Efficient Redundancy Techniques for Latency Reduction in Cloud Systems

Gauri Joshi, Emina Soljanin, and Gregory Wornell

Abstract

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 thus reduce average 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 provides a fundamental understanding of when and how redundancy gives a cost-efficient reduction in latency. For a general task service time distribution, we compare different redundancy strategies, for e.g. the number of redundant tasks, and time when they are issued and canceled. We get the insight that the *log-concavity* of the task service distribution is a key factor in determining whether adding redundancy helps. If the service distribution is log-convex, then adding maximum redundancy reduces both latency and cost. And if it is log-concave, then less redundancy, and early cancellation of redundant tasks is more effective. We also present a heuristic strategy that achieves a good latency-cost trade-off for an arbitrary service distribution.

This work also generalizes and extends some results in the famously hard analysis of fork-join queues.

I. Introduction

A. 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. A side-effect of the sharing of resources is the variability in the latency experienced by the user due to queueing, pre-emption by other jobs with higher priority, server outages etc. The problem becomes further aggravated when the user is executing a job with several parallel tasks on the cloud, because the slowest task becomes the bottleneck in job completion. Thus, ensuring seamless, low-latency service to the end-user 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, running a task on multiple machines and waiting for the earliest copy to finish can significantly reduce the latency [1]. Similarly, in cloud storage systems requests to access the content can be assigned to multiple replicas, such that it is only sufficient to download one replica. This can help reduce latency significantly.

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 which offer computing as a service, the server time spent is proportional to the money spent in renting the machines. In this work we aim to understand this trade-off between latency and computing cost and propose scheduling policies that can achieve a good trade-off. Our analysis also results in some fundamental advances in the analysis of queues with redundant requests.

B. Previous Work

Systems Work: One of the earliest instances of exploiting redundancy to reduce latency is the use of multiple routing paths to send packets in networks. See [2] for a detailed survey. A similar idea has also been recently studied in systems [3]. In large-scale cloud computing frameworks such as MapReduce [4], the slowest tasks of

G. Joshi (gauri@mit.edu) and G. Wornell (gww@mit.edu) are with the Department of Electrical Engineering and Computer Science, MIT, Cambridge 02139, USA. Emina Soljanin (emina@bell-labs.com) is with Bell Labs Alcatel-Lucent, Murray Hill NJ 07974, USA.

This work was supported in part by NSF under Grant No. CCF-1319828, AFOSR under Grant No. FA9550-11-1-0183, and a Schlumberger Faculty for the Future Fellowship.

TABLE I: Organization of main latency-cost analysis results presented in the rest of the paper. 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.

	$k = 1$ **(Replicated) Case**	**General** k
Full forking to all n **servers**	Section IV: Comparison of strategies with and without early task cancellation	Section VI: Bounds on latency and cost, and the diversity-parallelism trade-off
Partial forking to r **out of** n **servers**	Section V: Effect of r and the choice of servers on latency and cost	Section VII: Heuristic redundancy strategy for cost-efficient latency reduction

TABLE II: Latency-optimal and cost-optimal redundancy strategies for the $k = 1$ (replicated) case. 'Canceling redundancy early' means that instead of waiting for any 1 task to finish, we cancel redundant tasks as soon as any 1 task begins service.

	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), High load: $r = 1$ (fork to one)	$r = 1$	$r = n$	$r = n$

a job (stragglers) become a bottleneck in its completion. Several recent works in systems such as [5], [6] 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, and in particular the cost of resources. We now review some of that work.

Exponential Service Time: In distributed storage systems, erasure coding can be used to store a content file on n servers such that it can be recovered by accessing any k out of the n servers. The latency experienced by the user can be reduced by forking a download request to n servers and waiting for any k to respond. In [7], [8] we found bounds on the expected latency using the (n, k) fork-join model with exponential service time. This is a generalization of the (n, n) fork-join system, which was actively studied in queueing literature [9]–[11] around two decades ago. In recent years, there is a renewed interest in fork-join queues due to their application to distributed computing frameworks such as MapReduce.

Another related model with a centralized queue instead of queues at each of the n servers was analyzed in [12]. The latency behavior with heterogeneous job classes for the replicated ($k = 1$) case is presented in [13]. Other related works include [14], [15].

General Service Time: Few practical systems have exponentially distributed service time. For example, studies of download time traces from Amazon S3 [16], [17] indicate that the service time is not exponential in practice, but instead a shifted exponential. For service time distributions that are 'new-worse-than-used' [18], it is shown in [19] that it is optimal to fork a job to maximum number of servers. The choice of scheduling policy for new-worse-than-used (NWU) and new-better-than-used (NBU) distributions is also studied in [20]–[22]. 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 non-zero fixed cost of removal of redundant requests are presented in [21]. The total server time spent on each job is considered in [23], [24] for a distributed system without considering queueing of requests. In [25] we presented an analysis of the latency and cost of the (n, k) fork-join with and without early cancellation of redundant tasks.

C. Our Contributions

In this work, we consider a general service time distribution, unlike exponential service time assumed in many previous works. 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 also serves as a powerful tool to compare different redundancy strategies in the high traffic regime.

Table I gives the different scenarios considered for latency-cost analysis in the rest of the paper. This analysis 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.

- Suppose we fork a job to queues at n servers, and wait for any 1 out of n tasks to be served. An alternate strategy is to cancel the redundant tasks early, as soon as any 1 task reaches the head its queue. We 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 $\bar{F}_X$.
- Suppose we fork each job to only a subset r out of the n servers, and wait for any one of the r tasks to finish. Then 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 heuristic strategy to decide 1) how many servers to fork to, and 2) when to cancel the redundant tasks, for an arbitrary service time that may be neither log-concave nor log-convex.

D. Organization

In Section II we describe the fork-join system model and the latency and cost metrics. Section III gives the key concepts used in this work.

The organization of the main results in the rest of the paper is given in Table I. In Section IV and Section V we consider the replicated ($k = 1$) case, and get insights into choosing the best redundant queueing strategies, depending on log-concavity of the distribution. In Section IV we compare the strategies with and without early cancellation of redundancy, and in Section V we consider partial forking to a subset of the servers. In Section VI we determine bounds on latency and cost for any k, generalizing some of the fundamental results on fork-join queues. We also demonstrate a diversity-parallelism trade-off in choosing k. In Section VII we present a heuristic strategy, drawing from the insights from the latency-cost analysis in previous sections.

Section VIII summarizes the results and provides future perspectives. Some properties and examples of log-concavity are given in Appendix A. The proofs of the $k = 1$ and general k cases are deferred to Appendix B and Appendix C respectively.

II. Problem Formulation

A. Fork-Join Model and its Variants

Consider a distributed system with n statistically identical servers. Each incoming job is forked into n tasks, assigned to first-come first-serve queues at each of the n servers. The n tasks are designed in such a way that completion of any k tasks is sufficient to complete the job. When any k tasks are served, the remaining tasks are canceled. We refer to this system as the (n, k) fork-join system, defined formally as follows.

Definition 1 ((n, k) fork-join system)**.** *Each incoming job is forked into n tasks that join a first-come first-serve queue at each of the n servers. When any k tasks finish service, all remaining tasks are canceled and abandon their queues immediately.*

Fig. 1 illustrates the $(3, 2)$ fork-join system. The job exits the system when any 2 out of 3 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. General k arise in approximate computing algorithms, or in content download from a coded distributed storage system.

Instead of waiting for k tasks to finish, we could cancel the redundant tasks earlier when any k tasks reach the heads of their queues, or are in service. A similar idea has been proposed in systems work [6]. We refer to this variant as the (n, k) fork-early-cancel system defined formally as follows.

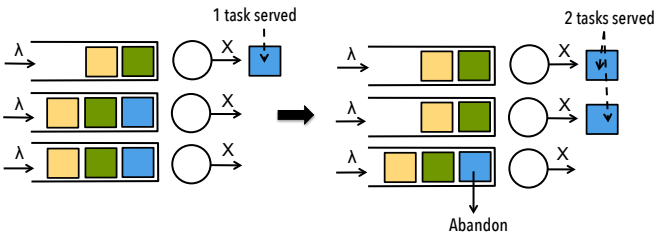


Fig. 1: The $(3, 2)$ fork-join system. When any 2 out of 3 tasks of a job are served (as seen for the blue job on the right), the third task abandons its queue and the job exits the system.

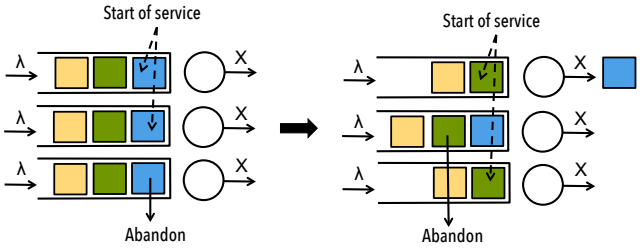


Fig. 2: The $(3, 2)$ fork-early-cancel system. When any 2 out of 3 tasks of a job are in service, the third task abandons (seen for the blue job on the left, and green job on the right).

Definition 2 ((n, k) fork-early-cancel system)**.** *Each incoming job is forked to the n servers. When any k tasks are in service, we cancel the redundant tasks immediately. If more than k tasks start service simultaneously, we preserve any k chosen uniformly at random.*

Fig. 2 illustrates the $(3, 2)$ fork-early-cancel system. Early cancellation can save the total time spent per job (computing cost) because the redundant tasks are canceled before the servers start working on them. But it could result in an increase in latency because we have to wait for all k remaining tasks to finish, the slowest of which could take a long time to finish. Thus, we lose the diversity advantage of having to waiting only for a subset of tasks to complete.

In another variant defined formally below, we fork a job to r out of the n servers. We refer to this as the (n, r, k) partial fork-join system defined as follows.

Definition 3 ((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, least-work-left etc. Partial forking can save on the computing cost as well as the network cost, which is proportional to the number of servers each job is forked to. In this work we develop insights into the best choice of r and the scheduling policy to achieve a good latency-cost trade-off.

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 VII we propose a heuristic algorithm to find a redundancy strategy that is a combination of partial forking and early cancellation.

B. Arrival and Service Distributions

Consider that jobs arrive to the system at rate λ per second, according to a Poisson process. The Poisson assumption is required only for the exact analysis and bounds of latency $\mathbb{E}[T]$ (defined below). All other results on cost $\mathbb{E}[C]$, and the insights into choosing the best redundancy strategy hold for any arrival process.

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, network congestion, and sharing of computing resources between multiple processes. We model it by the service time $X > 0$, with cumulative distribution function $F_X(x)$ and assume that it is i.i.d. across requests and servers. Dependence of service time across servers can be modeled by adding a constant to service time X. For example, some recent work [16], [17] 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.

In this paper we use $\bar{F}_X(x) = \Pr(X > x)$ to denote the tail probability function of X. We use $X_{k:n}$ to denote the k^{th} smallest of n i.i.d. random variables $X_1, X_2, \ldots, X_n$.

C. Latency and Cost Metrics

Using more redundant tasks (larger n) reduces latency, but generally results in additional cost of computing resources. We now define the metrics of the latency and resource cost whose trade-off is analyzed in the rest of the paper.

Definition 4 (Expected Latency)**.** *The expected latency $\mathbb{E}[T]$ is defined as the expected time from when a job arrives, until when k of its tasks are complete.*

Since we need to wait for the first k out of n tasks to be complete, we expect that $\mathbb{E}[T]$ will decrease as k reduces (or n increases).

Definition 5 (Expected Computing Cost)**.** *The expected computing cost $\mathbb{E}[C]$ is the expected total time spent by the servers serving a job, not including the time spent in the queue.*

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

Although we focus on $\mathbb{E}[C]$ as the cost metric in this paper, 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 each job is forked to, which is n for the (n, k) fork-join model described above. 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 [7], [8].

III. Preliminary Concepts

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

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

Intuitively, redundancy affects latency and cost in two opposing ways. Redundancy provides diversity of having to wait only for a subset of tasks to finish, thus reducing service time. But the redundant time spent by multiple servers can increasing the waiting time in queue for subsequent jobs. The second effect dominates at high arrival rates. Thus $\mathbb{E}[C]$, the expected total time spent by the servers on each job, it can be used to determine the service capacity of the system.

Claim 1 (Service Capacity in terms of $\mathbb{E}[C]$)**.** *For a system of n servers with a symmetric forking policy, and any arrival process with rate λ, the service capacity, that is, the maximum λ such that $\mathbb{E}[T] < \infty$ is*

$$\lambda_{max} = \frac{n}{\mathbb{E}[C]} \tag{1}$$

A symmetric forking policy is defined as follows.

Definition 6 (Symmetric Forking)**.** *In a symmetric forking policy, the tasks of each job are forked to all or a subset of the n servers such that the expected task arrival rate is equal across the servers.*

Most commonly used policies: random, round-robin, shortest queue etc. are symmetric across the n servers.

of Claim 1: Since the forking policy is symmetric, the mean time spent by each server per job is $\mathbb{E}[C]/n$. Thus the server utilization is $\rho = \lambda\mathbb{E}[C]/n$. To keep the system stable such that $\mathbb{E}[T] < \infty$, the server utilization must be less than 1. The result in (1) follows from this. ∎

The system with lower $\mathbb{E}[C]$ has lower $\mathbb{E}[T]$ when λ is close to the service capacity. Thus, Claim 1 serves as a powerful technique to compare the latency with different redundancy policies systems in the high λ regime.

B. Log-concavity of $\bar{F}_X$

When the tail distribution $\bar{F}_X$ of service time is either log-concave or log-convex, we get a clearer understanding of how redundancy affects latency and cost. For example, if we fork a job to all n servers and wait for any 1

copy to finish, the expected computing cost $\mathbb{E}[C] = n\mathbb{E}[X_{1:n}]$. It can be shown that $n\mathbb{E}[X_{1:n}]$ is non-decreasing (non-increasing) in n when $\bar{F}_X$ is log-concave (log-convex). Log-concavity of $\bar{F}_X$ is defined formally as follows.

Definition 7 (Log-concavity and log-convexity of $\bar{F}_X$)**.** *The tail distribution function $\bar{F}_X$ 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 X is log-concave (log-convex) in this paper, we mean that $\bar{F}_X$ is log-concave (log-convex). Some interesting properties and examples of log-concavity are given in Appendix A. We refer readers to [26] for a more detailed study of log-concave distributions.

A canonical log-concave distribution is the shifted exponential, denoted by *ShiftedExp*(Δ, μ). It is an exponential with rate μ, plus a constant shift $\Delta \geq 0$. An example of a log-convex distribution is the hyper-exponential distribution, denoted by *HyperExp*(μ_1, μ_2, p). It is a mixture of two exponentials with rates μ_1 and μ_2 where the exponential with rate μ_1 occurs with probability p.

Remark 1. *Log-concavity of X implies that X is 'new-better-than-used', a notion which is considered in [19]. Other names used to refer to new-better-than-used distributions are 'light-everywhere' in [21] and 'new-longer-than-used' in [22].*

Many random variables with log-concave (log-convex) $\bar{F}_X$ are light (heavy) tailed respectively, but neither imply the other. Unlike the tail of a distribution which 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.

C. Relative Task Start Times

The relative start times of the n tasks of a job is an important factor affecting the latency and cost. Let the relative start times of the tasks be $t_1 \leq t_2 \leq \cdots t_n$ where $t_1 = 0$ without loss of generality and t_i for $i > 1$ are measured from the instant when the earliest task starts service. For instance, if $n = 3$ tasks start at times 3, 4 and 7, then $t_1 = 0$, $t_2 = 4 - 3 = 1$ 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 ∞.

Let S be the time from when the earliest task starts service, until any k tasks finish. Thus it is the k^{th} smallest of $X_1 + t_1, X_2 + t_2, \cdots, X_n + t_n$, where X_i are i.i.d. with distribution F_X. The computing cost C is given by,

$$C = S + |S - t_2|^+ + \cdots + |S - t_n|^+. \tag{2}$$

Using (2) we get several crucial insights in the rest of the paper. For instance, in Section V 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 Claim 1 we can infer that it is optimal to fork a job to all n servers when $\bar{F}_X$ is log-convex.

IV. $k = 1$ 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 II. We get the insight that it is better to cancel redundant tasks early if $\bar{F}_X$ is log-concave. On the other hand, if $\bar{F}_X$ is log-convex, retaining the redundant tasks is better.

A. Latency-Cost Analysis

Lemma 1. *The latency T of the $(n, 1)$ fork-join system is equivalent in distribution to that of an $M/G/1$ queue with service time $X_{1:n}$.*

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

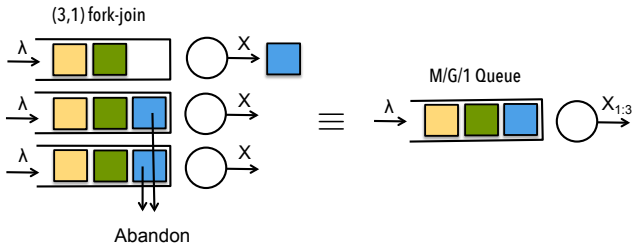


Fig. 3: Equivalence of the $(n, 1)$ fork-join system with an $M/G/1$ queue with service time $X_{1:n}$, the minimum of n i.i.d. random variables $X_1, X_2, \ldots, X_n$.

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

$$\mathbb{E}[T] = \mathbb{E}\left[T^{M/G/1}\right] = \mathbb{E}[X_{1:n}] + \frac{\lambda \mathbb{E}\left[X_{1:n}^2\right]}{2(1 - \lambda \mathbb{E}[X_{1:n}])} \tag{3}$$

$$\mathbb{E}[C] = n \cdot \mathbb{E}[X_{1:n}] \tag{4}$$

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

Proof: By Lemma 1, the latency of the $(n, 1)$ fork-join system is 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 (3). 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 $C = nX_{1:n}$ in (2) . ■

From (4) and Claim 1 we can infer the following result about the service capacity.

Corollary 1. *The service capacity of the* $(n, 1)$ *fork-join system is* $\lambda_{\max} = 1/\mathbb{E}[X_{1:n}]$*, which is non-decreasing in* n.

In Corollary 2, and Corollary 3 below we characterize how $\mathbb{E}[T]$ and $\mathbb{E}[C]$ vary with n.

Corollary 2. *For the* $(n, 1)$ *fork-join system with any service distribution* F_X*, the expected latency* $\mathbb{E}[T]$ *is non-increasing with* n.

The behavior of $\mathbb{E}[C] = n\mathbb{E}[X_{1:n}]$ as n increases depends on the log-concavity of X as given by Property 4 in Appendix A. Using that we can infer the following corollary about $\mathbb{E}[C]$.

Corollary 3. *If* $\bar{F}_X$ *is log-concave (log-convex), then* $\mathbb{E}[C]$ *is non-decreasing (non-increasing) in* n.

Fig. 4 and Fig. 5 show the expected latency versus cost for log-concave and log-convex $\bar{F}_X$, respectively. In Fig. 4, the arrival rate $\lambda = 0.25$, and X is shifted exponential *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), we can reduce latency without any additional cost. In Fig. 5, arrival rate $\lambda = 0.5$, and X is hyperexponential *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.

B. 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 $(n, 1)$ 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*

$$\mathbb{E}[T] = \mathbb{E}\left[T^{M/G/n}\right], \tag{5}$$

$$\mathbb{E}[C] = \mathbb{E}[X], \tag{6}$$

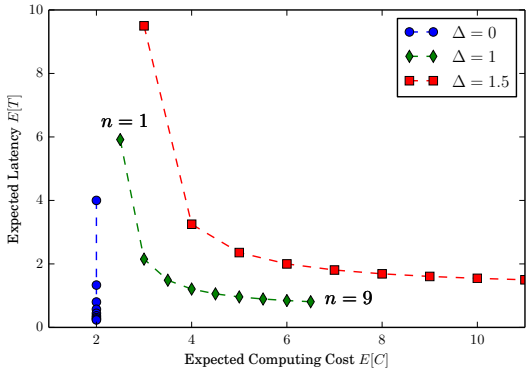


Fig. 4: The service time $X \sim \Delta + \text{Exp}(\mu)$ (log-concave), with $\mu = 0.5$, $\lambda = 0.25$. As n increases along each curve, $\mathbb{E}[T]$ decreases and $\mathbb{E}[C]$ increases. Only when $\Delta = 0$, latency reduces at no additional cost.

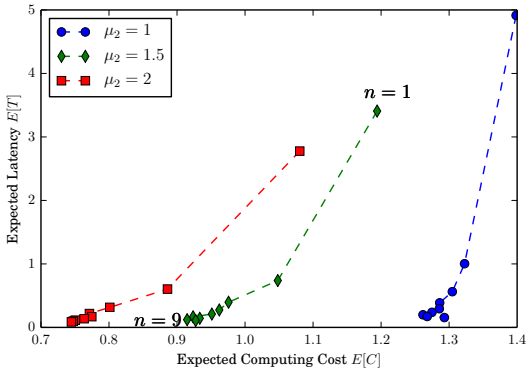


Fig. 5: The service time $X \sim \textit{HyperExp}(0.4, \mu_1, \mu_2)$ (log-convex), with $\mu_1 = 0.5$, different values of μ_2, and $\lambda = 0.5$. Expected latency and cost both reduce as n increases along each curve.

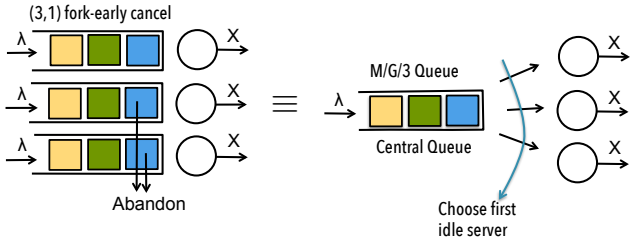


Fig. 6: Equivalence of the $(n, 1)$ fork-early cancel system to an $M/G/n$ queue with each server taking time $X \sim F_X$ to serve task, i.i.d. across servers and tasks.

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 tasks reaches the head of its queue, all others are canceled immediately. The redundant tasks help find the shortest queue, and exactly one task of each job is served by the first server that becomes idle. Thus, as illustrated in Fig. 6, the latency of the $(n, 1)$ fork-early-cancel system is equivalent in distribution to an $M/G/n$ queue. Hence $\mathbb{E}[T] = \mathbb{E}\left[T^{M/G/n}\right]$ and $\mathbb{E}[C] = \mathbb{E}[X]$. ■

The exact analysis of mean response time $\mathbb{E}\left[T^{M/G/n}\right]$ has long been an open problem in queueing theory. A well-known approximation given by [27] is,

$$\mathbb{E}\left[T^{M/G/n}\right] \approx \mathbb{E}[X] + \frac{\mathbb{E}\left[X^2\right]}{2\mathbb{E}[X]^2}\mathbb{E}\left[W^{M/M/n}\right] \tag{7}$$

where $\mathbb{E}\left[W^{M/M/n}\right]$ is the expected waiting time in an $M/M/n$ queueing system with load $\rho = \lambda\mathbb{E}[X]/n$. It can be evaluated using the Erlang-C model [28, Chapter 14].

Using Property 4 to compare the $\mathbb{E}[C]$ with and without early cancellation, given by Theorem 1 and Theorem 2 we get the following corollary.

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 equal to (less than or equal to) that of $(n, 1)$ fork-join join.*

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. In the high λ regime, we can use by Claim 1 and Corollary 4 to imply the following result about expected latency $\mathbb{E}[T]$.

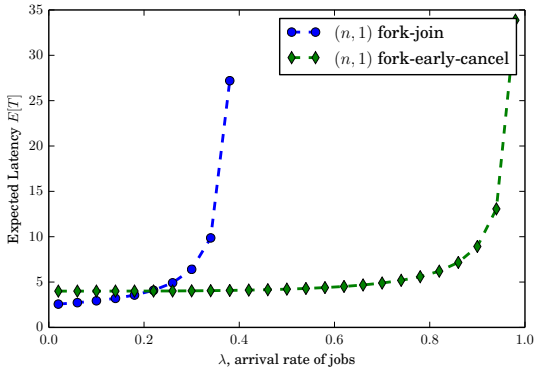


Fig. 7: For the $(4, 1)$ system with service time $X \sim$ *ShiftedExp*$(2, 0.5)$ which is log-concave, early cancellation is better in the high λ regime, as given by Corollary 5.

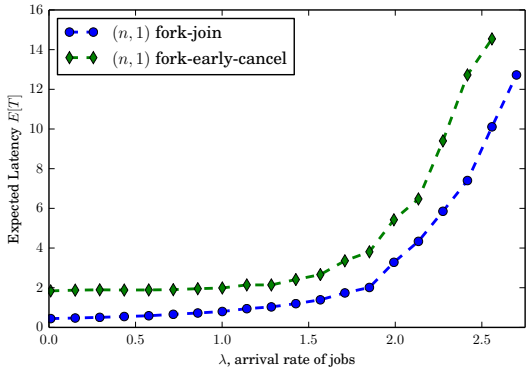


Fig. 8: For the $(4, 1)$ system with $X \sim$ *HyperExp*$(0.1, 1.5, 0.5)$, which is log-convex, early cancellation is worse in both low and high λ regimes, as given by Corollary 5.

Corollary 5. *If $\bar{F}_X$ is log-concave, 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.*

Fig. 7 and Fig. 8 illustrate Corollary 5. Fig. 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 Fig. 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, 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 indicates that early cancellation is better for larger Δ. When Δ is small, there is 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.

V. Partial Forking ($k = 1$ case)

In many cloud computing applications the number of servers n is large. Thus full forking of jobs to all servers can be expensive in the network cost of making remote-procedure-calls to issue and cancel the tasks. Hence it is more practical to fork a job to a subset r out of the n servers, referred to as the (n, r, k) partial-fork-join system in Definition 3. In this section we analyze the $k = 1$ case, that is, the $(n, r, 1)$ partial-fork-join system, where an incoming job is forked to some r out n servers and we wait for any 1 task to finish.

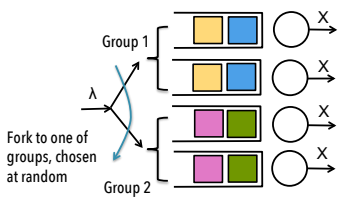
The r servers are chosen using a symmetric forking policy (Definition 6). Some examples of symmetric forking policies are:

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.

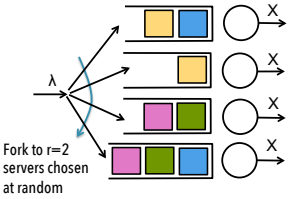
Fig. 9 illustrates the $(4, 2, 1)$ partial-fork-join system with the group-based random and the uniform-random forking policies. In the sequel, we develop insights into the best choice of r and forking policy for a given F_X.

A. 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 2 below.



(a) Group-based random



(b) Uniform random

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

Lemma 2 (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*

$$\mathbb{E}[T] = \mathbb{E}[X_{1:r}] + \frac{\lambda r \mathbb{E}\left[X_{1:r}^2\right]}{2(n - \lambda r \mathbb{E}[X_{1:r}])} \tag{8}$$

$$\mathbb{E}[C] = r\mathbb{E}[X_{1:r}] \tag{9}$$

Proof: Since 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 (9) and Claim 1, we can infer that the service capacity (maximum supported λ) for an $(n, r, 1)$ system with group-based random forking is

$$\lambda_{max} = \frac{n}{r\mathbb{E}[X_{1:r}]} \tag{10}$$

From (10) 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 high traffic regime. By Property 4 in Appendix A, the optimal r is $r = 1$ ($r = n$) for log-concave (log-convex) $\bar{F}_X$.

For other symmetric forking 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 below.

Theorem 3. *Consider an $(n, r, 1)$ partial-fork join system, with a symmetric forking policy. For any relative task start times t_i, $\mathbb{E}[C]$ can be bounded as follows.*

$$r\mathbb{E}[X_{1:r}] \geq \mathbb{E}[C] \geq \mathbb{E}[X] \qquad \text{if } \bar{F}_X \text{ is log-concave} \tag{11}$$

$$\mathbb{E}[X] \geq \mathbb{E}[C] \geq r\mathbb{E}[X_{1:r}] \qquad \text{if } \bar{F}_X \text{ is log-convex} \tag{12}$$

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 on both sides in (2), and show that for log-concave and log-convex $\bar{F}_X$, we get the bounds in (11) and (12), which are independent of the relative task start times t_i. The detailed proof is given in Appendix B.

In the sequel, we use the bounds in Theorem 3 to gain insights into choosing the best r and best scheduling policy when $\bar{F}_X$ is log-concave or log-convex.

B. Choosing the optimal r

By Property 4 in Appendix A, $r\mathbb{E}[X_{1:r}]$ is non-decreasing (non-increasing) with r for log-concave (log-convex) $\bar{F}_X$. Using this observation in conjuction with Theorem 3, we get the following result about which r minimizes $\mathbb{E}[C]$.

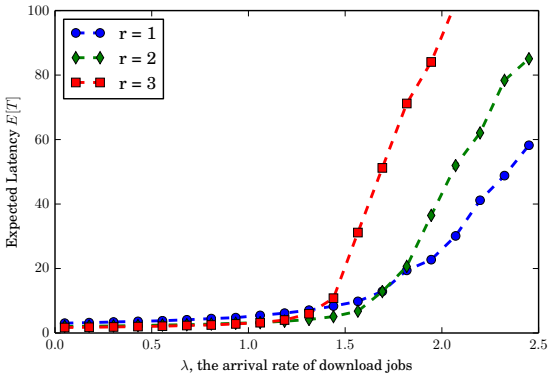


Fig. 10: For $X \sim$ *ShiftedExp*$(1, 0.5)$ which is log-concave, forking to less (more) servers reduces expected latency in the low (high) λ regime.

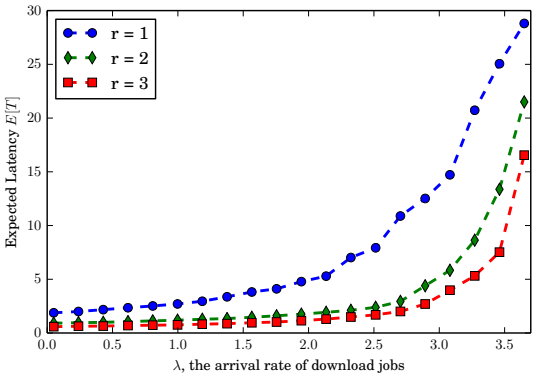


Fig. 11: For $X \sim$ *HyperExp*(p, μ_1, μ_2) with $p = 0.1$, $\mu_1 = 1.5$, and $\mu_2 = 0.5$ which is log-convex, forking to more servers (larger r) gives lower expected latency for all λ.

Corollary 6 (Cost versus 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).*

From Corollary 6 and Claim 1 we get the following result on which r minimizes the service capacity.

Corollary 7 (Service Capacity vs. r)**.** *For a system of n servers with symmetric forking of each job to r servers, $r = 1$ $(r = n)$ gives the highest service capacity when $\bar{F}_X$ is log-concave (log-convex).*

Now let us now determine the value of r that minimizes expected latency for log-concave and log-convex $\bar{F}_X$. When the arrival rate λ is small, $\mathbb{E}[T]$ is dominated by the service time $\mathbb{E}[X_{1:r}]$ which is non-increasing in r. This can be observed for the group-based forking policy by taking $\lambda \to 0$ in (8). Thus we get the following result.

Corollary 8 (Latency vs r in low traffic regime)**.** *Forking to all n servers, that is, $r = n$ gives the lowest$\mathbb{E}[T]$ in the low λ regime for any service time distribution F_X.*

Since the system with the higher service capacity has lower latency in the high traffic regime, we can infer the following from Corollary 7.

Corollary 9 (Latency vs. r in high traffic regime)**.** *Given the number of servers n and symmetric forking of each job to r servers, if $\bar{F}_X$ is log-concave (log-convex) then, $r = 1$ $(r = n)$ gives lowest $\mathbb{E}[T]$ in the high traffic regime.*

Corollary 8 and Corollary 9 are illustrated by Fig. 10 and Fig. 11 where $\mathbb{E}[T]$ 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 Fig. 10 the service time distribution is *ShiftedExp*(Δ, μ) (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. On the other hand in Fig. 11, for a log-convex distribution *HyperExp*(p, μ_1, μ_2), in the high load regime $\mathbb{E}[T]$ decreases as r increases.

Corollary 9 was previously proven for new-better-than-used (new-worse-than-used) instead of log-concave (log-convex) $\bar{F}_X$ in [19], [21], using a combinatorial argument. Our version is weaker because log-concavity implies new-better-than-used but the converse is not true in general (see Property 3 in Appendix A). Using Theorem 3, we get an alternative, and arguably simpler way to prove Corollary 9.

Corollary 8 and Corollary 9 imply that forking to more servers (larger r) gives lower $\mathbb{E}[T]$ in the low λ and high λ regimes. But we observe in Fig. 11 that this true for all λ. From (8) we can prove thus for the group-based random policy. But the proof for other symmetric forking policies remains open.

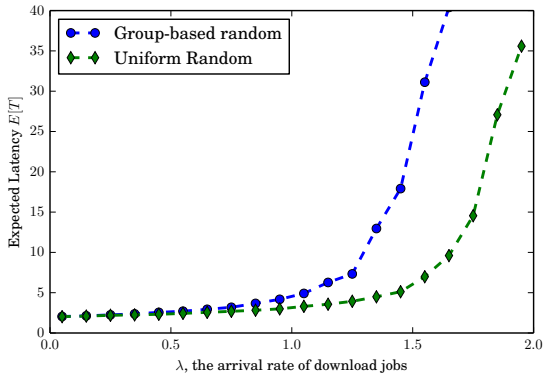


Fig. 12: For service distribution *ShiftedExp*$(1, 0.5)$ which is log-concave, uniform random scheduling (which staggers relative task start times) gives lower $\mathbb{E}[T]$ than group-based random for all λ. The system parameters are $n = 6$, $r = 2$.

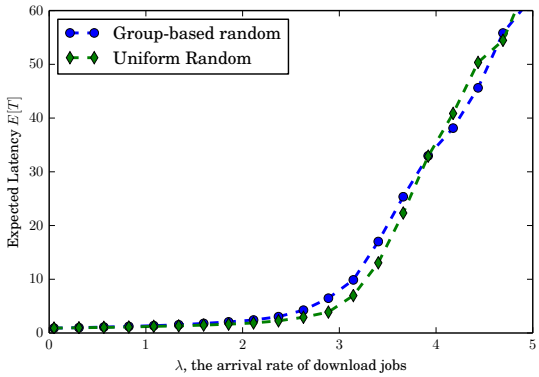


Fig. 13: For service distribution *HyperExp*$(0.1, 1.5, 0.5)$ which is log-convex, group-based scheduling gives lower $\mathbb{E}[T]$ than uniform random in the high λ regime. The system parameters are $n = 6$, $r = 2$.

C. Choosing the Forking Policy

For a given r, we now compare different policies of choosing the r servers for each job. By Theorem 3 we know that $\mathbb{E}[C] \geq r\mathbb{E}[X_{1:r}]$ for log-convex $\bar{F}_X$ for a symmetric policy with any relative task start times. Since $\mathbb{E}[C] = r\mathbb{E}[X_{1:r}]$ when all tasks start simutaneously, this implies that staggering the task start times does not help. For log-concave $\bar{F}_X$, since $\mathbb{E}[C] \leq r\mathbb{E}[X_{1:r}]$, higher diversity in the relative starting times of tasks gives lower $\mathbb{E}[T]$. We state this formally is Corollary 10 below.

Corollary 10. *For a given n and r, and task arrival distribution at each queue, if $\bar{F}_X$ is log-concave (log-convex), the symmetric forking policy which results in more diversity in the relative task start times gives lower (higher) latency in the high λ regime.*

This phenomenon is illustrated in Fig. 12 and Fig. 13 for the uniform random and group-based random forking policies. Uniform random forking gives more diversity in relative start times than the group-based random policy. Thus, in the high λ regime, uniform random forking gives lower latency for log-concave $\bar{F}_X$, as observed in Fig. 12. But for log-convex $\bar{F}_X$, group-based forking is better in the high λ regime as seen in Fig. 13. For low λ, uniform random forking is better for any $\bar{F}_X$ because it gives lower expected waiting time in queue.

VI. The General k Case

We now move from the $k = 1$ (replicated) case to general k, where a job requires any k tasks to complete. In practice, the general k case arises in large-scale parallel computing frameworks such as MapReduce, and in content download 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. The (n, r, k) partial-fork-join system is not considered here, but we present a heuristic strategy for it in Section VII.

For general k, there is also an interesting diversity-parallelism trade-off in choosing k. Having larger k means smaller size tasks, and thus smaller expected service time X per task. But the diversity in waiting for k out of n reduces as k increases. We demonstrate this trade-off in Section VI-B.

A. Latency and Cost of the (n, k) fork-join system

Unlike the $k = 1$ case, for general k exact analysis is hard because multiple jobs can be in service simultaneously (for e.g. blue and green job in Fig. 1). Even for the $k = n$ case studied in [10], [11], 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 Claim 1.

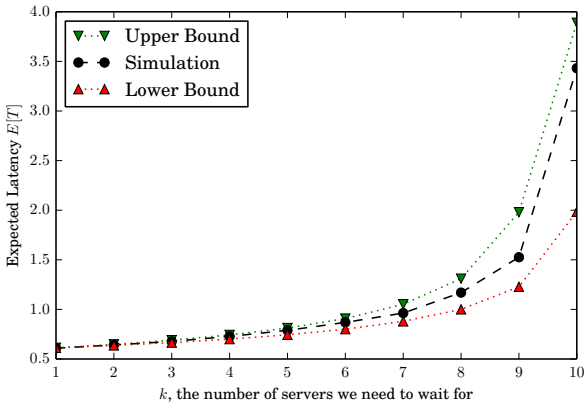


Fig. 14: Bounds on latency $\mathbb{E}[T]$ versus k (Theorem 4), alongside simulation values. The service time $X \sim$ *Pareto*$(0.5, 2.5)$, $n = 10$, and $\lambda = 0.5$. A tigher upper bound for $k = n$ is evaluated using Lemma 3.

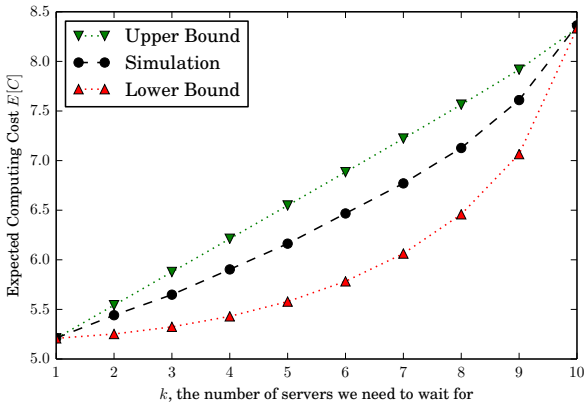


Fig. 15: Bounds on cost $\mathbb{E}[C]$ versus k (Theorem 5) alongside simulation values. The service time $X \sim$ *Pareto*$(0.5, 2.5)$, $n = 10$, and $\lambda = 0.5$. The bounds are tight for $k = 1$ and $k = n$.

Theorem 4 (Bounds on Latency)**.** *The latency* $\mathbb{E}[T]$ *is bounded as follows.*

$$\mathbb{E}[T] \leq \mathbb{E}[X_{k:n}] + \frac{\lambda((\mathbb{E}[X_{k:n}])^2 + \mathrm{Var}[X_{k:n}])}{2(1 - \lambda\mathbb{E}[X_{k:n}])} \tag{13}$$

$$\mathbb{E}[T] \geq \mathbb{E}[X_{k:n}] + \frac{\lambda((\mathbb{E}[X_{1:n}])^2 + \mathrm{Var}[X_{1:n}])}{2(1 - \lambda\mathbb{E}[X_{1:n}])} \tag{14}$$

where $\mathbb{E}[X_{k:n}]$ *and* $\mathrm{Var}[X_{k:n}]$ *are the mean and variance of* $X_{k:n}$*, the* k^{th} *smallest in* n *i.i.d. random variables* $X_1, X_2, \cdots, X_n$*, with* $X_i \sim F_X$*.*

In Fig. 14 we plot the bounds on latency alongside the simulation values for Pareto service time. The upper bound (13) becomes more loose as k increases, because the split-merge system considered to get the upper bound (see 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 3 below, by generalizing the approach used in [10].

Lemma 3 (Tighter Upper bound when $k = n$)**.** *For the case* $k = n$*, another upper bound on latency is given by,*

$$\mathbb{E}[T] \leq \mathbb{E}[\max(R_1, R_2, \cdots R_n)], \tag{15}$$

where R_i *are i.i.d. realizations of the response time* R *of an* $M/G/1$ *queue with arrival rate* λ*, service distribution* F_X*.*

Transform analysis [28, Chapter 25] 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,

$$R(s) = \frac{sX(s)\left(1 - \frac{\lambda}{\mathbb{E}[X]}\right)}{s - \lambda(1 - X(s))}, \tag{16}$$

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

The lower bound on latency (14) can be improved for shifted exponential F_X, generalizing the approach in [11] based on the memoryless property of the exponential tail.

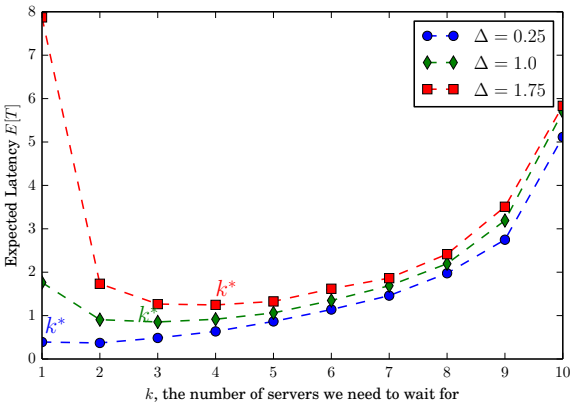


Fig. 16: Diversity-parallelism trade-off. As k increases, diversity decreases, but parallelism benefit is higher because the size of each task reduces. Task service time $X \sim$ *ShiftedExp*$(\Delta/k, 1.0)$, and arrival rate $\lambda = 0.5$.

Lemma 4 (Tighter Lower Bound for Shifted Exponential F_X)**.** *The latency* $\mathbb{E}[T]$ *is lower bounded by,*

$$\mathbb{E}[T] \geq \Delta + \frac{1}{n\mu} + \frac{\lambda\left(\left(\Delta + \frac{1}{n\mu}\right)^2 + \left(\frac{1}{n\mu}\right)^2\right)}{2\left(1 - \lambda\left(\Delta + \frac{1}{n\mu}\right)\right)} + \sum_{j=1}^{k-1} \frac{1}{(n-j)\mu - \lambda}. \tag{17}$$

Theorem 5 (Bounds on Cost)**.** *The expected computing cost* $\mathbb{E}[C]$ *can be bounded as follows.*

$$\mathbb{E}[C] \leq (k-1)\mathbb{E}[X] + (n-k+1)\mathbb{E}[X_{1:n-k+1}] \tag{18}$$

$$\mathbb{E}[C] \geq \sum_{i=1}^{k} \mathbb{E}[X_{i:n}] + (n-k)\mathbb{E}[X_{1:n-k+1}] \tag{19}$$

Fig. 15 shows the bounds alongside simulation plot of the computing cost $\mathbb{E}[C]$ when F_X is *Pareto*(x_m, α) 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. We observe that the bounds on $\mathbb{E}[C]$ are tight for $k = 1$ and $k = n$, which can also be inferred from (18) and (19).

B. Diversity-Parallelism Trade-off

In Fig. 14 and Fig. 15 we observe that the expected latency and cost increase with k. This is because, as k increases we need to wait for more tasks out of n to complete, resulting in higher latency and cost. But in most computing and storage applications, the service time X is proportional to the size of the task, which decreases as k increases. As a result, there is a diversity-parallelism trade-off in choosing the optimal k.

We demonstrate the diversity-parallelism trade-off in Fig. 16 where service time $X \sim$ *ShiftedExp*(Δ_k, μ), with $\mu = 1.0$, and $\Delta_k = \Delta/k$. The constant Δ is proportional to the size of the job, and hence the size of each task is proportional to $\Delta_k = \Delta/k$. The latency initially reduces with k because higher diversity dominates over lack of parallelism (larger value of Δ_k). As k increases beyond threshold k^*, the loss of diversity causes an increase in latency. The optimal k^* that minimizes latency increases with Δ, because the parallelism due to $\Delta_k = \Delta/k$ dominates over the diversity advantage of having $k < n$.

We can also observe the diversity-parallelism trade-off mathemetically in the low traffic regime, for $X \sim$ *ShiftedExp*$(\Delta/k, \mu)$. If we take $\lambda \to 0$ in (14) and (13), both bounds coincide and we get,

$$\lim_{\lambda \to \infty} \mathbb{E}[T] = \mathbb{E}[X_{k:n}] = \frac{\Delta}{k} + \frac{H_n - H_{n-k}}{\mu}, \tag{20}$$

where $H_n = \sum_{i=1}^{n} 1/i$, the n^{th} harmonic number. The parallelism benefit comes from the first term in (20), which reduces with k. The diversity of waiting for k out of n tasks causes the second term to increase with k. The optimal k^* that minimizes (20) strikes a balance between these two opposing trends.

C. 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 expected latency $\mathbb{E}[T]$ with early cancellation is given by*

$$\mathbb{E}[C] = k\mathbb{E}[X] \tag{21}$$

$$\mathbb{E}[T] \leq \mathbb{E}[\max(R_1, R_2, \cdots R_k)] \tag{22}$$

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

The Laplace-Stieltjes transform of the response time R of an $M/G/1$ queue with service distribution $F_X(x)$ and arrival rate is same as (16), with λ replaced by $\lambda k/n$.

By comparing the cost $\mathbb{E}[C] = k\mathbb{E}[X]$ in (21) 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 (18) is smaller than $k\mathbb{E}[X]$. Thus we can infer that early cancellation is not effective when X is log-convex, as we also observed in Fig. 8 for the $k = 1$ case.

VII. A Heuristic Redundancy Strategy

We have seen that redundancy is an effective method to reduce latency, while efficiently using the computing resources. In Section IV and Section V, we got strong insights into the optimal redundancy strategy for log-concave and log-convex service time, but it is not obvious to infer the best strategy for arbitrary service distributions. We now propose such a heuristic 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 a closed-form expressions of F_X and its order statistics are not known.

A. Generalized Fork-join Model

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

Definition 8 ((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.*
- *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 tasks finish, cancel all remaining tasks immediately.*

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

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 and (n, k) fork-early-cancel and (n, r, k) partial-fork-join systems respectively, as defined in Section II.

B. Choosing Parameters r_f and r

We propose a heuristic strategy to choose r_f and r to minimize expected latency $\mathbb{E}[T]$, subject to a computing cost constraint is $\mathbb{E}[C] \leq \gamma$, and a network cost constraint is $r_f \leq r_{max}$. 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.

Claim 2 (Heuristic Redundancy Strategy)**.** *Good heuristic choices of r_f and r to minimize $\mathbb{E}[T]$ subject to constraints $\mathbb{E}[C] \leq \gamma$ and $r_f \leq r_{max}$ are*

$$r_f^* = r_{max}, \tag{23}$$

$$r^* = \underset{r \in [0, r_{max}]}{\arg\min}\ \hat{T}(r), \quad s.t. \quad \hat{C}(r) \leq \gamma \tag{24}$$

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:

$$\hat{T}(r) \triangleq \mathbb{E}[X_{k:r}] + \frac{\lambda r \mathbb{E}\left[X_{k:r}^2\right]}{2(n - \lambda r \mathbb{E}[X_{k:r}])}, \tag{25}$$

$$\hat{C}(r) \triangleq r\mathbb{E}[X_{k:r}]. \tag{26}$$

To justify the strategy above, observe that for a given r, increasing r_f gives higher diversity in finding the shortest queues 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. So we conjecture that it is optimal to set $r_f = r_{max}$ in (23), the maximum value possible under network cost constraints. Changing r on the other hand does affect both the computing cost and latency significantly. Thus 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 (24).

The estimates $\hat{T}(r)$ and $\hat{C}(r)$ are obtained by generalizing Lemma 2 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 are as follows:

- 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) $\hat{C}(r)$.
- 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 $|\mathbb{E}[T] - \hat{T}(r)|$ increases with k.
- The estimates $\hat{T}(r)$ and $\hat{C}(r)$ are by definition independent of r_f, which is not true in practice. As explained above, 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 $\hat{C}(r)$.

C. Simulation Results

We now present simulation results comparing the heuristic given by Claim 2 to other strategies, including baseline case without any redundancy. 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.

In Fig. 17 we plot the latency $\mathbb{E}[T]$ versus computing cost $\mathbb{E}[C]$ with service time $X \sim$ *Pareto*$(1, 2.2)$, and different redundancy strategies. Other parameters are $n = 10$, $k = 1$, and arrival rate $\lambda = 0.25$. In comparison with the no redundancy case (blue dot), the heuristic strategy (red diamond) gives a significant reduction in latency,while satisfying $\mathbb{E}[C] \leq 5$ and $r_f \leq 7$. We also plot the latency-cost behavior as $r = r_f$ varies from 1 to n. Observe that using early cancellation ($r_f > r$) in the heuristic strategy gives a slight reduction in latency in comparison with the $r = r_f = 4$ point. The cost $\mathbb{E}[C]$ increases slightly, but remains less than γ.

In Fig. 18 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 *Exp*(2) and a shifted exponental *ShiftedExp*$(1, 1.5)$, each occuring with equal probability. All other parameters are same as in Fig. 17. The heuristic strategy found using Claim 2 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.

VIII. Concluding Remarks

In this paper we consider a redundancy model where each incoming job is forked to queues at multiple servers and we wait for any one replica to finish. We analyze how redundancy affects the latency, and the cost of computing

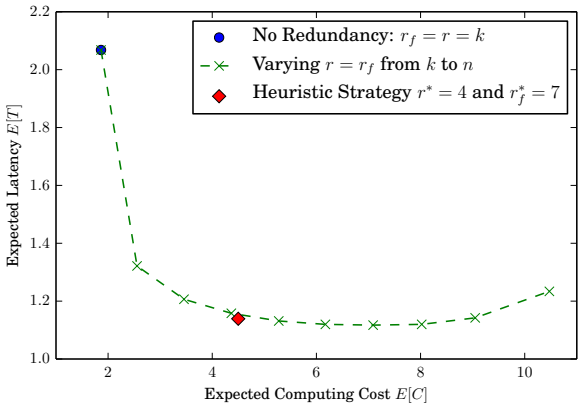


Fig. 17: Comparing the heuristic strategy with cost constraint $\gamma = 5$ and network constraint $r_{max} = 7$ to other redundancy strategies. The service time distribution is *Pareto*$(1, 2.2)$.

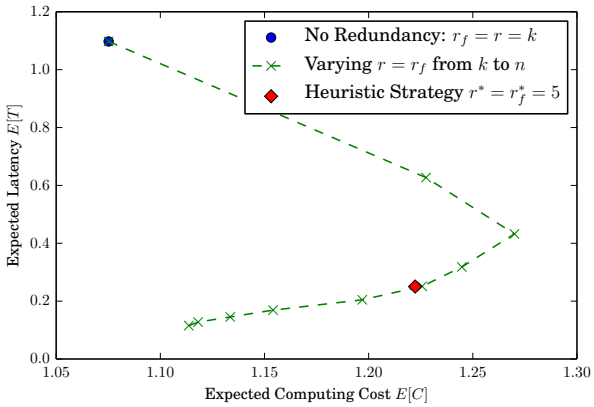


Fig. 18: Comparing the heuristic with cost constraint $\gamma = 2$ and network constraint $r_{max} = 5$ to other redundancy strategies. The service time distribution is an equiprobable mixture of *Exp*(2) and *ShiftedExp*$(1, 1.5)$.

time, and demonstrate how the log-concavity of service time is a key factor affecting the latency-cost trade-off. Some insights that we get are:

- For log-convex service time, forking to more servers (more redundancy) reduces both latency and cost. On the other hand, for log-concave service time, 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 design a heuristic redundancy strategy for an arbitrary service time distribution.

Ongoing work includes developing online strategies to simultaneously learn the service 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, for example crowdsourcing, algorithmic trading, manufacturing etc.

IX. Acknowledgements

We thank Sem Borst and Rhonda Righter for helpful suggestions to improve this work.

Appendix 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 [26].

Property 1 (Jensen's Inequality)**.** *If $\bar{F}_X$ is log-concave, then for $0 < \theta < 1$ and for all $x, y \in [0, \infty)$,*

$$\Pr(X > \theta x + (1-\theta)y) \geq \Pr(X > x)^{\theta} \Pr(X > y)^{1-\theta}. \tag{27}$$

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 (27) we get the Jensen's inequality which holds for concave functions. ■

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

$$\Pr(X > x) \leq \Pr(X > \theta x)^{1/\theta} \tag{28}$$

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

Proof: We can derive (28) by setting $y = 0$ in (27).

$$\Pr(X > \theta x + (1-\theta)0) \geq \Pr(X > x)^{\theta} \Pr(X > 0)^{1-\theta}, \tag{29}$$

$$\Pr(X > \theta x) \geq \Pr(X > x)^{\theta}. \tag{30}$$

To get (30) 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 (30) to power $1/\theta$ we get (28). The reverse inequality of log-convex $\bar{F}_X$ can be proved similarly. ∎

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

$$\Pr(X > x + t | X > t) \leq \Pr(X > x) \tag{31}$$

$$\Leftrightarrow \Pr(X > x + t) \leq \Pr(X > x) \Pr(X > t) \tag{32}$$

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

Proof:

$$\Pr(X > x) \Pr(X > t) \tag{33}$$

$$= \Pr\left(X > \frac{x}{x+t}(x+t)\right) \Pr\left(X > \frac{t}{x+t}(x+t)\right), \tag{34}$$

$$\geq \Pr(X > x + t)^{\frac{x}{x+t}} \Pr(X > x + t)^{\frac{t}{x+t}}, \tag{35}$$

where we apply Property 2 to (34) to get (35). Equation (31) follows from (35). ∎

Note that for exponential F_X which is memoryless, (31) holds with equality. Thus log-concave distributions can be thought to have 'optimistic memory', because the conditional tail probability decreases over time.

The definition of the notions 'new-better-than-used' in [19] is same as (31). By Property 3 log-concavity of $\bar{F}_X$ implies that X is new-better-than-used. New-better-than-used distributions are referred to as 'light-everywhere' in [21] and 'new-longer-than-used' in [22].

Property 4. *If X is log-concave (log-convex), $r\mathbb{E}[X_{1:r}]$ is non-decreasing (non-increasing) in r.*

Proof: Setting $\theta = r/r + 1$ in Property 2, we get

$$\Pr(X > x) \leq \Pr\left(X > \frac{xr}{r+1}\right)^{(r+1)/r}, \tag{36}$$

$$\Pr\left(X > \frac{x'}{r}\right)^{r} \leq \Pr\left(X > \frac{x'}{r+1}\right)^{r+1}, \tag{37}$$

$$\int_0^{\infty} \Pr\left(X > \frac{x'}{r}\right)^{r} dx' \leq \int_0^{\infty} \Pr\left(X > \frac{x'}{r+1}\right)^{r+1} dx', \tag{38}$$

$$r \int_0^{\infty} \Pr(X > y)^{r} dy \leq \int_0^{\infty} (r+1) \Pr(X > z)^{r+1} dz, \tag{39}$$

$$r\mathbb{E}[X_{1:r}] \leq (r+1)\mathbb{E}[X_{1:r+1}], \tag{40}$$

where in (37) we perform a change of variables to $x' = rx$. Integrating on both sides from 0 to ∞ we get (38). Again by doing change of variables $y = x'/r$ of the left-side and $z = x'/(r+1)$ on the right-side we get (39). By using the fact that the expected value of a non-negative random variable is equal to the integral of its tail distribution we get (40).

For log-convex X all the above inequalities are flipped to show that $r\mathbb{E}[X_{1:r}] \geq (r+1)\mathbb{E}[X_{1:r+1}]$.

∎

Remark 2. *If X is new-better-than-used (a weaker notion implied by log-concavity of X), it can be shown that*

$$\mathbb{E}[X] \leq r\mathbb{E}[X_{1:r}] \text{ for all } r \tag{41}$$

This is weaker than Property 4 which shows the monotonicity of $r\mathbb{E}[X_{1:r}]$ for log-concave (log-convex) X.

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 non-decreasing (non-increasing) 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. It is at most 1 for log-concave X, at least 1 for log-convex X, and equal to 1 when X is pure exponential.*

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

- *Exponential*
- *Hyper Exponential (Mixture of exponentials)*
- *Weibull with shape parameter $0 < c < 1$*
- *Gamma with shape parameter $0 < c < 1$*

APPENDIX B
PROOFS OF THE $k = 1$ CASE

Proof of Theorem 3: Using (2), we can express the cost C in terms of the relative task start times t_i, and S as follows.

$$C = S + |S - t_2|^+ + \cdots + |S - t_r|^+, \tag{42}$$

where S is the time between the start of service of the earliest task, and when any 1 of the r tasks finishes. The tail distribution of S is given by

$$\Pr(S > s) = \prod_{i=1}^{r} \Pr(X > s - t_i). \tag{43}$$

By taking expectation on both sides of (42) and simplifying we get,

$$\mathbb{E}[C] = \sum_{u=1}^{r} \int_{t_u}^{\infty} \Pr(S > s) ds, \tag{44}$$

$$= \sum_{u=1}^{r} u \int_{t_u}^{t_{u+1}} \Pr(S > s) ds, \tag{45}$$

$$= \sum_{u=1}^{r} u \int_{0}^{t_{u+1}-t_u} \Pr(S > t_u + x) dx, \tag{46}$$

$$= \sum_{u=1}^{r} u \int_{0}^{t_{u+1}-t_u} \prod_{i=1}^{u} \Pr(X > x + t_u - t_i) dx. \tag{47}$$

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 inequalities below reversed. We express the integral in (47) as,

$$\mathbb{E}[C] = \sum_{u=1}^{r} u \left(\int_0^\infty \prod_{i=1}^{u} \Pr(X > x + t_u - t_i) dx - \int_0^\infty \prod_{i=1}^{u} \Pr(X > x + t_{u+1} - t_i) dx \right), \tag{48}$$

$$= \sum_{u=1}^{r} \left(\int_0^\infty \prod_{i=1}^{u} \Pr\left(X > \frac{x'}{u} + t_u - t_i\right) dx' - \int_0^\infty \prod_{i=1}^{u} \Pr\left(X > \frac{x'}{u} + t_{u+1} - t_i\right) dx' \right), \tag{49}$$

$$= \mathbb{E}[X] + \sum_{u=2}^{r} \int_0^\infty \left(\prod_{i=1}^{u} \Pr\left(X > \frac{x'}{u} + t_u - t_i\right) - \prod_{i=1}^{u-1} \Pr\left(X > \frac{x'}{u-1} + t_u - t_i\right) \right) dx', \tag{50}$$

$$\geq \mathbb{E}[X], \tag{51}$$

where in (48) we express each integral in (47) as a difference of two integrals from 0 to ∞. In (49) we perform a change of variables $x = x'/u$. In (50) 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 (50) we use the fact that each term in the summation in (49) is positive when $\bar{F}_X$ is log-concave. This is shown in Lemma 5 below.

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 the inequalities below reversed.

$$\mathbb{E}[C] \leq \sum_{u=1}^{r} u \int_0^{t_{u+1}-t_u} \prod_{i=1}^{u} \Pr\left(X > \frac{u(x + t_u - t_i)}{r}\right)^{r/u} dx, \tag{52}$$

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

$$= \int_0^\infty \Pr\left(X > \frac{x'}{r}\right)^r dx' + \sum_{u=2}^{r} \left(\int_0^\infty \prod_{i=1}^{u} \Pr\left(X > \frac{x' + u(t_u - t_i)}{r}\right)^{r/u} dx' - \right.$$
$$\left. \int_0^\infty \prod_{i=1}^{u-1} \Pr\left(X > \frac{x' + (u-1)(t_u - t_i)}{r}\right)^{\frac{r}{u-1}} dx' \right), \tag{54}$$

$$\leq r\mathbb{E}[X_{1:r}], \tag{55}$$

where we get (52) by applying Property 2 to (47). In (53) we express the integral as a difference of two integrals from 0 to ∞, and perform a change of variables $x = x'/u$. In (54) 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 6 to show that each term in the summation in (54) is negative when $\bar{F}_X$ is log-concave. ■

Lemma 5. *If $\bar{F}_X$ is log-concave,*

$$\prod_{i=1}^{u} \Pr\left(X > \frac{x'}{u} + t_u - t_i\right) \geq \prod_{i=1}^{u-1} \Pr\left(X > \frac{x'}{u-1} + t_u - t_i\right). \tag{56}$$

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

Proof of Lemma 5: We bound the left hand side expression as follows.

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

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

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

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

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

where we use Property 3 to get (59). The inequality in (60) 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 the inequalities can be reversed. ∎

Lemma 6. *If $\bar{F}_X$ is log-concave,*

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

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

Proof of Lemma 6: We start by simplifying the left-hand side expression, raised to the power $(u-1)/r$.

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

$$= \prod_{i=1}^{u-1} \Pr\left(X > \frac{x}{r}\right)^{\frac{1}{u}} \Pr\left(X > \frac{x + u(t_u - t_i)}{r}\right)^{\frac{u-1}{u}} \tag{64}$$

$$\leq \prod_{i=1}^{u-1} \Pr\left(X > \frac{x + (u-1)(t_u - t_i)}{r}\right) \tag{65}$$

where (65) follows from the log-concavity of $\Pr(X > x)$, and the Jensen's equality. The inequality is reversed for log-convex $\bar{F}_X$. ∎

Appendix 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 the 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. So 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 [29, Chapter 5], and it reduces to the upper bound in (13).

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 1 task to finish. Then the expected waiting time in queue is equal to the second term in (13) 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 (14) on the expected latency. ∎

Proof of Lemma 3: The bound above is a generalization of the bound for the (n, n) fork-join system with exponential service time presented in [10]. To find the bound, we first observe that the response times of the n queues form a set of associated random variables [30]. 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 directly extended to the (n, k) fork-join system with $k < n$ because this property of associated variables does not hold for the k^{th} order statistic for $k < n$. ■

Proof of Lemma 4: First we derive the lower bound for the case when service time is a pure exponential with rate μ. The lower bound in (17) is a generalization of the bound for the (n, n) fork-join system derived in [11]. The bound for the (n, n) system is derived by considering that a job goes through n stages of processing. A job is said to be in the j^{th} stage if j out of n tasks have been served by their respective nodes for $0 \leq j \leq n-1$. The job waits for the remaining $n-j$ tasks to be served, after which it departs the system. For the (n, k) fork-join system, since we only need k tasks to finish service, each job now goes through k stages of processing. In the j^{th} stage, where $0 \leq j \leq k-1$, j tasks have been served and the job will depart when $k-j$ more tasks to finish service.

We now show that the service rate of a job in the j^{th} stage of processing is *at most* $(n-j)\mu$. Consider two jobs B_1 and B_2 in the i^{th} and j^{th} stages of processing respectively. Let $i > j$, that is, B_1 has completed more tasks than B_2. Job B_2 moves to the $(j+1)^{th}$ stage when one of its $n-j$ remaining tasks complete. If all these tasks are at the heads of their respective queues, the service rate for job B_2 is exactly $(n-j)\mu$. However, since $i > j$, B_1's task could be ahead of B_2's in one of the $n-j$ pending queues, due to which that task of B_2 cannot be immediately served. Hence, we have shown that the service rate of in the j^{th} stage of processing is at most $(n-j)\mu$. Thus, for pure exponential service time,

$$\mathbb{E}[T] \geq \sum_{j=0}^{k-1} \frac{1}{(n-j)\mu - \lambda} \tag{66}$$

For the shifted exponential distribution, we can find a closed-form expression for a lower bound on latency. The first term gives the time taken to serve the first of the k chunks of the file. The last term is the same as the last $k-1$ terms of the summation in (66). It is the expected sum of the residual response times, without considering the Δ shift of the distribution. ■

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 $n-k+1$ which start at the same time after the last tasks of the previous job are terminated. We consider a constraint that all the $k-1$ tasks in the first group and 1 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 tasks 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 (18).

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 1 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 (19). ■

Proof of Theorem 6: Since exactly k tasks are served, and others are cancelled 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 each job has k tasks that 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. But since there are no redundant tasks and the k queues are chosen at random, the expected latency on this (n,k,k) partial-fork-join system is larger than that on the (n,k) fork-early-cancel system.

Now let us upper bound the latency $\mathbb{E}\left[T^{(pf)}\right]$ of the partial fork system. Each queue has arrival rate $\lambda k/n$, and service distribution F_X. Using the approach in [10] 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,

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

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

The expected maximum can be numerically evaluated from distribution of R. From the transform analysis given in [28, Chapter 25], we know that the Laplace-Stieltjes transform $R(s)$ of the probability density of R is same as (16), but with λ replaced by $\lambda k/n$. ■

REFERENCES

[1] J. Dean and L. Barroso, "The Tail at Scale," *Communications of the ACM*, vol. 56, no. 2, pp. 74–80, 2013.

[2] G. Kabatiansky, K. E., and S. S., *Error correcting coding and security for data networks: analysis of the superchannel concept*, ch. 7. Wiley, 1st ed., Mar. 2005.

[3] A. Vulimiri, P. B. Godfrey, R. Mittal, J. Sherry, S. Ratnasamy, and S. Shenker, "Low latency via redundancy," in *ACM Conference on Emerging Networking Experiments and Technologies*, CoNEXT '13, pp. 283–294, ACM, 2013.

[4] J. Dean and S. Ghemawat, "MapReduce: simplified data processing on large clusters," *ACM Commun. Mag.*, vol. 51, pp. 107–113, Jan. 2008.

[5] G. Ananthanarayanan, A. Ghodsi, S. Shenker, and I. Stoica, "Effective straggler mitigation: Attack of the clones," in *USENIX Conference on Networked Systems Design and Implementation*, pp. 185–198, 2013.

[6] K. Ousterhout, P. Wendell, M. Zaharia, and I. Stoica, "Sparrow: Distributed, low latency scheduling," in *ACM Symposium on Operating Systems Principles (SOSP)*, pp. 69–84, 2013.

[7] G. Joshi, Y. Liu, and E. Soljanin, "Coding for fast content download," *Allerton Conf. on Communication, Control and Computing*, pp. 326–333, Oct. 2012.

[8] G. Joshi, Y. Liu, and E. Soljanin, "On the Delay-storage Trade-off in Content Download from Coded Distributed Storage," *IEEE Journal on Selected Areas on Communications*, May 2014.

[9] L. Flatto and S. Hahn, "Two parallel queues created by arrivals with two demands I," *SIAM Journal on Applied Mathematics*, vol. 44, no. 5, pp. 1041–1053, 1984.

[10] R. Nelson and A. Tantawi, "Approximate analysis of fork/join synchronization in parallel queues," vol. 37, pp. 739–743, Jun. 1988.

[11] E. Varki, A. Merchant, and H. Chen, "The M/M/1 fork-join queue with variable sub-tasks," *unpublished, available online.*

[12] N. Shah, K. Lee, and K. Ramachandran, "The mds queue: Analyzing the latency performance of erasure codes," *IEEE International Symposium on Information Theory*, July 2014.

[13] K. Gardner, S. Zbarsky, S. Doroudi, M. Harchol-Balter, E. Hyytiä, and A. Scheller-Wolf, "Reducing latency via redundant requests: Exact analysis," in *ACM SIGMETRICS*, Jun. 2015.

[14] A. Kumar, R. Tandon, and T. C. Clancy, "On the latency of heterogeneous mds queue," in *IEEE Global Communications Conference (GLOBECOM)*, pp. 2375–2380, Dec 2014.

[15] Y. Xiang, T. Lan, V. Aggarwal, and Y. F. R. Chen, "Joint latency and cost optimization for erasure-coded data center storage," *SIGMETRICS Performance Evaluation Review*, vol. 42, pp. 3–14, Sept. 2014.

[16] G. Liang and U. Kozat, "TOFEC: Achieving Optimal Throughput-Delay Trade-off of Cloud Storage Using Erasure Codes," *IEEE International Conference on Communications*, Apr. 2014.

[17] S. Chen, U. C. Kozat, L. Huang, P. Sinha, G. Liang, X. Liu, Y. Sun, and N. B. Shroff, "When Queueing Meets Coding: Optimal-Latency Data Retrieving Scheme in Storage Clouds," *IEEE International Conference on Communications*, Apr. 2014.

[18] J. Cao and Y. Wang, "The nbuc and nwuc classes of life distributions," *Journal of Applied Probability*, pp. 473–479, 1991.

[19] G. Koole and R. Righter, "Resource allocation in grid computing," *Journal of Scheduling*, vol. 11, pp. 163–173, June 2008.

[20] Y. Kim, R. Righter, and R. Wolff, "Job replication on multiserver systems," *Advances in Applied Probability*, vol. 41, pp. pp. 546–575, June 2009.

[21] N. Shah, K. Lee, and K. Ramchandran, "When do redundant requests reduce latency?," in *Allerton Conf. on Comm., Control and Computing*, Oct. 2013.

[22] Y. Sun, Z. Zheng, C. E. Koksal, K. Kim, and N. B. Shroff, "Provably delay efficient data retrieving in storage clouds," in *IEEE Conference on Computer Communications (INFOCOM)*, Apr. 2015.

[23] D. Wang, G. Joshi, and G. Wornell, "Efficient task replication for fast response times in parallel computation," *ACM Sigmetrics short paper*, June 2014.

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

[25] G. Joshi, E. Soljanin, and G. Wornell, "Queues with redundancy: Latency-cost analysis," in *ACM SIGMETRICS Workshop on Mathematical Modeling and Analysis*, jun 2015.

[26] M. Bagnoli and T. Bergstrom, "Log-concave probability and its applications," *Economic Theory*, vol. 26, no. 2, pp. pp. 445–469, 2005.

[27] A. M. Lee and P. A. Loughton, "Queueing process associated with airline passenger check-in," *Operations Research Quarterly*, pp. 56–71, 1957.
[28] M. Harchol-Balter, *Performance Modeling and Design of Computer Systems: Queueing Theory in Action*. Cambridge University Press, 2013.
[29] R. Gallager, *Stochastic Processes: Theory for Applications*. Cambridge University Press, 1st ed., 2013.
[30] J. Esary, F. Proschan, and D. Walkup, "Association of random variables, with applications," *Annals of Mathematics and Statistics*, vol. 38, pp. 1466–1474, Oct. 1967.