gained significant attention in recent years is the use of redundancy. In cloud computing, replicating a task on multiple machines and waiting for the earliest copy to finish can significantly reduce the latency [Dean and Barroso 2013]. Similarly, in cloud storage systems, requests to access a content can be assigned to multiple replicas such that it is only sufficient to download one replica. However, redundancy can result in increased use of resources, such as computing time, and network bandwidth. In frameworks such as Amazon EC2 and Microsoft Azure that offer computing as a service, the computing time spent on a job is proportional to the cost of renting the machines.

1.2. Organization of This Work

In this work, we aim to understand the trade-off between latency and computing cost, and propose efficient strategies to add redundancy. We focus on a redundancy model called the (n, k) *fork-join model*, where a job is forked into n tasks such that completion of any k tasks is sufficient to finish the job. In Section 2, we formally define this model and its variants. Section 3 summarizes related previous work and our contributions. Section 4 gives the key preliminary concepts used in this work.

The rest of the article studies different variants of the (n, k) fork-join model in increasing order of generality, as shown in Table I. In Sections 5 and 6, we focus on the $k = 1$ (replicated) case. Section 5 considers full replication of a job at all n servers and compares different strategies of canceling redundant tasks. In Section 6, we consider partial replication at r out of n servers.

In Sections 7 and 8, we move to the general k case, which requires a significantly different style of analysis than the $k = 1$ case. In Section 7, we consider full forking to all n servers and determine bounds on latency and cost, generalizing some of the fundamental work on fork-join queues. For partial forking, we propose a general redundancy strategy in Section 8. System designers looking for a practical redundancy strategy rather than theoretical analysis may skip ahead to Section 8 after the problem setup in Section 2.

Finally, Section 9 summarizes the results and provides future perspectives. Properties and examples of log-concavity are given in Appendix A. Proofs of the $k = 1$ and general k cases are deferred to Appendix B and Appendix C, respectively.

2. SYSTEM MODEL

2.1. Fork-Join Model and Its Variants

Definition 1 ((n, k) *Fork-Join System*). Consider a distributed system with n statistically identical servers. Jobs arrive to the system at rate λ, according to a Poisson

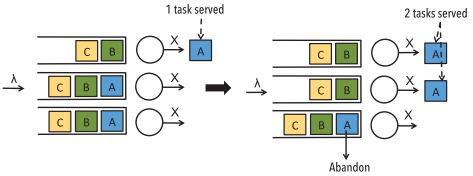


Fig. 1. The $(3, 2)$ fork-join system. When any two out of three tasks of a job are served (as seen for Job A on the right), the third task abandons its queue and the job exits the system.

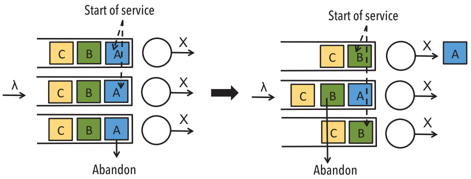


Fig. 2. The $(3, 2)$ fork-early-cancel system. When and two out of three tasks of a job are in service, the third task abandons (seen for Job A on the left and Job B on the right).

process.[1] Each job is forked into n tasks that join first-come first-served queues at each of the n servers. The job is said to be complete when any k tasks are served. At this instant, all remaining tasks are canceled and abandon their respective queues immediately.

After a task of the job reaches the head of its queue, the time taken to serve it can be random due to various factors, such as disk seek time and sharing of computing resources between multiple processes. We model this service time by a random variable $X > 0$, with cumulative distribution function (CDF) $F_X(x)$. The tail distribution (inverse CDF) of X is denoted by $\bar{F}_X(x) = \Pr(X > x)$. We use $X_{k:n}$ to denote the k^{th} smallest of n independent and identically distributed random variables $X_1, X_2, \ldots, X_n$.

We assume that the service time X is independent and identically distributed across tasks and servers. Thus, if a task is replicated at two different servers, the service times of the replicas are independent and identically distributed. Dependence of service time on the task itself can be modeled by adding a constant Δ to X. More generally, Δ may be a random variable. Although we do not consider this case here, the results in this article (particularly Section 5) can be extended to consider correlated service times.

Figure 1 illustrates the $(3, 2)$ fork-join system. The job exits the system when any two out of three tasks are complete. The $k = 1$ case corresponds to a replicated system where a job is sent to all n servers and we wait for one of the replicas to be served. The (n, k) fork-join system with $k > 1$ can serve as a useful model to study content access latency from an (n, k) erasure-coded distributed storage system. Approximate computing applications that require only a fraction of tasks of a job to be complete can also be modeled using the (n, k) fork-join system.

We consider the following two variants of this system, which could save the amount of redundant time spent by the servers of each job:

(1) (n, k) *fork-early-cancel system*: Instead of waiting for k tasks to finish, the redundant tasks are canceled when any k tasks reach the heads of their queues and start service. If more than k tasks start service simultaneously, we retain any k chosen uniformly at random. Figure 2 illustrates the $(3, 2)$ fork-early-cancel system. In Section 5, we compare the (n, k) systems with and without early cancellation.

(2) (n, r, k) *partial fork-join system:* Each incoming job is forked into $r > k$ out of the n servers. When any k tasks finish service, the redundant tasks are canceled immediately and the job exits the system. The r servers can be chosen according to different scheduling policies, such as random, round-robin, and least-work-left

[1] The Poisson assumption is required only for the exact analysis and bounds on latency (Equations (5), (7), (8), (11), (16), (17), and (24)). All other results on $\mathbb{E}[C]$ and comparison of replication strategies in heavy traffic hold for any arrival process.

(see Chapter 24 of Harchol-Balter [2013] for definitions). In Section 6, we develop insights into the best choice of r and the scheduling policy.

Other variants of the fork-join system include a combination of partial forking and early cancellation, or delaying invocation of some of the redundant tasks. Although not studied in detail here, our analysis techniques can be extended to these variants. In Section 8, we propose a general redundancy strategy that is a combination of partial forking and early cancellation.

2.2. Latency and Cost Metrics

We now define the metrics of the latency and resource cost whose trade-off is analyzed in the rest of the article.

Definition 2 (*Latency*). The latency T is defined as the time from the arrival of a job until it is served. In other words, it is the response time experienced by the job.

In this work, we focus on analyzing the expected latency $\mathbb{E}[T]$. Although $\mathbb{E}[T]$ is a good indicator of the average behavior, system designers are often interested in the tail $\Pr(T > t)$ of the latency. For many queueing problems, determining the distribution of response time T requires the assumption of exponential service time. To consider arbitrary, nonexponential service time distribution F_X, we settle for analyzing $\mathbb{E}[T]$ here.

Definition 3 (*Computing Cost*). The computing cost C is the total time spent by the servers serving a job, not including the time spent in the queue.

In computing-as-a-service frameworks, the computing cost is proportional to money spent on renting machines to run a job on the cloud.[2]

3. PREVIOUS WORK AND MAIN CONTRIBUTIONS

3.1. Related Previous Work

Systems work. The use of redundancy to reduce latency is not new. One of the earliest instances is the use of multiple routing paths [Maxemchuk 1975] to send packets in networks (see Chapter 7 of Kabatiansky et al. [2005] for a detailed survey of other related work). A similar idea has been studied [Vulimiri et al. 2013] in the context of DNS queries. In large-scale cloud computing frameworks, several recent works in systems [Dean and Ghemawat 2008; Ananthanarayanan et al. 2013; Ousterhout et al. 2013] explore straggler mitigation techniques where redundant replicas of straggling tasks are launched to reduce latency. Although the use of redundancy has been explored in systems literature, there is little work on the rigorous analysis of how it affects latency, particularly the cost of resources. Next we review some of that work.

Exponential service time. The (n, k) fork-join system was first proposed in Joshi et al. [2012, 2014] to analyze content download latency from erasure-coded distributed storage. These works consider that a content file coded into n chunks can be recovered by accessing any k out of the n chunks, where the service time X of each chunk is exponential. Even with the exponential assumption, analyzing the (n, k) fork-join system is a hard problem. It is a generalization of the (n, n) fork-join system, which was actively

[2] Although we focus on this cost metric, we note that redundancy also results in a network cost of making remote-procedure calls (RPCs) made to assign tasks of a job and cancel redundant tasks. It is proportional to the number of servers to which each job is forked, which is n for the (n, k) fork-join model described earlier. In the context of distributed storage, redundancy also results in increased use of storage space, proportional to n/k. The trade-off between delay and storage is studied in Joshi et al. [2012, 2014].

Table II. Latency-Optimal and Cost-Optimal Redundancy Strategies for the $k = 1$ (Replicated) Case

	Log-Concave Service Time		**Log-Convex Service Time**	
	Latency Optimal	**Cost Optimal**	**Latency Optimal**	**Cost Optimal**
Cancel Redundancy Early or Keep It?	*Low load*: Keep redundancy, *High load*: Cancel early	Cancel early	Keep redundancy	Keep redundancy
Partial Forking to *r* Out of *n* Servers	*Low load*: $r = n$ (fork to all), H*igh load*: $r = 1$ (fork to one)	$r = 1$	$r = n$ (fork to all)	$r = n$ (fork to all)

Note: *Canceling redundancy early* means that instead of waiting for any one task to finish, we cancel redundant tasks as soon as any one task begins service.

studied in queueing literature [Flatto and Hahn 1984; Nelson and Tantawi 1988; Varki et al. 2008] about two decades ago.

An analysis of latency with heterogeneous job classes for the replicated ($k = 1$) case with distributed queues was presented in Gardner et al. [2015]. Other related works include Shah et al. [2014], Kumar et al. [2014], Xiang et al. [2014], and Kadhe et al. [2015]. A common thread in all of these works is that they also assume exponential service time.

General service time. Few practical systems have exponentially distributed service time. For example, studies of download time traces from Amazon S3 [Liang and Kozat 2014; Chen et al. 2014] indicate that the service time is not exponential in practice but instead is a shifted exponential. For service time distributions that are "new-worse-than-used" (NWU) [Cao and Wang 1991], it is shown in Koole and Righter [2008] that it is optimal to replicate a job at all servers in the system. The choice of scheduling policy for NWU and "new-better-than-used" (NBU) distributions is studied in Kim et al. [2009], Shah et al. [2013], and Sun et al. [2015]. The NBU and NWU notions are closely related to the log-concavity of service time studied in this work.

The cost of redundancy. If we assume exponential service time, then redundancy does not cause any increase in cost of server time. But since this is not true in practice, it is important to determine the cost of using redundancy. Simulation results with nonzero fixed cost of removal of redundant requests are presented in Shah et al. [2013]. The expected computing cost $\mathbb{E}[C]$ spent per job was previously considered in Wang et al. [2014, 2015] for a distributed system without considering queueing of requests. In Joshi et al. [2015], we presented an analysis of the latency and cost of the (n, k) fork-join with and without early cancellation of redundant tasks.

3.2. Main Contributions

The main differences between this and previous works are the following: (1) we consider a general service time distribution instead of exponential service time, and (2) we analyze the impact of redundancy on the latency, as well as the computing cost (total server time spent per job). Incidentally, our computing cost metric $\mathbb{E}[C]$ also serves as a powerful tool to compare different redundancy strategies under high load.

The latency-cost analysis of the fork-join system and its variants gives us the insight that the log-concavity (and respectively the log-convexity) of $\bar{F}_X$, the tail distribution of service time, is a key factor in choosing the redundancy strategy. Here are some examples, which are also summarized in Table II:

—By comparing the $(n, 1)$ systems (fork to n, wait for any 1) with and without early cancellation, we can show that early cancellation of redundancy can reduce both latency and cost for log-concave $\bar{F}_X$, but it is not effective for log-convex $\bar{F}_X$.

—For the $(n, r, 1)$ partial-fork-join system (fork to r out of n, wait for any 1), we can show that forking to more servers (larger r) is both latency and cost optimal for log-convex $\bar{F}_X$. But for log-concave $\bar{F}_X$, larger r reduces latency only in the low traffic regime and always increases the computing cost.

Using these insights, we also develop a general redundancy strategy to decide how many servers to fork to, and when to cancel the redundant tasks, for an arbitrary service time that may be neither log-concave nor log-convex.

4. PRELIMINARY CONCEPTS

We now present some preliminary concepts that are vital to understanding the results presented in the rest of the article.

4.1. Using $\mathbb{E}[C]$ to Compare Systems

Since the cost metric $\mathbb{E}[C]$ is the expected time spent by servers on each job, higher $\mathbb{E}[C]$ implies a higher expected waiting time for subsequent jobs. Thus, $\mathbb{E}[C]$ can be used to compare the latency to different redundancy policies in the heavy traffic regime. In particular, we compare policies that are symmetric across the servers, defined formally as follows.

Definition 4 (*Symmetric Policy*). With a symmetric scheduling policy, the tasks of each job are forked to one or more servers such that the expected task arrival rate is equal across all servers.

Most commonly used policies, such as random, round-robin, and join the shortest queue (JSQ), are symmetric across the n servers. In Lemma 1, we express the stability region of the system in terms of $\mathbb{E}[C]$.

Lemma 1 (Stability Region in Terms of $\mathbb{E}[C]$). *A system of n servers with a symmetric redundancy policy is stable—that is, the mean response time $E[T] < \infty$, only if the arrival rate λ (with any arrival process) satisfies*

$$\lambda < \frac{n}{\mathbb{E}[C]}. \tag{1}$$

Thus, the maximum arrival rate that can be supported is $\lambda_{max} = n/\mathbb{E}[C]$, where $\mathbb{E}[C]$ depends on the redundancy scheduling policy.

Proof of Lemma 1. For a symmetric policy, the mean time spent by each server per job is $\mathbb{E}[C]/n$. Thus, the server utilization is $\rho = \lambda\mathbb{E}[C]/n$. By the server utilization version of Little's law, ρ must be less than 1 for the system to be stable. The result follows from this. □

Definition 5 (*Service Capacity λ^*_{max}*). The service capacity of the system λ^*_{max} is the maximum achievable λ_{max} over all symmetric policies.

From Lemma 1 and Definition 5, we can infer Corollary 1.

Corollary 1. *The redundancy strategy that minimizes $\mathbb{E}[C]$ results in the lowest $\mathbb{E}[T]$ in the heavy traffic regime ($\lambda \to \lambda^*_{max}$).*

Note that as λ approaches λ^*_{max}, the expected latency $\mathbb{E}[T] \to \infty$ for all strategies whose $\lambda_{max} < \lambda^*_{max}$.

4.2. Log-Concavity of $\bar{F}_X$

If we fork a job to all r idle servers and wait for any one copy to finish, the expected computing cost $\mathbb{E}[C] = r\mathbb{E}[X_{1:r}]$, where $X_{1:r} = \min(X_1, X_2, \ldots, X_r)$, the minimum of

r independent and identically distributed realizations of random variable X. The behavior of this cost function depends on whether the tail distribution $\bar{F}_X$ of service time is log-concave or log-convex. Log-concavity of $\bar{F}_X$ is defined formally as follows.

Definition 6 (*Log-Concavity and Log-Convexity of* $\bar{F}_X$). The tail distribution $\bar{F}_X$ is said to be log-concave (log-convex) if $\log \Pr(X > x)$ is concave (convex) in x for all $x \in [0, \infty)$.

For brevity, when we say that X is log-concave (log-convex) in this article, we mean that $\bar{F}_X$ is log-concave (log-convex). Lemma 2 states how $r\mathbb{E}[X_{1:r}]$ varies with r for log-concave (log-convex) $\bar{F}_X$.

Lemma 2 (Expected Minimum). *If X is log-concave (log-convex), then $r\mathbb{E}[X_{1:r}]$ is nondecreasing (nonincreasing) in r.*

The proof of Lemma 2 can be found in Appendix A. Note that the exponential distribution is both log-concave and log-convex, and thus $r\mathbb{E}[X_{1:r}]$ remains constant as r varies. This can also be seen from the fact that when $X \sim Exp(\mu)$, an exponential with rate μ, $X_{1:r}$ is an exponential with rate $r\mu$. Then $r\mathbb{E}[X_{1:r}] = 1/\mu$, a constant independent of r.

Log-concave and log-convex distributions have been studied in economics and reliability theory and have many interesting properties. Properties relevant to this work are given in Appendix A. We refer readers to Bagnoli and Bergstrom [2005]. In Remark 1, we highlight one key property that provides intuitive understanding of log-concavity.

Remark 1. It is well known that the exponential distribution is memoryless. Log-concave distributions have "optimistic memory"—that is, the expected remaining service time of a task decreases with the time elapsed. On the other hand, log-convex distributions have "pessimistic memory."

Distributions with optimistic memory are referred to as NBU [Koole and Righter 2008], light-everywhere [Shah et al. 2013], or new-longer-than-used [Sun et al. 2015]. Log-concavity of X implies that X is NBU (see Property 3 in Appendix A for the proof).

A natural question is this: what are examples of log-concave and log-convex distributions that arise in practice? A canonical example of a log-concave distribution is the shifted exponential distribution $ShiftedExp(\Delta, \mu)$, which is exponential with rate μ, plus a constant shift $\Delta \geq 0$, is log-concave. Recent work [Liang and Kozat 2014; Chen et al. 2014] on analysis of content download from Amazon S3 observed that X is shifted exponential, where Δ is proportional to the size of the content and the exponential part is the random delay in starting the data transfer. Another example of log-concave service time is the uniform distribution over any convex set.

Log-convex service times occur when there is high variability in service time. CPU service times are often approximated by the hyperexponential distribution, which is a mixture of two or more exponentials. In this article, we focus on mixtures of two exponentials with decay rates μ_1 and μ_2, respectively, where the exponential with rate μ_1 occurs with probability p. We denote this distribution by $HyperExp(\mu_1, \mu_2, p)$. If a server is generally fast (rate μ_1) but it can slow down (rate $\mu_2 < \mu_1$) with probability $1 - p$, then the overall service time distribution would be $X \sim HyperExp(\mu_1, \mu_2, p)$.

Many practical systems also have service times that are neither log-concave nor log-convex. In this work, we use the Pareto distribution $Pareto(x_m, \alpha)$ as an example of such distributions. Its tail distribution is given by

$$\Pr(X > x) = \begin{cases} \left(\frac{x_m}{x}\right)^{\alpha} & x \geq x_m, \\ 1 & \text{otherwise.} \end{cases} \tag{2}$$

The tail distribution in (2) is log-convex for $x \geq x_m$, but not for all $x \geq 0$ due the initial delay of x_m. Thus, overall, the Pareto distribution is neither log-concave nor log-convex.

Remark 2. Log-concave (log-convex) distributions are reminiscent of another well-known class of distributions: light (heavy) tailed distributions. Many random variables with log-concave (log-convex) $\bar{F}_X$ are light (heavy) tailed, respectively, but neither property implies the other. For example, the Pareto distribution defined earlier is heavy tailed but is neither log-concave nor log-convex. Whereas the tail of a distribution characterizes how the maximum $\mathbb{E}[X_{n:n}]$ behaves for large n, log-concavity (log-convexity) of $\bar{F}_X$ characterizes the behavior of the minimum $\mathbb{E}[X_{1:n}]$, which is of primary interest in this work.

4.3. Relative Task Start Times

Since the tasks of the job experience different waiting times in their respective queues, they start being served at different times. The relative start times of the n tasks of a job is an important factor affecting the latency and cost. We denote the relative start times by $t_1 \leq t_2 \leq \cdots \leq t_n$, where $t_1 = 0$ without loss of generality. For instance, if $n = 3$ tasks start at absolute times 3, 4, and 7, then their relative start times are $t_1 = 0$, $t_2 = 4 - 3 = 1$, and $t_3 = 7 - 3 = 4$. In the case of partial forking when only r tasks are invoked, we can consider $t_{r+1}, \cdots t_n$ to be ∞.

For the replicated case ($k = 1$), let S be the time from when the earliest replica of a task starts service until any one replica finishes. It is the minimum of $X_1 + t_1, X_2 + t_2, \ldots, X_n + t_n$, where X_i are independent and identically distributed with distribution F_X. The tail distribution $\Pr(S > s)$ is given by

$$\Pr(S > s) = \prod_{i=1}^{n} \Pr(X > s - t_n). \tag{3}$$

The computing cost C can be expressed in terms of S and t_i as follows:

$$C = S + (S - t_2)^+ + \cdots + (S - t_n)^+ . \tag{4}$$

Using (4), we get several crucial insights in the rest of the article. For instance, in Section 6, we show that when $\bar{F}_X$ is log-convex, having $t_1 = t_2 = \cdots = t_n = 0$ gives the lowest $\mathbb{E}[C]$. Then using Lemma 1, we can infer that it is optimal to fork a job to all n servers when $\bar{F}_X$ is log-convex.

5. *k* = 1 CASE WITHOUT AND WITH EARLY CANCELLATION

In this section, we analyze the latency and cost of the $(n, 1)$ fork-join system, and the $(n, 1)$ fork-early-cancel system defined in Section 2. We get the insight that it is better to cancel redundant tasks early if $\bar{F}_X$ is log-concave. However, if $\bar{F}_X$ is log-convex, retaining the redundant tasks is better.

5.1. Latency-Cost Analysis

Theorem 1. *The expected latency and computing cost of an* $(n, 1)$ *fork-join system are given by*

$$\mathbb{E}[T] = \mathbb{E}[T^{M/G/1}] = \mathbb{E}[X_{1:n}] + \frac{\lambda \mathbb{E}[X_{1:n}^2]}{2(1 - \lambda \mathbb{E}[X_{1:n}])}, \tag{5}$$

$$\mathbb{E}[C] = n \cdot \mathbb{E}[X_{1:n}], \tag{6}$$

where $X_{1:n} = \min(X_1, X_2, \ldots, X_n)$ *for independent and identically distributed* $X_i \sim F_X$.

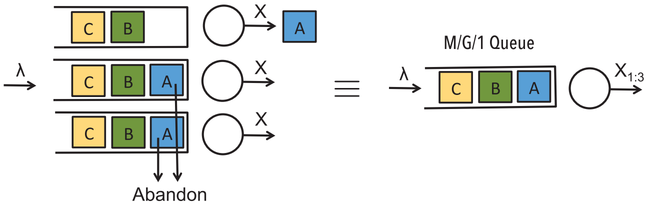


Fig. 3. Equivalence of the $(n, 1)$ fork-join system with an $M/G/1$ queue with service time $X_{1:n}$, the minimum of n independent and identically distributed random variables $X_1, X_2, \ldots, X_n$.

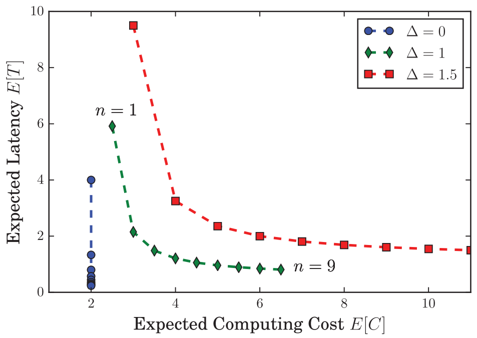


Fig. 4. The service time $X \sim \Delta + \text{Exp}(\mu)$ (log-concave), with $\mu = 0.5, \lambda = 0.25$. As n increases along each curve, $\mathbb{E}[T]$ decreases and $\mathbb{E}[C]$ increases. Only when $\Delta = 0$ does latency reduce at no additional cost.

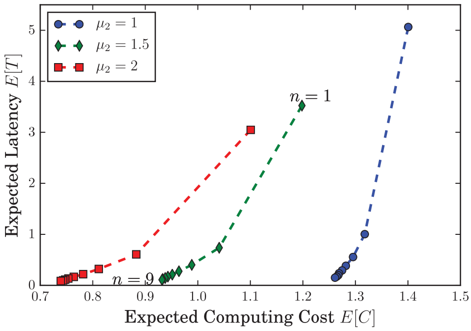


Fig. 5. The service time $X \sim \mathit{HyperExp}(0.4, \mu_1, \mu_2)$ (log-convex), with $\mu_1 = 0.5$, different values of μ_2, and $\lambda = 0.5$. Expected latency and cost both reduce as n increases along each curve.

Proof. Consider the first job that arrives to a $(n, 1)$ fork-join system when all servers are idle. The n tasks of this job start service simultaneously at their respective servers. The earliest task finishes after time $X_{1:n}$, and all other tasks are canceled immediately. Thus, the tasks of all subsequent jobs arriving to the system also start simultaneously at the n servers as illustrated in Figure 3. Hence, arrival and departure events, and the latency of an $(n, 1)$ fork-join system, are equivalent in distribution to an $M/G/1$ queue with service time $X_{1:n}$.

The expected latency of an $M/G/1$ queue is given by the Pollaczek-Khinchine formula (5). The expected cost $\mathbb{E}[C] = n\mathbb{E}[X_{1:n}]$ because each of the n servers spends $X_{1:n}$ time on the job. This can also be seen by noting that $S = X_{1:n}$ when $t_i = 0$ for all i, and thus by (4), $C = nX_{1:n}$. □

From (5), it is easy to see that for any service time distribution F_X, the expected latency $\mathbb{E}[T]$ is nonincreasing with n. The behavior of $\mathbb{E}[C]$ follows from Lemma 2 as given by Corollary 2.

Corollary 2. *If $\bar{F}_X$ is log-concave (log-convex), then $\mathbb{E}[C]$ is nondecreasing (nonincreasing) in n.*

Figures 4 and 5 show analytical plots of the expected latency versus cost for log-concave and log-convex $\bar{F}_X$, respectively. In Figure 4, the arrival rate $\lambda = 0.25$, and X is shifted exponential $\mathit{ShiftedExp}(\Delta, 0.5)$, with different values of Δ. For $\Delta > 0$, there is a trade-off between expected latency and cost. Only when $\Delta = 0$—that is, X is a pure exponential (which is generally not true in practice)—can we reduce latency without any additional cost. In Figure 5, arrival rate $\lambda = 0.5$, and X is hyperexponential

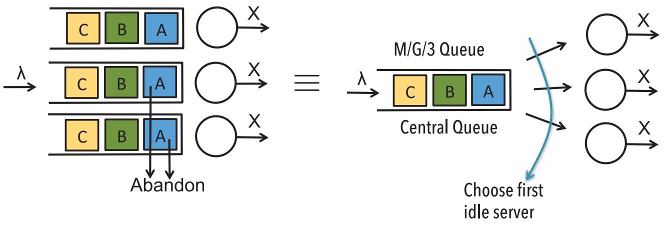


Fig. 6. Equivalence of the $(n, 1)$ fork-early-cancel system to an $M/G/n$ queue with each server taking time $X \sim F_X$ to serve the task independent and identically distributed across servers and tasks.

HyperExp$(0.4, 0.5, \mu_2)$, with different values of μ_2. We get a simultaneous reduction in $\mathbb{E}[T]$ and $\mathbb{E}[C]$ as n increases. The cost reduction is steeper as μ_2 increases.

Instead of holding the arrival rate λ constant, if we consider that it scales linearly with n, then the latency $\mathbb{E}[T]$ may not always decrease with n. In Corollary 3, we study the behavior as n varies.

COROLLARY 3. *If the arrival rate $\lambda = \lambda_0 n$, scaling linearly with n, then the latency $\mathbb{E}[T]$ decreases with n if $\bar{F}_X$ is log-convex. If $\bar{F}_X$ is log-concave, then $\mathbb{E}[T]$ increase with n in heavy traffic.*

PROOF. If $\lambda = \lambda_0 n$, then latency $\mathbb{E}[T]$ in (5) can be rewritten as

$$\mathbb{E}[T] = \mathbb{E}[X_{1:n}] + \frac{\lambda_0 n \mathbb{E}\left[X_{1:n}^2\right]}{2(1 - \lambda_0 n \mathbb{E}[X_{1:n}])}. \tag{7}$$

If $\bar{F}_X$ is log-convex, then by Lemma 2 we know that $n\mathbb{E}[X_{1:n}]$ decreases with n. Similarly, $n\mathbb{E}[X_{1:n}^2]$ also decreases with n (the proof follows similarly as Lemma 2. Hence, we can conclude that the latency in (7) decreases with n for log-convex $\bar{F}_X$. However, if $\bar{F}_X$ is log-concave, then $n\mathbb{E}[X_{1:n}]$ and $n\mathbb{E}[X_{1:n}^2]$ increase with n. Thus, in the heavy traffic regime ($\lambda \rightarrow \lambda^*_{max}$), when the second term in (7) dominates, $\mathbb{E}[T]$ increases with n. □

5.2. Early Task Cancellation

We now analyze the $(n, 1)$ fork-early-cancel system, where we cancel redundant tasks as soon as any task reaches the head of its queue. Intuitively, early cancellation can save computing cost, but the latency could increase due to the loss of diversity advantage provided by retaining redundant tasks. Comparing it to the $(n, 1)$ fork-join system, we gain the insight that early cancellation is better when $\bar{F}_X$ is log-concave but ineffective for log-convex $\bar{F}_X$.

THEOREM 2. *The expected latency and cost of the* $(n, 1)$ *fork-early-cancel system are given by*

$$\mathbb{E}[T] = \mathbb{E}[T^{M/G/n}], \tag{8}$$

$$\mathbb{E}[C] = \mathbb{E}[X], \tag{9}$$

where $T^{M/G/n}$ is the response time of an $M/G/n$ queueing system with service time $X \sim F_X$.

PROOF. In the $(n, 1)$ fork-early-cancel system, when any one task reaches the head of its queue, all others are canceled immediately. The redundant tasks help find the queue with the least work left, and exactly one task of each job is served by the first server that becomes idle. Thus, as illustrated in Figure 6, the latency of the $(n, 1)$ fork-early-cancel

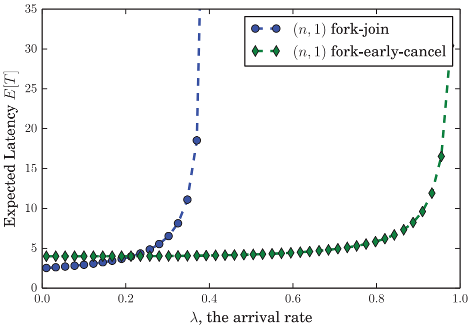


Fig. 7. For the (4, 1) system with service time $X \sim$ *ShiftedExp*$(2, 0.5)$, which is log-concave, early cancellation is better in the high λ regime, as given by Corollary 5.

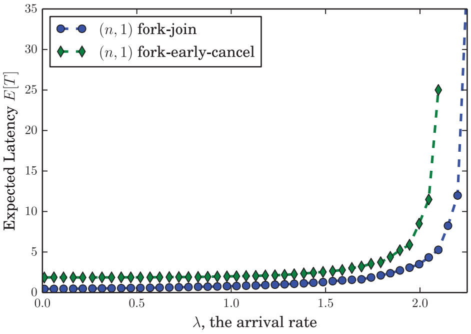


Fig. 8. For the (4, 1) system with $X \sim$ *HyperExp* $(0.1, 1.5, 0.5)$, which is log-convex, early cancellation is worse in both low and high λ regimes, as given by Corollary 5.

system is equivalent in distribution to an $M/G/n$ queue. Hence, $\mathbb{E}[T] = \mathbb{E}[T^{M/G/n}]$ and $\mathbb{E}[C] = \mathbb{E}[X]$. □

The exact analysis of mean response time $\mathbb{E}[T^{M/G/n}]$ has long been an open problem in queueing theory. A well-known approximation given by Lee and Longton [1959] is

$$\mathbb{E}[T^{M/G/n}] \approx \mathbb{E}[X] + \frac{\mathbb{E}[X^2]}{2\mathbb{E}[X]^2}\mathbb{E}[W^{M/M/n}], \tag{10}$$

where $\mathbb{E}[W^{M/M/n}]$ is the expected waiting time in an $M/M/n$ queueing system with load $\rho = \lambda\mathbb{E}[X]/n$. This expected waiting time can be evaluated using the Erlang-C model (see Chapter 14 of Harchol-Balter [2013]). A related work that studies the centralized queue model that the $(n, 1)$ fork-early-cancel system is equivalent to is Visschers et al. [2012], which considers the case of heterogeneous job classes with exponential service times.

Next we compare the latency and cost with and without early cancellation given by Theorems 1 and 2. Corollary 4 follows from Lemma 2.

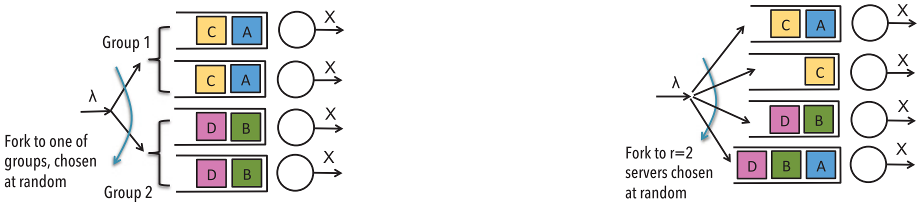
COROLLARY 4. *If $\bar{F}_X$ is log-concave (log-convex), then $\mathbb{E}[C]$ of the $(n, 1)$ fork-early-cancel system is greater than or equal to (less than or equal to) that of the $(n, 1)$ fork-join system.*

In the low λ regime, the $(n, 1)$ fork-join system gives lower $\mathbb{E}[T]$ than $(n, 1)$ fork-early-cancel because of higher diversity due to redundant tasks. By Corollary 1, in the high λ regime, the system with lower $\mathbb{E}[C]$ has lower expected latency.

COROLLARY 5. *If $\bar{F}_X$ is log-concave, then early cancellation gives higher $\mathbb{E}[T]$ than $(n, 1)$ fork-join when λ is small and lower in the high λ regime. If $\bar{F}_X$ is log-convex, then early cancellation gives higher $\mathbb{E}[T]$ for both low and high λ.*

Figures 7 and 8 illustrate Corollary 5. Figure 7 shows a comparison of $\mathbb{E}[T]$ with and without early cancellation of redundant tasks for the (4, 1) system with service time $X \sim$ *ShiftedExp*$(2, 0.5)$. We observe that early cancellation gives lower $\mathbb{E}[T]$ in the high λ regime. In Figure 8, we observe that when X is *HyperExp*$(0.1, 1.5, 0.5)$, which is log-convex, early cancellation is worse for both small and large λ.

In general, early cancellation is better when X is less variable (lower coefficient of variation). For example, in a comparison of $\mathbb{E}[T]$ with $(n, 1)$ fork-join and $(n, 1)$ fork-early-cancel systems as Δ, the constant shift of service time *ShiftedExp*(Δ, μ) varies, indicating that early cancellation is better for larger Δ. When Δ is small, there is


Group 1
Group 2
λ
Fork to one of groups, chosen at random
C
A
D
B
X
Fork to r=2 servers chosen at random

(a) Group-based random

(b) Uniform random

Fig. 9. A $(4, 2, 1)$ partial-fork-join system, where each job is forked to $r = 2$ servers, chosen according to the group-based random or uniform random policies.

more randomness in the service time of a task, and hence keeping the redundant tasks running gives more diversity and lower $\mathbb{E}[T]$. But as Δ increases, task service times are more deterministic, due to which it is better to cancel the redundant tasks early.

6. PARTIAL FORKING ($k = 1$ CASE)

For applications with a large number of servers n, full forking of jobs to all servers can be expensive in terms of the network cost of issuing and canceling the tasks. In this section, we analyze the $k = 1$ case of the (n, r, k) fork-join system, where an incoming job is forked to some r out n servers and we wait for any one task to finish. The r servers are chosen using a symmetric policy (Definition 4). The following are some examples of symmetric policies:

(1) *Group-based random*: This policy holds when r divides n. The n servers are divided into n/r groups of r servers each. A job is forked to one of these groups, chosen uniformly at random.
(2) *Uniform random:* A job is forked to any r out of n servers, chosen uniformly at random.

Figure 9 illustrates the $(4, 2, 1)$ partial-fork-join system with the group-based random and the uniform random policies. In the sequel, we develop insights into the best r and the choice of servers for a given service time distribution F_X.

Remark 3 (*Relation to Power-of-r Scheduling*). Power-of-r scheduling [Mitzenmacher 1996] is a well-known policy in multiserver systems. It chooses r out of the n servers at random and assigns an incoming task to the shortest queue among them. A major advantage of the power-of-r policy is that even with $r << n$, the latency achieved by it is close to the JSQ policy (equivalent to power-of-r with $r = n$).

The $(n, r, 1)$ partial-fork-join system with uniform random policy also chooses r queues at random. However, instead of choosing the shortest queue, it creates replicas of the task at all queues. The replicas help find the queue with the least work left, which gives better load balancing than joining the shortest queue. But unlike power-of-r, servers might spend redundant time on replicas that will eventually be canceled.

6.1. Latency-Cost Analysis

In the group-based random policy, the job arrivals are split equally across the groups, and each group behaves like an independent $(r, 1)$ fork-join system. Thus, the expected latency and cost follow from Theorem 1 as given in Lemma 3.

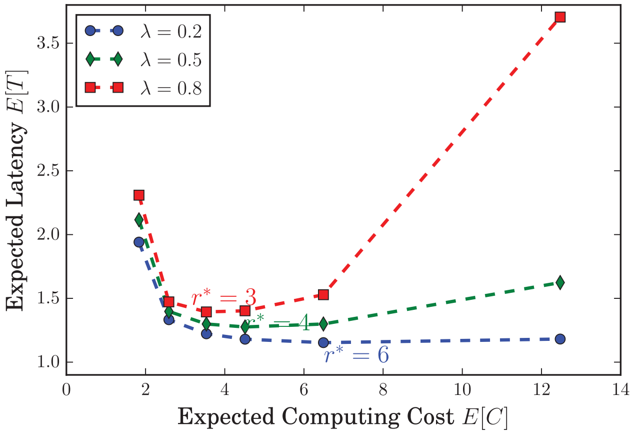


Fig. 10. Analytical plot of latency versus cost for $n = 12$ servers. Each job is replicated at r servers chosen by the group-based random policy, with r increasing as 1, 2, 3, 4, 6, and 12 along each curve. The task service time $X \sim Pareto(1, 2.2)$. As λ increases, the replicas increase queueing delay. Thus, the optimal r^* that minimizes $\mathbb{E}[T]$ shifts downward as λ increases.

Lemma 3 (Group-Based Random). *The expected latency and cost when each job is forked to one of n/r groups of r servers each are given by*

$$\mathbb{E}[T] = \mathbb{E}[X_{1:r}] + \frac{\lambda r \mathbb{E}[X_{1:r}^2]}{2(n - \lambda r \mathbb{E}[X_{1:r}])}, \tag{11}$$

$$\mathbb{E}[C] = r\mathbb{E}[X_{1:r}]. \tag{12}$$

Proof. The job arrivals are split equally across the n/r groups such that the arrival rate to each group is a Poisson process with rate $\lambda r/n$. The r tasks of each job start service at their respective servers simultaneously, and thus each group behaves like an independent $(r, 1)$ fork-join system with Poisson arrivals at rate $\lambda r/n$. Hence, the expected latency and cost follow from Theorem 1. □

Using (12) and Lemma 1, we can infer that the service capacity (maximum supported λ) for an $(n, r, 1)$ system with group-based random policy is

$$\lambda_{max} = \frac{n}{r\mathbb{E}\left[X_{1:r}\right]}. \tag{13}$$

From (13), we can infer that the r that minimizes $r\mathbb{E}[X_{1:r}]$ results in the highest service capacity, and hence the lowest $\mathbb{E}[T]$ in the heavy traffic regime. By Lemma 2, the optimal r is $r = 1$ ($r = n$) for log-concave (log-convex) $\bar{F}_X$. For distributions that are neither log-concave nor log-convex, an intermediate r may be optimal and we can determine it using Lemma 3. For example, Figure 10 shows a plot of latency versus cost as given by Lemma 3 for $n = 12$ servers. The task service time $X \sim Pareto(1, 2.2)$. Each job is replicated at r servers according to the group-based random policy, with r varying along each curve. Initially increasing r reduces the latency, but beyond r^*, the replicas cause an increase in the queueing delay. This increase in queueing delay is more dominant for higher λ. Thus, the optimal r^* decreases as λ increases.

For other symmetric policies, it is difficult to get an exact analysis of $\mathbb{E}[T]$ and $\mathbb{E}[C]$ because the tasks of a job can start at different times. However, we can get bounds on $\mathbb{E}[C]$ depending on the log-concavity of X, given in Theorem 3.

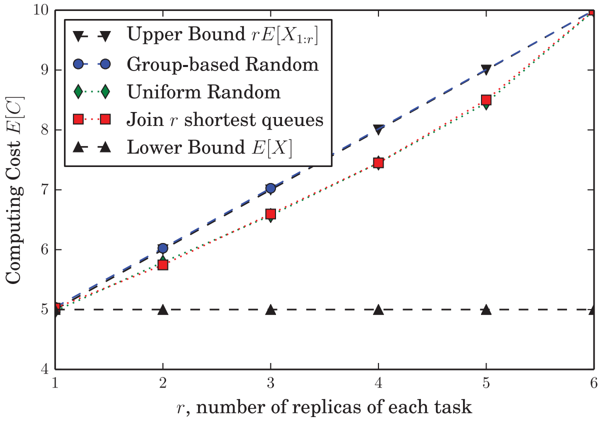


Fig. 11. Expected cost $\mathbb{E}[C]$ versus r for $X \sim \mathit{ShiftedExp}(1, 0.25)$, $n = 6$ servers, arrival rate $\lambda = 0.5$, and different scheduling policies. The upper bound $r\mathbb{E}[X_{1:r}]$ is exact for the group-based random policy and fairly tight for other policies.

Theorem 3. *Consider an $(n, r, 1)$ partial-fork join system, where a job is forked into tasks at r out of n servers chosen according to a symmetric policy. For any relative task start times t_i, $\mathbb{E}[C]$ can be bounded as follows:*

$$r\mathbb{E}[X_{1:r}] \geq \mathbb{E}[C] \geq \mathbb{E}[X] \qquad \text{if } \bar{F}_X \text{ is log-concave,} \tag{14}$$

$$\mathbb{E}[X] \geq \mathbb{E}[C] \geq r\mathbb{E}[X_{1:r}] \quad \text{if } \bar{F}_X \text{ is log-convex.} \tag{15}$$

In the extreme case when $r = 1$, $\mathbb{E}[C] = \mathbb{E}[X]$, and when $r = n$, $\mathbb{E}[C] = n\mathbb{E}[X_{1:n}]$.

To prove Theorem 3, we take expectation in (4) and show that for log-concave and log-convex $\bar{F}_X$, we get the bounds in (14) and (15), which are independent of the relative task start times t_i. The detailed proof is given in Appendix B.

In Figure 11, we show the bounds given by (14) for log-concave distributions alongside simulation values for different scheduling policies. The service time $X \sim \mathit{ShiftedExp}(1, 0.25)$, and arrival rate $\lambda = 0.5$. Since all replicas start simultaneously with the group-based random policy, the upper bound $\mathbb{E}[C] \geq r\mathbb{E}[X_{1:r}]$ is tight for any r. For other scheduling policies, the bound is more loose for the policy that staggers relative start times of replicas to a greater extent.

6.2. Optimal Value of *r*

We can use the bounds in Theorem 3 to gain insights into choosing the best r when $\bar{F}_X$ is log-concave or log-convex. In particular, we study two extreme traffic regimes: low traffic ($\lambda \to 0$) and heavy traffic ($\lambda \to \lambda^*_{max}$), where λ^*_{max} is the service capacity of the system introduced in Definition 5.

Corollary 6 (Expected Cost vs. r). *For a system of n servers with symmetric forking of each job to r servers, $r = 1$ $(r = n)$ minimizes the expected cost $\mathbb{E}[C]$ when $\bar{F}_X$ is log-concave (log-convex).*

The proof follows from Lemma 2 in that $r\mathbb{E}[X_{1:r}]$ is nondecreasing (nonincreasing) with r for log-concave (log-convex) $\bar{F}_X$.

Lemma 4 (Expected Latency vs. r). *In the low traffic regime, forking to all servers $(r = n)$ gives the lowest $\mathbb{E}[T]$ for any service time distribution F_X. In the heavy traffic regime, $r = 1$ $(r = n)$ gives the lowest $\mathbb{E}[T]$ if $\bar{F}_X$ is log-concave (log-convex).*

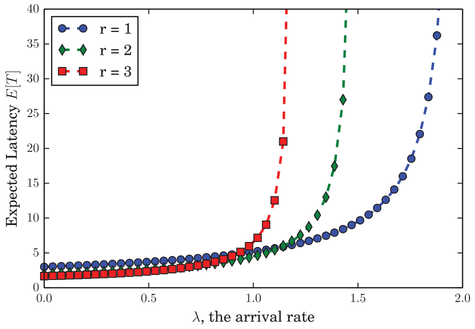


Fig. 12. For $X \sim ShiftedExp(1, 0.5)$, which is log-concave, forking to less (more) servers reduces expected latency in the low (high) λ regime. Each job is replicated at r out of $n = 6$ servers, chosen by the group-based random policy.

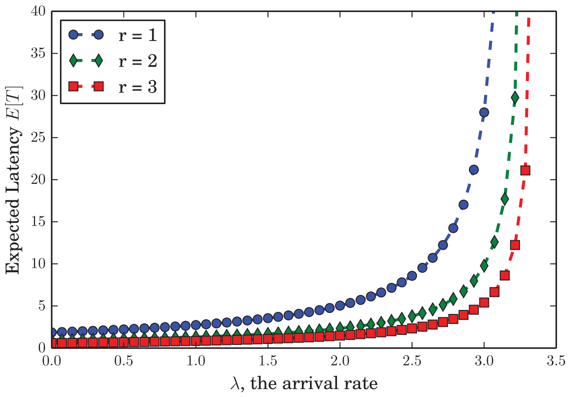


Fig. 13. For $X \sim HyperExp(p, \mu_1, \mu_2)$ with $p = 0.1$, $\mu_1 = 1.5$, and $\mu_2 = 0.5$, which is log-convex, larger r gives lower expected latency for all λ. Each job is replicated at r out of $n = 6$ servers, chosen according to the group-based random policy.

Proof. In the low traffic regime with $\lambda \to 0$, the waiting time in the queue tends to zero. Thus, all replicas of a task start service at the same time, irrespective of the scheduling policy. Then the expected latency is $\mathbb{E}[T] = \mathbb{E}[X_{1:r}]$, which decreases with r. Thus, $r = n$ gives the lower $\mathbb{E}[T]$ for any service time distribution F_X.

By Corollary 1, the optimal replication strategy in heavy traffic is the one that minimizes $\mathbb{E}[C]$. For log-convex $\bar{F}_X$, $r = n$ achieves the lower bound $\mathbb{E}[C] = n\mathbb{E}[X_{1:n}]$ in (15) with equality. Thus, $r = n$ is the optimal strategy in the heavy traffic regime. For log-concave $\bar{F}_X$, $r = 1$ achieves the lower bound $\mathbb{E}[C] = \mathbb{E}[X]$ in (14) with equality. Thus, in heavy traffic, $r = 1$ gives lowest $\mathbb{E}[T]$ for log-concave $\bar{F}_X$. □

Lemma 4 is illustrated by Figures 12 and 13, where $\mathbb{E}[T]$ calculated analytically using (11) is plotted versus λ for different values of r. Each job is assigned to r servers chosen uniformly at random from $n = 6$ servers. In Figure 12, the service time distribution is $ShiftedExp(\Delta, \mu)$ (which is log-concave) with $\Delta = 1$ and $\mu = 0.5$. When λ is small, more redundancy (higher r) gives lower $\mathbb{E}[T]$, but in the high λ regime, $r = 1$ gives lowest $\mathbb{E}[T]$ and highest service capacity. However, in Figure 13, for a log-convex distribution $HyperExp(p, \mu_1, \mu_2)$, in the high load regime $\mathbb{E}[T]$ decreases as r increases.

Lemma 4 was previously proven for NBU (NWU) instead of log-concave (log-convex) $\bar{F}_X$ in Shah et al. [2013] and Koole and Righter [2008] using a combinatorial argument. Using Theorem 3, we get an alternative and arguably simpler way to prove this result. Note that our version is weaker because log-concavity implies NBU, but the converse is not true in general (see Property 3 in Appendix A).

Due to the network cost of issuing and canceling the replicas, there may be an upper limit $r \le r_{max}$ on the number of replicas. The optimal strategy under this constraint is given by Lemma 5.

Lemma 5 (Optimal r Under $r \le r_{max}$). *For log-convex $\bar{F}_X$, $r = r_{max}$ is optimal. For log-concave $\bar{F}_X$, $r = 1$ is optimal in heavy traffic.*

The proof is similar to Lemma 4 with n replaced by r_{max}.

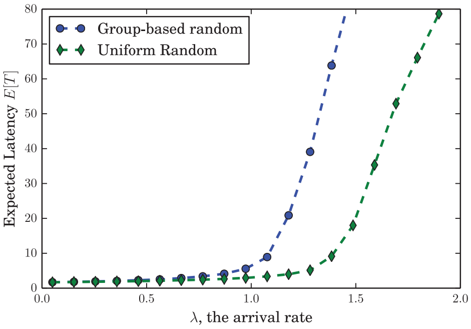


Fig. 14. For service time distribution *ShiftedExp* $(1, 0.5)$, which is log-concave, uniform random scheduling (which staggers relative task start times) gives lower $\mathbb{E}[T]$ than group-based random for all λ. The system parameters are $n = 6, r = 3$.

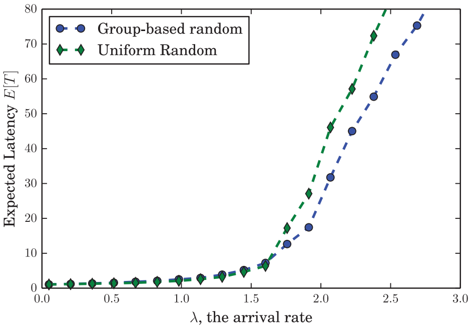


Fig. 15. For service time distribution *HyperExp* $(0.1, 2.0, 0.2)$, which is log-convex, group-based scheduling gives lower $\mathbb{E}[T]$ than uniform random in the high λ regime. The system parameters are $n = 6, r = 3$.

6.3. Choice of the *r* Servers

For a given r, we now compare different policies of choosing the r servers for each job. The choice of the r servers determines the relative starting times of the tasks. By using the bounds in Theorem 3 that hold for any relative task start times, we get the following result.

Lemma 6 (Cost of Different Policies). *Given r, if $\bar{F}_X$ is log-concave (log-convex), the symmetric policy that results in the tasks starting at the same time ($t_i = 0$ for all $1 \leq i \leq r$) results in higher (lower) $\mathbb{E}[C]$ than one that results in $0 < t_i < \infty$ for one or more i.*

Proof. The symmetric policy that results in $t_i = 0$ for all $1 \leq i \leq r$ (e.g., the group-based random policy) results in $\mathbb{E}[C] = r\mathbb{E}[X_{1:r}]$. By Theorem 3, if $\bar{F}_X$ is log-concave, $\mathbb{E}[C] \leq r\mathbb{E}[X_{1:r}]$ for any symmetric policy. Thus, for log-concave distributions, the symmetric policy that results in $0 < t_i < \infty$ for one or more i gives lower $\mathbb{E}[C]$ than the group-based random policy. However, for log-convex distributions, $\mathbb{E}[C] \geq r\mathbb{E}[X_{1:r}]$ with any symmetric policy. Thus, the policies that result in relative task start times $t_i = 0$ for all $1 \leq i \leq r$ give lower $\mathbb{E}[C]$ than other symmetric policies. □

Lemma 7 (Latency in the High λ Regime). *Given r, if $\bar{F}_X$ is log-concave (log-convex), the symmetric policy that results in the tasks starting at the same time ($t_i = 0$ for all $1 \leq i \leq r$) results in higher (lower) $\mathbb{E}[T]$ in the heavy traffic regime than one that results in $0 < t_i < \infty$ for some i.*

Proof. By Corollary 1, the optimal replication strategy in heavy traffic is the one that minimizes $\mathbb{E}[C]$. Then the proof follows from Lemma 6. □

Lemma 7 is illustrated by Figures 14 and 15 for $n = 6$ and $r = 3$. The simulations are run for 100 workloads with 1,000 jobs each. The r tasks may start at different times with the uniform random policy, whereas they always start simultaneously with group-based random policy. Thus, in the high λ regime, the uniform random policy results in lower latency for log-concave $\bar{F}_X$, as observed in Figure 14. But for log-convex $\bar{F}_X$, group-based forking is better in the high λ regime, as seen in Figure 15. For low λ, uniform random policy is better for any $\bar{F}_X$ because it gives lower expected waiting time in the queue.

7. THE GENERAL *K* CASE

We now move to the general k case, where a job requires any k out of n tasks to complete. In practice, the general k case arises in large-scale parallel computing frameworks such as MapReduce and in content downloads from coded distributed storage systems. In this section, we present bounds on the latency and cost of the (n, k) fork-join and (n, k) fork-early-cancel systems. In Section 7.2, we demonstrate an interesting diversity-parallelism trade-off in choosing k.

7.1. Latency and Cost of the (*n*, *k*) Fork-Join System

Unlike the $k = 1$ case, for general k the exact analysis is hard because multiple jobs can be in service simultaneously (e.g., Job A and Job B in Figure 1). Even for the $k = n$ case studied in Nelson and Tantawi [1988] and Varki et al. [208], only bounds on latency are known. We generalize those latency bounds to any k and also provide bounds on cost $\mathbb{E}[C]$. The analysis of $\mathbb{E}[C]$ can be used to estimate the service capacity using Lemma 1.

THEOREM 4 (BOUNDS ON LATENCY). *The latency $\mathbb{E}[T]$ is bounded as follows:*

$$\mathbb{E}[T] \leq \mathbb{E}[X_{k:n}] + \frac{\lambda \mathbb{E}\left[X_{k:n}^2\right]}{2(1 - \lambda \mathbb{E}[X_{k:n}])}, \tag{16}$$

$$\mathbb{E}[T] \geq \mathbb{E}[X_{k:n}] + \frac{\lambda \mathbb{E}\left[X_{1:n}^2\right]}{2(1 - \lambda \mathbb{E}[X_{1:n}])}. \tag{17}$$

The proof is given in Appendix C. In Figure 16, we plot the bounds on latency alongside the simulation values for Pareto service time. The upper bound (16) becomes more loose as k increases, because the split-merge system considered to get the upper bound (see the proof of Theorem 4) becomes worse as compared to the fork-join system. For the special case $k = n$, we can improve the upper bound in Lemma 8 by generalizing the approach used in Nelson and Tantawi [1988].

LEMMA 8 (TIGHTER UPPER BOUND WHEN $k = n$). *For the case $k = n$, another upper bound on latency is given by*

$$\mathbb{E}[T] \leq \mathbb{E}[\max(R_1, R_2, \cdots R_n)], \tag{18}$$

where R_i are independent and identically distributed realizations of the response time R of an $M/G/1$ queue with arrival rate λ and service time distribution F_X.

The proof is given in Appendix C. Transform analysis (see Chapter 25 of Harchol-Balter [2013]) can be used to determine the distribution of R, the response time of an $M/G/1$ queue in terms of $F_X(x)$. The Laplace-Stieltjes transform $R(s)$ of the probability density function of $f_R(r)$ of R is given by

$$R(s) = \frac{sX(s)\left(1 - \frac{\lambda}{\mathbb{E}[X]}\right)}{s - \lambda(1 - X(s))}, \tag{19}$$

where $X(s)$ is the Laplace-Stieltjes transform of the service time distribution $f_X(x)$.

The lower bound on latency (17) can be improved for shifted exponential F_X, generalizing the approach in Varki et al. [2008] based on the memoryless property of the exponential tail.

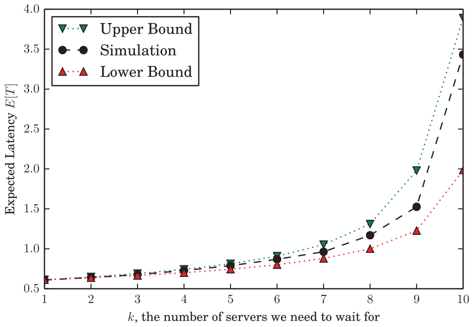


Fig. 16. Bounds on latency $\mathbb{E}[T]$ versus k (Theorem 4) alongside simulation values. The service time $X \sim Pareto(0.5, 2.5)$, $n = 10$, and $\lambda = 0.5$. A tighter upper bound for $k = n$ is evaluated using Lemma 8.

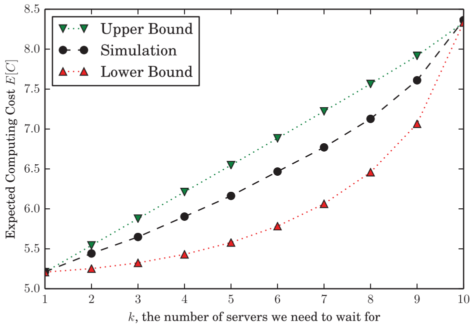


Fig. 17. Bounds on cost $\mathbb{E}[C]$ versus k (Theorem 5) alongside simulation values. The service time $X \sim Pareto(0.5, 2.5)$, $n = 10$, and $\lambda = 0.5$. The bounds are tight for $k = 1$ and $k = n$.

THEOREM 5 (BOUNDS ON COST). *The expected computing cost $\mathbb{E}[C]$ can be bounded as follows:*

$$\mathbb{E}[C] \leq (k-1)\mathbb{E}[X] + (n-k+1)\mathbb{E}[X_{1:n-k+1}], \tag{20}$$

$$\mathbb{E}[C] \geq \sum_{i=1}^{k} \mathbb{E}[X_{i:n}] + (n-k)\mathbb{E}[X_{1:n-k+1}]. \tag{21}$$

The proof is given in Appendix C. Figure 17 shows the bounds alongside the simulation plot of the computing cost $\mathbb{E}[C]$ when F_X is $Pareto(x_m, \alpha)$ with $x_m = 0.5$ and $\alpha = 2.5$. The arrival rate $\lambda = 0.5$, and $n = 10$ with k varying from 1 to 10 on the x-axis. The simulation is run for 100 iterations of 1,000 jobs. We observe that the bounds on $\mathbb{E}[C]$ are tight for $k = 1$ and $k = n$, which can also be inferred from (20) and (21).

7.2. Diversity-Parallelism Trade-Off

In Figure 16, we observed the expected latency increases with k, because we need to wait for more tasks to complete, and the service time X is independent of k. But in most computing and storage applications, the service time X decreases as k increases because each task becomes smaller. We refer to this as the parallelism benefit of splitting a job into more tasks. But as k increases, we lose the diversity benefit provided by redundant tasks and have to wait only for a subset of the tasks to finish. Thus, there is a diversity-parallelism trade-off in choosing the optimal k^* that minimizes latency $\mathbb{E}[T]$. We demonstrate the diversity-parallelism trade-off in simulation plot Figure 18 for service time $X \sim ShiftedExp(\Delta_k, \mu)$, with $\mu = 1.0$, and $\Delta_k = \Delta/k$. As k increases, we lose diversity but the parallelism benefit is higher because each task is smaller. As Δ increases, the optimal k^* shifted upward because the service time distribution becomes "less random," and thus there is less diversity benefit.

We can also observe the diversity-parallelism trade-off mathematically in the low traffic regime for $X \sim ShiftedExp(\Delta/k, \mu)$. If we take $\lambda \to 0$ in (17) and (16), both bounds coincide and we get

$$\lim_{\lambda \to \infty} \mathbb{E}[T] = \mathbb{E}[X_{k:n}] = \frac{\Delta}{k} + \frac{H_n - H_{n-k}}{\mu}, \tag{22}$$

where $H_n = \sum_{i=1}^{n} 1/i$, the n^{th} harmonic number. The parallelism benefit comes from the first term in (22), which reduces with k. The diversity of waiting for k out of n tasks

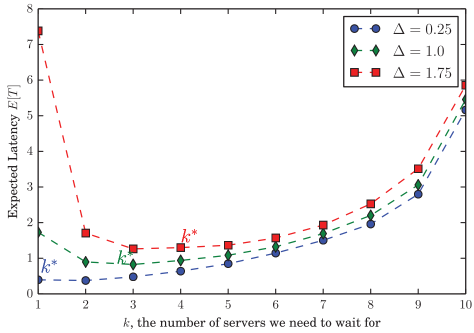


Fig. 18. Expected latency versus k for task service time $X \sim ShiftedExp(\Delta/k, 1.0)$ and arrival rate $\lambda = 0.5$. As k increases, we lose diversity but the parallelism benefit is higher because each task is smaller.

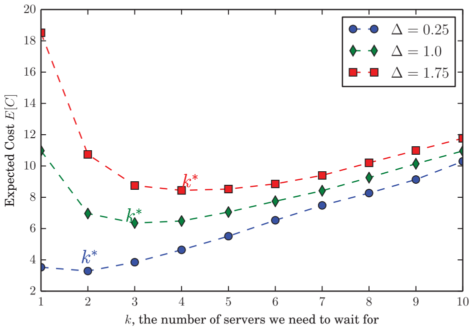


Fig. 19. Expected cost versus k for task service time $X \sim ShiftedExp(\Delta/k, 1.0)$ and arrival rate $\lambda = 0.5$. As k increases, we lose diversity but the parallelism benefit is higher because each task is smaller.

causes the second term to increase with k. The optimal k^* that minimizes (22) strikes a balance between these two opposing trends.

Figure 19 shows a similar diversity-parallelism trade-off in choosing k to minimize the computing cost $\mathbb{E}[C]$. In the heavy traffic regime, by Corollary 1 the policy that minimizes $\mathbb{E}[C]$ also minimizes $\mathbb{E}[T]$. Thus, the same k^* will minimize both $\mathbb{E}[T]$ and $\mathbb{E}[C]$.

7.3. Latency and Cost of the (*n*, *k*) Fork-Early-Cancel System

We now analyze the latency and cost of the (n, k) fork-early-cancel system where the redundant tasks are canceled as soon as any k tasks start service.

THEOREM 6 (LATENCY-COST WITH EARLY CANCELLATION). *The cost $\mathbb{E}[C]$ and an upper bound on the expected latency $\mathbb{E}[T]$ with early cancellation is given by*

$$\mathbb{E}[C] = k\mathbb{E}[X], \tag{23}$$

$$\mathbb{E}[T] \leq \mathbb{E}[\max(R_1, R_2, \cdots R_k)], \tag{24}$$

where R_i are independent and identically distributed realizations of R, the response time of an $M/G/1$ queue with arrival rate $\lambda k/n$ and service time distribution F_X.

The proof is given in Appendix C. The Laplace-Stieltjes transform of the response time R of an $M/G/1$ queue with service time distribution $F_X(x)$ and arrival rate is the same as (19), with λ replaced by $\lambda k/n$.

By comparing the cost $\mathbb{E}[C] = k\mathbb{E}[X]$ in (23) to the bounds in Theorem 5 without early cancellation, we can get insights into when early cancellation is effective for a given service time distribution F_X. For example, when $\bar{F}_X$ is log-convex, the upper bound in (20) is smaller than $k\mathbb{E}[X]$. Thus, we can infer that the (n, k) fork-early-cancel system is always worse than the (n, k) fork-join system when X is log-convex. We also observed this phenomenon in Figure 8 for the $k = 1$ case.

8. GENERAL REDUNDANCY STRATEGY

From the analysis in Sections 5 and 6, we get insights into designing the best redundancy strategy for log-concave and log-convex service time. But it is not obvious to infer the best strategy for arbitrary service time distributions or when only empirical traces of the service time are given. We now propose such a redundancy strategy to minimize the latency, subject to computing and network cost constraints. This strategy can also be used on traces of task service time when closed-form expressions of F_X and its order statistics are not known.

8.1. Generalized Fork-Join Model

We first introduce a general fork-join variant that is a combination of the partial fork introduced in Section 2 and partial early cancellation of redundant tasks.

Definition 7 (*(n, r_f, r, k) Fork-Join System*). For a system of n servers and a job that requires k tasks to complete, we do the following:

—Fork the job to r_f out of the n servers chosen uniformly at random.
—When any $r \leq r_f$ tasks are at the head of queues or in service already, cancel all other tasks immediately. If more than r tasks start service simultaneously, retain r randomly chosen ones out of them.
—When any $k \leq r$ tasks finish, cancel all remaining tasks immediately.

Note that k tasks may finish before some r start service, and thus we may not need to perform the partial early cancellation in the preceding second step.

Recall that the n servers have service time distribution X that is independent and identically distributed across the servers and tasks. The $r_f - r$ tasks that are canceled early help find the shortest r out of the r_f queues, thus reducing waiting time. From the r tasks retained, waiting for any k to finish provides diversity and hence reduces service time.

The special cases (n, n, n, k), (n, n, k, k), and (n, r, r, k) correspond to the (n, k) fork-join, (n, k) fork-early-cancel, and (n, r, k) partial-fork-join systems, respectively, which are defined in Section 2.

8.2. Choosing Parameters r_f and r

We now propose a strategy to choose r_f and r that minimize expected latency $\mathbb{E}[T]$ subject to a computing cost constraint $\mathbb{E}[C] \leq \gamma$, and a network cost constraint $r_f \leq r_{max}$. We impose the second constraint because forking to more servers results in higher network cost of remote-procedure calls (RPCs) to launch and cancel the tasks.

Definition 8 (*Proposed Redundancy Strategy*). Choose r_f and r to minimize $\mathbb{E}[T]$ subject to constraints $\mathbb{E}[C] \leq \gamma$ and $r_f \leq r_{max}$. The solutions are

$$r_f^* = r_{max}, \tag{25}$$

$$r^* = \underset{r \in [0, r_{max}]}{\arg\min}\, \hat{T}(r), \quad \text{subject to} \quad \hat{C}(r) \leq \gamma, \tag{26}$$

where $\hat{T}(r)$ and $\hat{C}(r)$ are estimates of the expected latency $\mathbb{E}[T]$ and cost $\mathbb{E}[C]$, defined as follows:

$$\hat{T}(r) \triangleq \mathbb{E}[X_{k:r}] + \frac{\lambda r \mathbb{E}[X_{k:r}^2]}{2(n - \lambda r \mathbb{E}[X_{k:r}])}, \tag{27}$$

$$\hat{C}(r) \triangleq r\mathbb{E}[X_{k:r}]. \tag{28}$$

To justify the preceding strategy, observe that for a given r, increasing r_f gives higher diversity in finding the queues with the least work left and thus reduces latency. Since $r_f - r$ tasks are canceled early before starting service, r_f affects $\mathbb{E}[C]$ only mildly, through the relative task start times of r tasks that are retained. Thus, we conjecture that it is optimal to set $r_f = r_{max}$ in (25), the maximum value possible under network cost constraints. However, changing r does affect both the computing cost and latency significantly. Therefore, to determine the optimal r, we minimize $\hat{T}(r)$ subject to constraints $\hat{C}(r) \leq \gamma$ and $r \leq r_{max}$ as given in (26).

The estimates $\hat{T}(r)$ and $\hat{C}(r)$ are obtained by generalizing Lemma 3 for group-based random forking to any k and r that may not divide n. When the order statistics of F_X are hard to compute, or F_X itself is not explicitly known, $\hat{T}(r)$ and $\hat{C}(r)$ can be also be found using empirical traces of X.

The sources of inaccuracy in the estimates $\hat{T}(r)$ and $\hat{C}(r)$ are as follows:

(1) For $k > 1$, the latency estimate $\hat{T}(r)$ is a generalization of the split-merge queueing upper bound in Theorem 4. Since the bound becomes loose as k increases, the error $|\hat{T}(r) - \mathbb{E}[T]|$ increases with k.
(2) The estimates $\hat{T}(r)$ and $\hat{C}(r)$ are by definition independent of r_f, which is not true in practice. As explained earlier, for $r_f > r$, the actual $\mathbb{E}[T]$ is generally less than $\hat{T}(r)$, and $\mathbb{E}[C]$ can be slightly higher or lower than $\hat{C}(r)$.
(3) Since the estimates $\hat{T}(r)$ and $\hat{C}(r)$ are based on group-based forking, they consider that all r tasks start simultaneously. Variability in relative task start times can result in actual latency and cost that are different from the estimates. For example, from Theorem 3, we can infer that when $\bar{F}_X$ is log-concave (log-convex), the actual computing cost $\mathbb{E}[C]$ is less than (greater than) $\hat{C}(r)$.

The preceding factor (1) is the largest source of inaccuracy, especially for larger k and λ. Since the estimate $\hat{T}_r$ is an upper bound on the actual latency, the r^* and r_f^* recommended by the strategy are smaller than or equal to their optimal values. Factors (2) and (3) only affect the relative task start times and generally result in a smaller error in estimating $\mathbb{E}[T]$ and $\mathbb{E}[C]$.

8.3. Simulation Results

We now present simulation results comparing the proposed strategy given in Definition 8 to the (n, r, k) partial-fork-join system with r varying from k to n. The service time distributions considered here are neither log-concave nor log-convex, thus making it hard to directly infer the best redundancy strategy using the analysis presented in the previous sections. The simulations are run for 100 workloads with 1,000 jobs each.

In Figure 20, the service time $X \sim \mathit{Pareto}(1, 2.2)$, $n = 10$, $k = 1$, and arrival rate $\lambda = 0.7$. The computing and network cost constraints are $\mathbb{E}[C] \leq 5$ and $r_f \leq 8$, respectively. We observe that the proposed strategy gives a significant latency reduction as compared to the no redundancy case ($r = k$ in the (n, r, k) partial-fork-join system). We observe that the proposed strategy gives a latency-cost trade-off that is better than the (n, r, k) partial-fork-join system. Using partial early cancellation ($r_f > r$) gives an additional reduction in latency by providing greater diversity and helping us find the r out of r_f queues with the least work left.

In Figure 21, we show a case where the cost $\mathbb{E}[C]$ does not always increase with the amount of redundancy r. The task service time X is a mixture of an exponential $\mathit{Exp}(2)$ and a shifted exponential $\mathit{ShiftedExp}(1, 1.5)$, each occurring with equal probability. The other parameters are $n = 10$, $k = 1$, and arrival rate $\lambda = 0.3$. The proposed strategy found using Definition 8 is $r^* = r_f^* = r_{max} = 5$, limited by the $r_f \leq r_{max}$ constraint rather than the $\mathbb{E}[C] \leq \gamma$ constraint. Since $r_f = r$, it coincides exactly with the (n, r, k) partial-fork-join system.

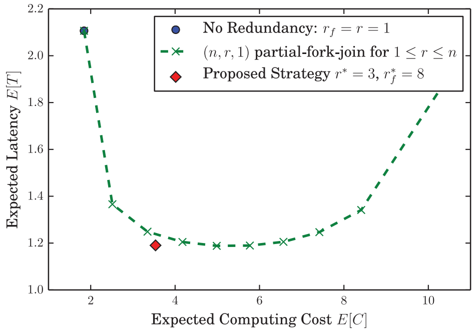


Fig. 20. The latency-cost trade-off of the proposed redundancy strategy is close to that of the best (n, r, k) partial-fork-join system. Service time $X \sim$ *Pareto*$(1, 2.2)$, and the cost constraints are $\mathbb{E}[C] \leq 5$ and $r \leq r_f \leq 8$ The first constraint is active in this example.

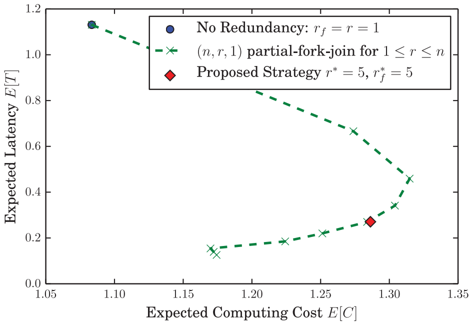


Fig. 21. The latency-cost trade-off of the proposed redundancy strategy is close to that of the best (n, r, k) partial-fork-join system. The service time X is an equiprobable mixture of *Exp*(2) and *ShiftedExp*$(1, 1.5)$, and the cost constraints are $\mathbb{E}[C] \leq 2$ and $r \leq r_f \leq 5$. The second constraint is active in this example.

9. CONCLUDING REMARKS

In this article, we consider a redundancy model where each incoming job is forked to queues at multiple servers and wait for any one replica to finish. We analyze how redundancy affects the latency, and the cost of computing time, and demonstrate how the log-concavity of service time is a key factor affecting the latency-cost trade-off. The following are some key insights from this analysis:

—For log-convex service times, forking to more servers (more redundancy) reduces both latency and cost. However, for log-concave service times, more redundancy can reduce latency only at the expense of an increase in cost.

—Early cancellation of redundant requests can save both latency and cost for log-concave service time, but it is not effective for log-convex service time.

Using these insights, we also propose a general redundancy strategy for an arbitrary service time distribution, which may be neither log-concave nor log-convex. This strategy can also be used on empirical traces of service time, when a closed-form expression of the distribution is not known.

Ongoing work includes developing online strategies to simultaneously learn the service time distribution and the best redundancy strategy. More broadly, the proposed redundancy techniques can be used to reduce latency in several applications beyond the realm of cloud storage and computing systems, such as crowdsourcing, algorithmic trading, and manufacturing.

APPENDIXES

A. LOG-CONCAVITY OF $\bar{F}_X$

In this section, we present some properties and examples of log-concave and log-convex random variables that are relevant to this work. For more properties, please see Bagnoli and Bergstrom [2005].

PROPERTY 1 (JENSEN'S INEQUALITY). *If $\bar{F}_X$ is log-concave, then for $0 < \theta < 1$ and for all $x, y \in [0, \infty)$,*

$$\Pr(X > \theta x + (1 - \theta)y) \geq \Pr(X > x)^{\theta} \Pr(X > y)^{1-\theta}. \tag{29}$$

The inequality is reversed if $\bar{F}_X$ is log-convex.

PROOF. Since $\bar{F}_X$ is log-concave, $\log \bar{F}_X$ is concave. Taking log on both sides on (29), we get the Jensen's inequality, which holds for concave functions. □

In past literature, saying that X is log-concave usually means that f is log-concave. This implies that F and $\bar{F}$. However log-convex f does not always imply log-convexity of F and $\bar{F}$.

PROPERTY 2 (SCALING). *If $\bar{F}_X$ is log-concave, for $0 < \theta < 1$,*

$$\Pr(X > x) \leq \Pr(X > \theta x)^{1/\theta}. \tag{30}$$

The inequality is reversed if $\bar{F}_X$ is log-convex.

PROOF. We can derive (30) by setting $y = 0$ in (29):

$$\Pr(X > \theta x + (1 - \theta)0) \geq \Pr(X > x)^{\theta} \Pr(X > 0)^{1-\theta}, \tag{31}$$

$$\Pr(X > \theta x) \geq \Pr(X > x)^{\theta}. \tag{32}$$

To get (32), we observe that if $\bar{F}_X$ is log-concave, then $\Pr(X > 0)$ has to be 1. Otherwise, log-concavity is violated at $x = 0$. Raising both sides of (32) to power $1/\theta$, we get (30). The reverse inequality of log-convex $\bar{F}_X$ can be proved similarly. □

PROPERTY 3 (SUBMULTIPLICATIVITY). *If $\bar{F}_X$ is log-concave, the conditional tail probability of X satisfies for all $t, x > 0$*

$$\Pr(X > x + t | X > t) \leq \Pr(X > x), \tag{33}$$

$$\Leftrightarrow \Pr(X > x + t) \leq \Pr(X > x) \Pr(X > t). \tag{34}$$

The preceding inequalities are reversed if $\bar{F}_X$ is log-convex.

PROOF.

$$\Pr(X > x) \Pr(X > t) \tag{35}$$

$$= \Pr\left(X > \frac{x}{x+t}(x + t)\right) \Pr\left(X > \frac{t}{x+t}(x + t)\right), \tag{36}$$

$$\geq \Pr(X > x + t)^{\frac{x}{x+t}} \Pr(X > x + t)^{\frac{t}{x+t}}, \tag{37}$$

where we apply Property 2 to (36) to get (37). Equation (33) follows from (37). □

Note that for exponential F_X, which is memoryless, (33) holds with equality. Thus, log-concave distributions can be thought to have "optimistic memory" because the conditional tail probability decreases over time. However, log-convex distributions have "pessimistic memory" because the conditional tail probability increases over time. The definition of the notion NBU in Koole and Righter [2008] is same as (33). By Property 3, log-concavity of $\bar{F}_X$ implies that X is NBU. NBU distributions are referred to as light-everywhere in Shah et al. [2013] and new-longer-than-used in Sun et al. [2015].

PROPERTY 4 (MEAN RESIDUAL LIFE). *If $\bar{F}_X$ is log-concave (log-convex), $\mathbb{E}[X - t | X > t]$, the mean residual life after time $t > 0$ has elapsed is nonincreasing (nondecreasing) in t.*

Proof of Lemma 2. Lemma 2 is true for log-concave $\bar{F}_X$ if $r\mathbb{E}[X_{1:r}] \leq (r+1)\mathbb{E}[X_{1:r+1}]$ for all integers $r \geq 1$. This inequality can be simplified as follows:

$$r\mathbb{E}[X_{1:r}] \leq (r+1)\mathbb{E}[X_{1:r+1}] \tag{38}$$

$$\Leftrightarrow r\int_0^\infty \Pr(X_{1:r} > x)dx \leq \int_0^\infty (r+1)\Pr(X_{1:r+1} > x)dx, \tag{39}$$

$$\Leftrightarrow r\int_0^\infty \Pr(X > x)^r dx \leq \int_0^\infty (r+1)\Pr(X > x)^{r+1} dx, \tag{40}$$

$$\Leftrightarrow \int_0^\infty \Pr\left(X > \frac{x'}{r}\right)^r dx' \leq \int_0^\infty \Pr\left(X > \frac{x'}{r+1}\right)^{r+1} dx'. \tag{41}$$

We get (39) using the fact that the expected value of a nonnegative random variable is equal to the integral of its tail distribution. To get (40), observe that since $X_{1:r} = \min(X_1, X_2, \cdots, X_r)$ for independent and identically distributed X_i, we have $\Pr(X_{1:r} > x) = \Pr(X > x)^r$ for all $x > 0$. Similarly, $\Pr(X_{1:r+1} > x) = \Pr(X > x)^{r+1}$. Next we perform a change of variables on both sides of (40) to get (41).

Now we use Property 2 to compare the two integrands in (41). Setting $\theta = r/r+1$ and $x = x'/r$ in Property 2, we get

$$\Pr\left(X > \frac{x'}{r}\right)^r \leq \Pr\left(X > \frac{x'}{r+1}\right)^{r+1} \qquad \text{for all } x' \geq 0. \tag{42}$$

Hence, by (42) and the equivalences in (38) through (41), it follows that for log-concave $\bar{F}_X$, $r\mathbb{E}[X_{1:r}]$ is nondecreasing in r. For log-convex $\bar{F}_X$, we can show that $r\mathbb{E}[X_{1:r}]$ is nonincreasing in r by reversing all preceding inequalities.

Property 5 (Hazard Rates). *If $\bar{F}_X$ is log-concave (log-convex), then the hazard rate $h(x)$, which is defined by $-\bar{F}'_X(x)/\bar{F}_X(x)$, is nondecreasing (nonincreasing) in x.*

Property 6 (Coefficient of Variation). *The coefficient of variation $C_v = \sigma/\mu$ is the ratio of the standard deviation σ and mean μ of random variable X. For log-concave (log-convex) X, $C_v \leq 1$ ($C_v \geq 1$), and $C_v = 1$ when X is pure exponential.*

Property 7 (Examples of Log-Concave $\bar{F}_X$). *The following distributions have log-concave $\bar{F}_X$:*

—*Shifted exponential (exponential plus constant $\Delta > 0$)*
—*Uniform over any convex set*
—*Weibull with shape parameter $c \geq 1$*
—*Gamma with shape parameter $c \geq 1$*
—*Chi-squared with degrees of freedom $c \geq 2$.*

Property 8 (Examples of Log-Convex $\bar{F}_X$). *The following distributions have log-convex $\bar{F}_X$:*

—*Exponential*
—*Hyperexponential (mixture of exponentials)*
—*Weibull with shape parameter $0 < c < 1$*
—*Gamma with shape parameter $0 < c < 1$.*

B. PROOFS FOR THE $K = 1$ Case

Proof of Theorem 3. Using (4), we can express the cost C in terms of the relative task start times t_i and S as follows. Since only r tasks are invoked, the relative start times $t_{r+1}, \ldots, t_n$ are equal to ∞:

$$C = S + (S - t_2)^+ + \cdots + (S - t_r)^+ , \tag{43}$$

where S is the time between the start of service of the earliest task and when any one of the r tasks finishes. The tail distribution of S is given by

$$\Pr(S > s) = \prod_{i=1}^{r} \Pr(X > s - t_i). \tag{44}$$

By taking expectation on both sides of (43) and simplifying, we get

$$\begin{aligned}
\mathbb{E}[C] &= \sum_{u=1}^{r} \int_{t_u}^{\infty} \Pr(S > s) ds, && (45) \\
&= \sum_{u=1}^{r} u \int_{t_u}^{t_{u+1}} \Pr(S > s) ds, && (46) \\
&= \sum_{u=1}^{r} u \int_{0}^{t_{u+1} - t_u} \Pr(S > t_u + x) dx, && (47) \\
&= \sum_{u=1}^{r} u \int_{0}^{t_{u+1} - t_u} \prod_{i=1}^{u} \Pr(X > x + t_u - t_i) dx. && (48)
\end{aligned}$$

We now prove that for log-concave $\bar{F}_X$, $\mathbb{E}[C] \geq \mathbb{E}[X]$. The proof that $\mathbb{E}[C] \leq \mathbb{E}[X]$ when $\bar{F}_X$ is log-convex follows similarly with all following inequalities reversed. We express the integral in (48) as

$$\begin{aligned}
\mathbb{E}[C] &= \sum_{u=1}^{r} u \left(\int_0^\infty \prod_{i=1}^{u} \Pr(X > x + t_u - t_i) dx - \int_0^\infty \prod_{i=1}^{u} \Pr(X > x + t_{u+1} - t_i) dx \right), && (49) \\
&= \sum_{u=1}^{r} \left(\int_0^\infty \prod_{i=1}^{u} \Pr\left(X > \frac{x'}{u} + t_u - t_i\right) dx' - \int_0^\infty \prod_{i=1}^{u} \Pr\left(X > \frac{x'}{u} + t_{u+1} - t_i\right) dx' \right), && (50) \\
&= \mathbb{E}[X] + \sum_{u=2}^{r} \int_0^\infty \left(\prod_{i=1}^{u} \Pr\left(X > \frac{x'}{u} + t_u - t_i\right) - \prod_{i=1}^{u-1} \Pr\left(X > \frac{x'}{u-1} + t_u - t_i\right) \right) dx', && (51) \\
&\geq \mathbb{E}[X], && (52)
\end{aligned}$$

where in (49) we express each integral in (48) as a difference of two integrals from 0 to ∞. In (50), we perform a change of variables $x = x'/u$. In (51), we rearrange the grouping of the terms in the sum; the u^{th} negative integral is put in the $u + 1$ term of the summation. Then the first term of the summation is simply $\int_0^\infty \Pr(X > x) dx$, which is equal to $\mathbb{E}[X]$. In (51), we use the fact that each term in the summation in (50) is positive when $\bar{F}_X$ is log-concave. This is shown in Lemma 9.

Next we prove that for log-concave $\bar{F}_X$, $\mathbb{E}[C] \leq r\mathbb{E}[X_{1:r}]$. Again, the proof of $\mathbb{E}[C] \geq r\mathbb{E}[X_{1:r}]$ when $\bar{F}_X$ is log-convex follows with all of the following inequalities reversed:

$$\mathbb{E}[C] \leq \sum_{u=1}^{r} u \int_0^{t_{u+1}-t_u} \prod_{i=1}^{u} \Pr\left(X > \frac{u(x+t_u-t_i)}{r}\right)^{r/u} dx, \tag{53}$$

$$= \sum_{u=1}^{r} \left(\int_0^{\infty} \prod_{i=1}^{u} \Pr\left(X > \frac{x' + u(t_u - t_i)}{r}\right)^{r/u} dx' - \int_0^{\infty} \prod_{i=1}^{u} \Pr\left(X > \frac{x' + u(t_{u+1} - t_i)}{r}\right)^{r/u} dx' \right), \tag{54}$$

$$= \int_0^{\infty} \Pr\left(X > \frac{x'}{r}\right)^{r} dx' + \sum_{u=2}^{r} \left(\int_0^{\infty} \prod_{i=1}^{u} \Pr\left(X > \frac{x' + u(t_u - t_i)}{r}\right)^{r/u} dx' \right.$$
$$\left. - \int_0^{\infty} \prod_{i=1}^{u-1} \Pr\left(X > \frac{x' + (u-1)(t_u - t_i)}{r}\right)^{\frac{r}{u-1}} dx' \right), \tag{55}$$

$$\leq r\mathbb{E}[X_{1:r}], \tag{56}$$

where we get (53) by applying Property 2 to (48). In (54), we express the integral as a difference of two integrals from 0 to ∞ and perform a change of variables $x = x'/u$. In (55), we rearrange the grouping of the terms in the sum; the u^{th} negative integral is put in the $u + 1$ term of the summation. The first term is equal to $r\mathbb{E}[X_{1:r}]$. We use Lemma 10 to show that each term in the summation in (55) is negative when $\bar{F}_X$ is log-concave.

LEMMA 9. *If $\bar{F}_X$ is log-concave,*

$$\prod_{i=1}^{u} \Pr\left(X > \frac{x'}{u} + t_u - t_i\right) \geq \prod_{i=1}^{u-1} \Pr\left(X > \frac{x'}{u-1} + t_u - t_i\right). \tag{57}$$

The inequality is reversed for log-convex $\bar{F}_X$.

PROOF OF LEMMA 9. We bound the left-hand side expression as follows:

$$\prod_{i=1}^{u} \Pr\left(X > \frac{x}{u} + t_u - t_i\right) = \Pr(S > t_u) \prod_{i=1}^{u} \Pr\left(X > \frac{x}{u} + t_u - t_i | X > t_u - t_i\right), \tag{58}$$

$$= \Pr(S > t_u) \Pr\left(X > \frac{x}{u}\right)^{\frac{u-1}{u-1}} \times \prod_{i=1}^{u-1} \Pr\left(X > \frac{x}{u} + t_u - t_i | X > t_u - t_i\right), \tag{59}$$

$$\geq \Pr(S > t_u) \prod_{i=1}^{u-1} \Pr\left(X > \frac{x}{u} + t_u - t_i | X > t_u - t_i\right)^{\frac{u}{u-1}}, \tag{60}$$

$$\geq \Pr(S > t_u) \prod_{i=1}^{u-1} \Pr\left(X > \frac{x}{u-1} + t_u - t_i | X > t_u - t_i\right), \tag{61}$$

$$= \prod_{i=1}^{u-1} \Pr\left(X > \frac{x}{u-1} + t_u - t_i\right), \tag{62}$$

where we use Property 3 to get (60). The inequality in (61) follows from applying Property 2 to the conditional distribution $\Pr(Y > x'/u) = \Pr(X > x'/u + t_u - t_i | X > t_u - t_i)$, which is also log-concave.

For log-convex $\bar{F}_X$, all nequalities can be reversed. □

LEMMA 10. *If $\bar{F}_X$ is log-concave,*

$$\prod_{i=1}^{u} \Pr\left(X > \frac{x + u(t_u - t_i)}{r}\right)^{r/u} \leq \prod_{i=1}^{u-1} \Pr\left(X > \frac{x + (u-1)(t_u - t_i)}{r}\right)^{\frac{r}{u-1}}. \tag{63}$$

The inequality is reversed for log-convex $\bar{F}_X$.

PROOF OF LEMMA 10. We start by simplifying the left-hand side expression, raised to the power $(u-1)/r$:

$$\begin{aligned}\prod_{i=1}^{u} \Pr\left(X > \frac{x + u(t_u - t_i)}{r}\right)^{(u-1)/u} &= \Pr\left(X > \frac{x}{r}\right)^{\frac{u-1}{u}} \prod_{i=1}^{u-1} \Pr\left(X > \frac{x + u(t_u - t_i)}{r}\right)^{\frac{u-1}{u}}, &(64)\\ &= \prod_{i=1}^{u-1} \Pr\left(X > \frac{x}{r}\right)^{\frac{1}{u}} \Pr\left(X > \frac{x + u(t_u - t_i)}{r}\right)^{\frac{u-1}{u}}, &(65)\\ &\leq \prod_{i=1}^{u-1} \Pr\left(X > \frac{x + (u-1)(t_u - t_i)}{r}\right), &(66)\end{aligned}$$

where (66) follows from the log-concavity of $\Pr(X > x)$ and the Jensen's equality. The inequality is reversed for log-convex $\bar{F}_X$.

C. PROOFS FOR GENERAL *K*

PROOF OF THEOREM 4. To find the upper bound on latency, we consider a related queueing system called the *split-merge* queueing system. In the split-merge system, all queues are blocked and cannot serve subsequent jobs until k out of n tasks of the current job are complete. Thus, the latency of the split-merge system serves as an upper bound on that of the fork-join system. In the split-merge system, we observe that jobs are served one by one, and no two jobs are served simultaneously. Thus, it is equivalent to an $M/G/1$ queue with Poisson arrival rate λ and service time $X_{k:n}$. The expected latency of an $M/G/1$ queue is given by the Pollaczek-Khinchine formula (see Chapter 5 of Gallager [2013]), and it reduces to the upper bound in (16).

To find the lower bound, we consider a system where the job requires k out of n tasks to complete, but all jobs arriving before it require only one task to finish. Then the expected waiting time in the queue is equal to the second term in (16) with k set to 1. Adding the expected service time $\mathbb{E}[X_{k:n}]$ to this lower bound on expected waiting time, we get the lower bound (17) on the expected latency. □

PROOF OF LEMMA 8. This upper bound is a generalization of the bound on the mean response time of the (n, n) fork-join system with exponential service time presented in Nelson and Tantawi [1988]. To find the bound, we first observe that the response times experienced by the tasks in the n queues form a set of associated random variables [Esary et al. 1967]. Then we use the property of associated random variables that their expected maximum is less than that for independent variables with the same marginal distributions. Unfortunately, this approach cannot be extended to the $k < n$ case, as this property of associated variables does not hold for the k^{th}-order statistic for $k < n$. □

PROOF OF THEOREM 5. A key observation used in proving the cost bounds is that at least $n-k+1$ out of the n tasks of a job i start service at the same time. This is because when the k^{th} task of Job $(i-1)$ finishes, the remaining $n-k$ tasks are canceled immediately. These $n-k+1$ queues start working on the tasks of Job i at the same time.

To prove the upper bound, we divide the n tasks into two groups: the $k-1$ tasks that can start early and the $n-k+1$ that start at the same time after the last tasks of the previous job are terminated. We consider a constraint that all $k-1$ tasks in the first group and one of the remaining $n-k+1$ tasks needs to be served for completion of the job. This gives an upper bound on the computing cost because we are not taking into account the case where more than one task from the second group can finish service before the $k-1$ tasks in the first group. For the $n-k+1$ tasks in the second group, the computing cost is equal to $n-k+1$ times the time taken for one of them to complete. The computing time spent on the first $k-1$ tasks is at most $(k-1)\mathbb{E}[X]$. Adding this to the second group's cost, we get the upper bound (20).

We observe that the expected computing cost for the k tasks that finish is at least $\sum_{i=1}^{k}\mathbb{E}[X_{i:n}]$, which takes into account full diversity of the redundant tasks. Since we need k tasks to complete in total, at least one of the $n-k+1$ tasks that start simultaneously needs to be served. Thus, the computing cost of the $(n-k)$ redundant tasks is at least $(n-k)\mathbb{E}[X_{1:n-k+1}]$. Adding this to the lower bound on the first group's cost, we get (21). □

PROOF OF THEOREM 6. Since exactly k tasks are served and others are canceled before they start service, it follows that the expected computing cost $\mathbb{E}[C] = k\mathbb{E}[X]$. In the sequel, we find an upper bound on the latency of the (n,k) fork-early-cancel system.

First observe that in the (n,k) fork-early-cancel system, the $n-k$ redundant tasks that are canceled early help find the k shortest queues. The expected task arrival rate at each server is $\lambda k/n$, which excludes the redundant tasks that are canceled before they start service.

Consider an (n,k,k) partial fork system without redundancy, where the k tasks of each job are assigned to k out of n queues chosen uniformly at random. The job exits the system when all k tasks are complete. The expected task arrival rate at each server is $\lambda k/n$, same as the (n,k) fork-early-cancel system. However, the (n,k) fork-early-cancel system gives lower latency because having the $n-k$ redundant tasks provides diversity and helps find the k shortest queues. Thus, the latency of the (n,k,k) partial-fork-join system is bounded below by that of the (n,k) fork-early-cancel system.

Now let us upper bound the latency $\mathbb{E}[T^{(pf)}]$ of the partial fork system. Each queue has arrival rate $\lambda k/n$ and service time distribution F_X. Using the approach in Nelson and Tantawi [1988], we can show that the response times (waiting plus service time) R_i, $1 \leq i \leq k$ of the k queues serving each job form a set of associated random variables. Then by the property that the expected maximum of k associated random variables is less than the expected maximum of k independent variables with the same marginal distributions, we can show that

$$\mathbb{E}[T] \leq \mathbb{E}[T^{(pf)}], \tag{67}$$

$$\leq \mathbb{E}[\max(R_1, R_2, \ldots R_k)]. \tag{68}$$

The expected maximum can be numerically evaluated from distribution of R. From the transform analysis given in Chapter 25 of Harchol-Balter [2013], we know that the Laplace-Stieltjes transform $R(s)$ of the probability density of R is the same as (19) but with λ replaced by $\lambda k/n$. □

ACKNOWLEDGMENTS

We thank Da Wang, Devavrat Shah, Sem Borst, and Rhonda Righter for helpful suggestions to improve this work.

REFERENCES

Ganesh Ananthanarayanan, Ali Ghodsi, Scott Shenker, and Ion Stoica. 2013. Effective straggler mitigation: Attack of the clones. In *Proceedings of the 2013 USENIX Conference on Networked Systems Design and Implementation*. 185–198.

Mark Bagnoli and Ted Bergstrom. 2005. Log-concave probability and its applications. *Economic Theory* 26, 2, 445–469.

Jinhua Cao and Yuedong Wang. 1991. The NBUC and NWUC classes of life distributions. *Journal of Applied Probability* 28, 2, 473–479.

Shengbo Chen, Ulas C. Kozat, Longbo Huang, Prasun Sinha, Guanfeng Liang, Xin Liu, Yin Sun, and Ness B. Shroff. 2014. When queueing meets coding: Optimal-latency data retrieving scheme in storage clouds. In *Proceedings of the 2014 IEEE International Conference on Communications*.

Jeffrey Dean and Luis Barroso. 2013. The tail at scale. *Communications of the ACM* 56, 2, 74–80.

Jeffrey Dean and Sanjay Ghemawat. 2008. MapReduce: Simplified data processing on large clusters. *Communications of the ACM* 51, 1, 107–113.

J. Esary, F. Proschan, and D. Walkup. 1967. Association of random variables, with applications. *Annals of Mathematics and Statistics* 38, 5, 1466–1474.

L. Flatto and S. Hahn. 1984. Two parallel queues created by arrivals with two demands I. *SIAM Journal on Applied Mathematics* 44, 5, 1041–1053.

R. Gallager. 2013. *Stochastic Processes: Theory for Applications*. Cambridge University Press, Cambridge, UK.

K. Gardner, S. Zbarsky, S. Doroudi, M. Harchol-Balter, E. Hyytiä, and A. Scheller-Wolf. 2015. Reducing latency via redundant requests: Exact analysis. In *Proceedings of the 2015 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems*. 347–360.

Mor Harchol-Balter. 2013. *Performance Modeling and Design of Computer Systems: Queueing Theory in Action*. Cambridge University Press, Cambridge, UK.

G. Joshi, Y. Liu, and E. Soljanin. 2012. Coding for fast content download. In *Proceedings of the 2012 Allerton Conference on Communication, Control, and Computing*. 326–333.

G. Joshi, Y. Liu, and E. Soljanin. 2014. On the delay-storage trade-off in content download from coded distributed storage. *IEEE Journal on Selected Areas on Communications* 32, 5, 989–997.

Gauri Joshi, Emina Soljanin, and Gregory Wornell. 2015. Queues with redundancy: Latency-cost analysis. In *Proceedings of the 2015 ACM SIGMETRICS Workshop on Mathematical Modeling and Analysis*.

G. Kabatiansky, E. Krouk, and S. Semenov (Eds.). 2005. Coding of messages at the transport layer of the data network. In *Error Correcting Coding and Security for Data Networks: Analysis of the Superchannel Concept*. Wiley, 191–211.

Swanand Kadhe, Emina Soljanin, and Alex Sprintson. 2015. Analyzing the download time of availability codes. In *Proceedings of the 2015 International Symposium on Information Theory (ISIT'15)*.

Yusik Kim, Rhonda Righter, and Ronald Wolff. 2009. Job replication on multiserver systems. *Advances in Applied Probability* 41, 2, 546–575.

Ger Koole and Rhonda Righter. 2008. Resource allocation in grid computing. *Journal of Scheduling* 11, 3, 163–173.

A. Kumar, R. Tandon, and T. C. Clancy. 2014. On the latency of heterogeneous MDS queue. In *Proceedings of the 2014 IEEE Global Communications Conference (GLOBECOM'14)*. 2375–2380.

A. M. Lee and P. A. Longton. 1959. Queueing process associated with airline passenger check-in. *Journal of the Operational Research Society* 10, 1, 56–71.

Guangfeng Liang and Ulas Kozat. 2014. TOFEC: Achieving optimal throughput-delay trade-off of cloud storage using erasure codes. In *Proceedings of the 2014 IEEE International Conference on Communications*.

N. F. Maxemchuk. 1975. Dispersity routing. In *Proceedings of the 1975 International Conference on Communications (ICC'75)*. 41.10–41.13.

M. Mitzenmacher. 1996. *The Power of Two Choices in Randomized Load Balancing*. Ph.D. Dissertation. University of California at Berkeley.

R. Nelson and A. Tantawi. 1988. Approximate analysis of fork/join synchronization in parallel queues. *IEEE Transactions on Computers* 37, 6, 739–743.

K. Ousterhout, P. Wendell, M. Zaharia, and I. Stoica. 2013. Sparrow: Distributed, low latency scheduling. In *Proceedings of the 2013 ACM Symposium on Operating Systems Principles (SOSP'13)*. 69–84.

Nihar Shah, Kangwook Lee, and Kannan Ramchandran. 2013. When do redundant requests reduce latency? In *Proceedings of the 2013 Allerton Conference on Communication, Control, and Computing*.

Nihar Shah, Kangwook Lee, and Kannan Ramachandran. 2014. The MDS queue: Analyzing the latency performance of erasure codes. In *Proceedings of the 2014 IEEE International Symposium on Information Theory*.

Yin Sun, Zizhan Zheng, Can Emre Koksal, Kyu-Han Kim, and Ness B. Shroff. 2015. Provably delay efficient data retrieving in storage clouds. In *Proceedings of the 2015 IEEE Conference on Computer Communications (INFOCOM'15)*.

Elizabeth Varki, Arif Merchant, and Hui Chen. 2008. The M/M/1 Fork-Join Queue with Variable Sub-Tasks. Retrieved March 30, 2017, from http://www.cs.unh.edu/∼varki/publication/2002-nov-open.pdf.

Jeremy Visschers, Ivo Adan, and Gideon Weiss. 2012. A product form solution to a system with multi-type jobs and multi-type servers. *Queueing Systems* 70, 3, 269–298.

A. Vulimiri, P. B. Godfrey, R. Mittal, J. Sherry, S. Ratnasamy, and S. Shenker. 2013. Low latency via redundancy. In *Proceedings of the 2013 ACM Conference on Emerging Networking Experiments and Technologies (CoNEXT'13)*. ACM, New York, NY, 283–294.

D. Wang, G. Joshi, and G. Wornell. 2014. Efficient Task Replication for Fast Response Times in Parallel Computation. Retrieved March 30, 2017, from http://allegro.mit.edu/pubs/posted/conference/2014-wang-joshi-wornell-sigmetrics.pdf.

D. Wang, G. Joshi, and G. Wornell. 2015. Using straggler replication to reduce latency in large-scale parallel computing (extended version). arXiv:1503.03128 [cs.dc].

Yu Xiang, Tian Lan, Vaneet Aggarwal, and Yih Farn R. Chen. 2014. Joint latency and cost optimization for erasure-coded data center storage. *SIGMETRICS Performance Evaluation Review* 42, 2, 3–14.

Received October 2015; revised November 2016; accepted February 2017