Efficient Replication of Queued Tasks to Reduce Latency in Cloud Systems

Gauri Joshi
EECS Dept., MIT
Cambridge, MA 02139, USA
Email: gauri@mit.edu

Emina Soljanin
Bell Labs, Alcatel-Lucent
Murray Hill NJ 07974, USA
Email: emina@bell-labs.com

Gregory Wornell
EECS Dept., MIT
Cambridge MA 02139, USA
Email: gww@mit.edu

***Abstract*—In cloud computing systems, assigning a job 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. Although adding redundant replicas always reduces service time, the total computing time spent per job may be higher, thus increasing waiting time in queue. The total time spent per job is also proportional to the cost of computing resources. We analyze how different redundancy strategies, for eg. number of replicas, and the time when they are issued and canceled, affect the latency and computing cost. We get the insight that the *log-concavity* of the service time distribution is a key factor in determining whether adding redundancy reduces latency and cost. If the service distribution is log-convex, then adding maximum redundancy reduces both latency and cost. And if it is log-concave, then having fewer replicas and canceling the redundant requests early is more effective.**

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, executing a task on multiple machines and waiting of one 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 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 [2] to send packets in networks. See [3, Chapter 7] for a detailed survey of other related work. A similar idea has also been recently studied in [4]. In large-scale cloud computing frameworks such as MapReduce [5], the slowest tasks of a job (stragglers) become a bottleneck in its completion. Several recent works in systems such as [6], [7] 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. Thus download latency can be reduced by forking each request to all n servers and waiting for any k to respond. In [8], [9] 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 [10], [11]. In recent years, there is a renewed interest in fork-join queues due to their application to distribution 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]. Most recently, [13] presents an analysis of latency with heterogeneous job classes for the replicated ($k = 1$) case with exponential service time.

General Service Time: Few practical systems have exponentially distributed service time. For example, studies of download time traces from Amazon S3 [14], [15] 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' [16], it is shown in [17] that it is optimal to fork a job to maximum number of servers. The

TABLE I: Optimal redundancy strategies when the service time is log-concave or log-convex. 'Canceling redundancy early' means that we cancel redundant tasks when any 1 task reaches the head of its queue, instead of waiting for it to be served.

	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)	$r = 1$	$r = n$	$r = n$

choice of scheduling policy for new-worse-than-used (NWU) and new-better-than-used (NBU) distributions is also studied in [18]–[20]. The NBU and NWU notions are closely related to the log-concavity of service time studied in this work.

The Cost of Redundancy: If the service time is assumed to be exponential, then adding redundancy does not cause any increase in cost of computing time. But since the exponential assumption does not generally hold 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 is considered in [19]. The total server time spent on each job is considered in [21], [22] for a distributed system without queueing of requests. In [23] 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, and also the computing cost (total server time spent per job). Incidentally, our computing cost metric serves as a powerful tool to compare different redundancy strategies in the high traffic regime.

The analysis gives the insight that the log-concavity (log-convexity) of the tail distribution $\bar{F}_X$ of service time is a key factor in determining when redundancy helps. Here are some instances, that are also summarized in Table I. For example, a redundancy strategy is to fork each job to queues at n servers, and wait for any one replica to finish. An alternate strategy is to cancel the redundant replicas as soon as any one reaches the head of its queue. 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 $\bar{F}_X$. In another instance, suppose we fork each job to only a subset r out of the n servers. Then we can show that forking to more servers (larger r) is always better 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.

II. Problem Formulation

A. Fork-Join Model and its Variants

Consider a distributed system with n statistically identical servers. We define the $(n, 1)$ fork-join system as follows.

Definition 1 ($(n, 1)$ fork-join system)**.** *Each incoming job is forked into n tasks that join first-come first-serve queues at the n servers. When any one task is served, all remaining tasks are canceled and abandon their queues immediately.*

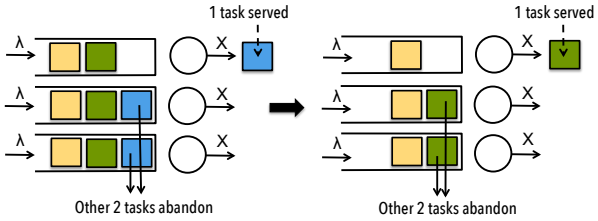


Fig. 1: The $(3, 1)$ fork-join system. When any 1 out of 3 tasks of a job is served, the remaining 2 tasks abandon their queues immediately.

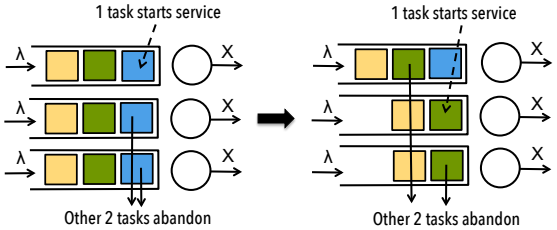


Fig. 2: The $(3, 1)$ fork-early cancel system. When any 1 out of 3 tasks of a job starts service, the others abandon their queues.

This is a special case of the (n, k) fork-join system considered in [8], [9] where any k out of n tasks are sufficient to complete the job. General $k > 1$ arise in approximate computing, or in content download from erasure coded distributed storage. Fig. 1 illustrates the $(3, 1)$ fork-join system.

Instead of waiting for any one task to finish, we could cancel the redundant tasks early, when any task starts service. A similar idea has been proposed in systems work [7]. We refer to this variant as the $(n, 1)$ fork-early cancel system defined formally as follows.

Definition 2 ($(n, 1)$ fork-early cancel system)**.** *Each incoming job is forked to the n servers. When any task starts service, we cancel the redundant tasks immediately. If more than one tasks start service simultaneously, we preserve any one task chosen uniformly at random.*

Fig. 2 illustrates the $(3, 1)$ fork-early cancel system. Early cancellation can save the total time spent per job (computing

cost), but could result in an increase in latency because of loss of diversity due to cancelling tasks earlier. In Section IV we compare the $(n, 1)$ fork-join and the $(n, 1)$ fork-early-cancel systems.

Due to the network cost of issuing and canceling tasks, it may be prohibitively expensive to fork a job to all n servers. Thus, we consider a partial forking variant defined as follows.

Definition 3 ($(n, r, 1)$ partial fork-join system)**.** *Each incoming job is forked into r out of the n servers. When any one task is served, 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 total computing time as well as the network cost, which is proportional to the number of servers each job is forked to. In Section V 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 considered here, our analysis techniques can be extended to these variants.

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). The results for 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 virtualization, disk seek time, server outages, preemption by other jobs etc. We model this task service time by the random variable $X > 0$, with cumulative distribution function $F_X(x)$ and assume that it is i.i.d. across requests and servers. Dependence of the service time on the job size can be modeled by adding a constant to X. For example, some recent work [14], [15] 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 $\bar{F}_X(x)$ to denote the tail probability function $\Pr(X > x)$ of X. We use $X_{1:n}$ to denote the smallest of n i.i.d. random variables $X_1, X_2, \ldots X_n$.

C. 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 paper.

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

Definition 5 (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.*

If a task is canceled before it reaches the head of its queue, the cost incurred at that server is zero. 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 not analyzed explicitly in this paper, we note that there is also a network cost of issuing and canceling the redundant tasks, proportional to n for the $(n, 1)$ fork-join and fork-early-cancel system, and r in the $(n, r, 1)$ partial-fork-join system.

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

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

Definition 6 (Symmetric Policy)**.** *In a symmetric policy, the tasks of each job are forked one or more 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. In Claim 1, we express the service capacity in terms on $\mathbb{E}[C]$. Corollary 1 then follows because higher service capacity implies lower latency in the high load regime.

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

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

of Claim 1. For a symmetric policy, the mean time spent by each server per job is $\mathbb{E}[C]/n$. Thus the server utilization is $\rho = \lambda \mathbb{E}[C]/n$. 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. □

Corollary 1. *The symmetric redundancy strategy that results in a lower $\mathbb{E}[C]$, also gives lower $\mathbb{E}[T]$ in the high traffic regime, when λ is close to the service capacity.*

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 clear insights into how redundancy affects latency and cost. 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 $\bar{F}_X$ 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). An interesting implication of log-concavity is that if $\bar{F}_X$ is log-concave,

$$\Pr(X > x + t | X > t) \leq \Pr(X > x) \tag{2}$$

The inequality is reversed if $\bar{F}_X$ is log-convex[1]. Equality holds for the exponential distribution, which is both log-convex and log-concave. As a result the mean residual life $\mathbb{E}[X - t | X > t]$ decreases (increases) with the elapsed time t if $\bar{F}_X$ is log-concave (log-convex).

The numerical results in this paper use the shifted exponential, and hyper exponential as examples of log-concave and log-convex distributions respectively. The shifted exponential, denoted by *ShiftedExp*(Δ, μ) is an exponential with rate μ, plus a constant shift $\Delta \geq 0$. 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.

If we fork a job to all r idle servers and wait for any 1 copy to finish, the expected computing cost $\mathbb{E}[C] = r\mathbb{E}[X_{1:r}]$. Lemma 1 below gives how $r\mathbb{E}[X_{1:r}]$ varies with r for log-concave (log-convex) $\bar{F}_X$. It is central to proving several key results in this paper.

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

The proof of Lemma 1 is omitted here and can be found in the extended version [24]. We refer readers to [25] for other properties and examples of log-concave distributions.

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 task start times 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 $t_3 = 7 - 3 = 4$ respectively. 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 one task finishes. Thus it is minimum 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 tail distribution of S is given by

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

The computing cost C is given by,

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

[1]The definition of the notion 'new-better-than-used' considered in [17] is same as (2). Other names used to refer to new-better-than-used distributions are 'light-everywhere' in [19] and 'new-longer-than-used' in [20].

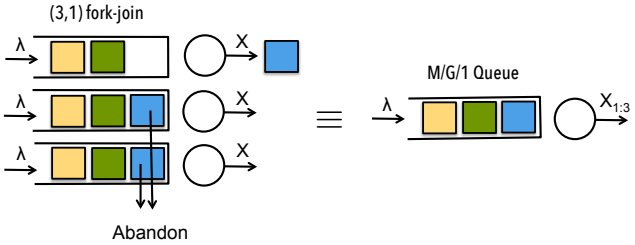


Fig. 3: Equivalence of the $(n, 1)$ fork-join system with an $M/G/1$ queue with service time $X_{1:n}$, the minimum of n i.i.d. random variables $X_1, X_2, \ldots, X_n$.

The relative task start times t_i affect C in two opposing ways. The negative part of each term in (4) increases with t_i, but the expected value of S increases (3) with t_i. By analyzing (4) 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. $(n, 1)$ System with and without 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 2. *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 $X_{1:n}$. □

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

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

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

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

Proof. By Lemma 2, 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 (5). The

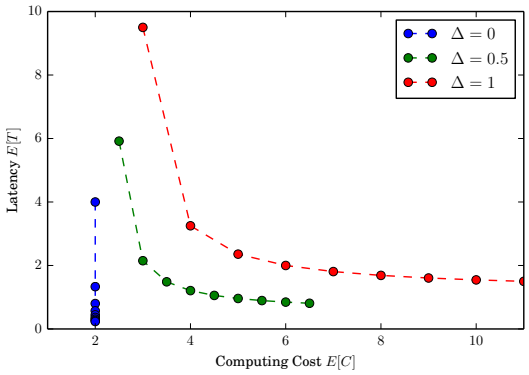


Fig. 4: The service time $X \sim \Delta{+}\text{Exp}(\mu)$ (log-concave), with $\mu = 0.5$, $\lambda = 0.25$. As n increases along each curve, $\mathbb{E}[T]$ decreases and $\mathbb{E}[C]$ increases. Only when $\Delta = 0$, latency reduces at no additional cost.

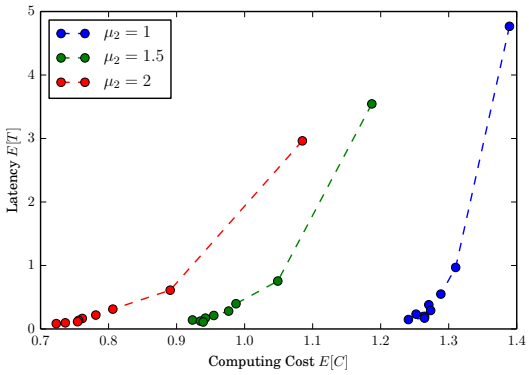


Fig. 5: The service time $X \sim \textit{HyperExp}(0.4, \mu_1, \mu_2)$ (log-convex), with $\mu_1 = 0.5$, different values of μ_2, and $\lambda = 0.5$. Expected latency and cost both reduce as n increases along each curve.

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 (4) . □

In Corollary 2 and Corollary 3 we characterize how $\mathbb{E}[T]$ and $\mathbb{E}[C]$ vary with n. The behavior of $\mathbb{E}[C]$ follows from Lemma 1.

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.*

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 $\textit{ShiftedExp}(\Delta, 0.5)$, 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 $\textit{HyperExp}(0.4, 0.5, \mu_2)$ 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 $\bar{F}_X$.

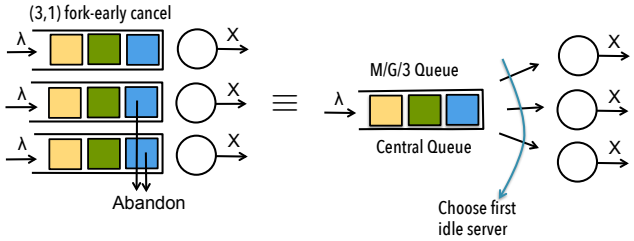


Fig. 6: Equivalence of the $(n, 1)$ fork-early cancel system to an $M/G/n$ queue with each server taking time $X \sim F_X$ to serve task, i.i.d. across servers and tasks.

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

$$\mathbb{E}[T] = \mathbb{E}\left[T^{M/G/n}\right], \tag{7}$$

$$\mathbb{E}[C] = \mathbb{E}[X], \tag{8}$$

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 [26] is,

$$\mathbb{E}\left[T^{M/G/n}\right] \approx \mathbb{E}[X] + \frac{\mathbb{E}\left[X^2\right]}{2\mathbb{E}[X]^2}\mathbb{E}\left[W^{M/M/n}\right] \tag{9}$$

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 [27, Chapter 14].

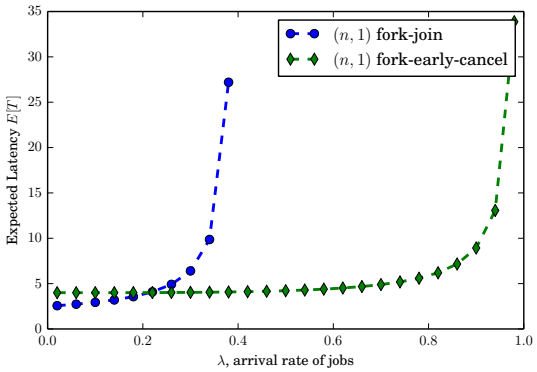


Fig. 7: For the $(4,1)$ system with service time $X \sim$ *ShiftedExp*$(2, 0.5)$ which is log-concave, early cancellation is better in the high λ regime, as given by Corollary 5.

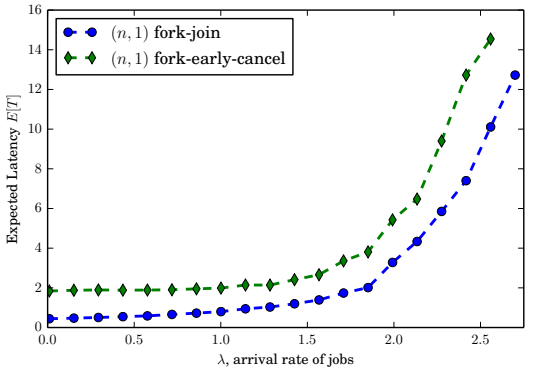


Fig. 8: For the $(4,1)$ system with $X \sim$ *HyperExp*$(0.1, 1.5, 0.5)$, which is log-convex, early cancellation is worse in both low and high λ regimes, as given by Corollary 5.

We now compare the latency and cost with and without early cancellation given by Theorem 2 and Theorem 1. Corollary 4 below follows from Lemma 1.

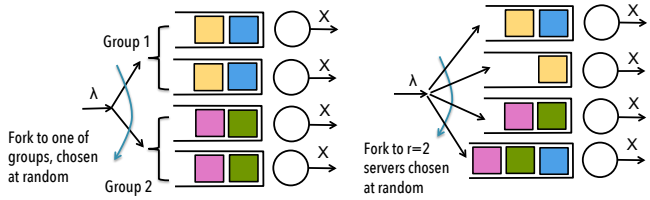
Corollary 4. *If $\bar{F}_X$ is log-concave (log-convex), then $\mathbb{E}[C]$ of the $(n,1)$ fork-early-cancel system is greater than equal to (less than or equal to) that of $(n,1)$ fork-join system.*

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, the high λ regime, the system with lower $\mathbb{E}[C]$ has lower expected latency.

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 random (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 part 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, 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 and the diversity benefit of having redundant tasks is smaller.



(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 policies.

V. $(n,r,1)$ PARTIAL-FORK-JOIN SYSTEM

We now analyze the latency-cost trade-off for 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 policy (Definition 6). Some examples of symmetric 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 policies. In the sequel, we develop insights into the best r and the choice of servers for a given service distribution F_X.

Remark 1 (Relation to Power-of-r Scheduling)**.** *In the $(n,r,1)$ partial-fork-join system with uniform random policy, we assign the job to r randomly chosen queues, and wait for the earliest copy to finish. Instead, in the power-of-r scheduling, a job is assigned to the shortest of r randomly chosen queues. Power-of-r thought of as early cancellation of $r-1$ out of r tasks, even before they join the queues.*

A. Latency-Cost Analysis

In the group-based random policy, each group behaves as an $(r,1)$ fork-join system, the r tasks of a job starting service simultaneously. Thus, the expected latency and cost follow from Theorem 1 as given in Lemma 3 below.

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*

$$\mathbb{E}[T] = \mathbb{E}[X_{1:r}] + \frac{\lambda r \mathbb{E}[X_{1:r}^2]}{2(n - \lambda r \mathbb{E}[X_{1:r}])} \quad (10)$$

$$\mathbb{E}[C] = r\mathbb{E}[X_{1:r}] \quad (11)$$

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

$$\lambda_{max} = \frac{n}{r\mathbb{E}[X_{1:r}]} \quad (12)$$

From (12) 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 Lemma 1, the optimal r is $r = 1$ ($r = n$) for log-concave (log-convex) $\bar{F}_X$.

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 below.

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.*

$$r\mathbb{E}[X_{1:r}] \geq \mathbb{E}[C] \geq \mathbb{E}[X] \quad \text{if } \bar{F}_X \text{ is log-concave} \quad (13)$$

$$\mathbb{E}[X] \geq \mathbb{E}[C] \geq r\mathbb{E}[X_{1:r}] \quad \text{if } \bar{F}_X \text{ is log-convex} \quad (14)$$

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 (4), and show that for log-concave and log-convex $\bar{F}_X$, we get the bounds in (13) and (14), which are independent of the relative task start times t_i. The detailed proof is omitted here, but can be found in the extended version [24].

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. Optimal value of r

By Lemma 1, $r\mathbb{E}[X_{1:r}]$ is non-decreasing (non-increasing) with r for log-concave (log-convex) $\bar{F}_X$. By this fact and Theorem 3, we get the following corollaries about how $\mathbb{E}[C]$ and $\mathbb{E}[T]$ vary with r.

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).*

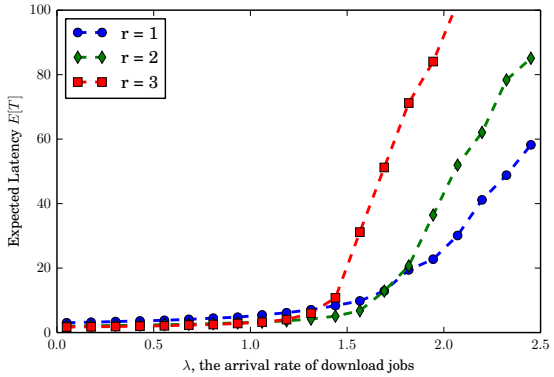


Fig. 10: For $X \sim$ *ShiftedExp*$(1, 0.5)$ which is log-concave, forking to less (more) servers reduces expected latency in the low (high) λ regime.

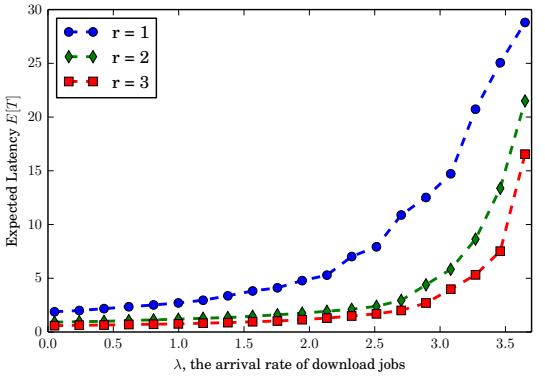


Fig. 11: For $X \sim$ *HyperExp*(p, μ_1, μ_2) with $p = 0.1$, $\mu_1 = 1.5$, and $\mu_2 = 0.5$ which is log-convex, forking to more servers (larger r) gives lower expected latency for all λ.

Corollary 7 (Expected Latency vs. r)**.** *In the low-traffic regime, forking to all servers ($r = n$) gives the lowest $\mathbb{E}[T]$ in the low λ regime for any service time distribution F_X. In the high traffic regime, $r = 1$ ($r = n$) gives lowest $\mathbb{E}[T]$ if $\bar{F}_X$ is log-concave (log-convex).*

Corollary 7 is 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 7 was previously proven for new-better-than-used (new-worse-than-used) instead of log-concave (log-convex) $\bar{F}_X$ in [17], [19], using a combinatorial argument.

Using Theorem 3, we get an alternative, and arguably simpler way to prove this result. Note that our version is slightly weaker because log-concavity implies new-better-than-used but the converse is not true in general.

C. 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. If all the r tasks start at the same time (true for the group-based random policy), $\mathbb{E}[C] = r\mathbb{E}[X_{1:r}]$. By comparing with the bounds in Theorem 3 that hold for any relative task start times we get the following result.

Corollary 8 (Cost for 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 some i.*

Corollary 9 (Latency in 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 high traffic regime than one that results in $0 < t_i < \infty$ for some i.*

For example, lets us compare the group-based random and uniform random policies. 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, that uniform random policy results lower latency for log-concave $\bar{F}_X$. But for log-convex $\bar{F}_X$, group-based forking is better in the high λ regime.

VI. Concluding Remarks

We consider a redundancy model where a computing job is replicated at multiple servers, and we wait for any one copy to finish and cancel the rest. 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 in determining the best redundancy strategy. For example, if the service time is log-convex, adding maximum redundancy reduces both latency and cost. For log-concave service time, can reduce latency, but increases the cost of computing time. Thus, adding fewer replicas, and canceling redundant tasks early is more effective, especially in the high traffic regime.

Using these insights, in [24], we propose a general redundancy strategy for an arbitrary service time distribution, that may be neither log-concave nor log-convex. 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.

VII. Acknowledgments

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

References

[1] J. Dean and L. Barroso, "The Tail at Scale," *Communications of the ACM*, vol. 56, no. 2, pp. 74–80, 2013.
[2] N. F. Maxemchuk, "Dispersity routing," pp. 41.10–41.13, Jun. 1975.
[3] G. Kabatiansky, E. Krouk and S. Semenov, *Error correcting coding and security for data networks: analysis of the superchannel concept*, ch. 7. Wiley, 1st ed., Mar. 2005.
[4] A. Vulimiri, P. B. Godfrey, R. Mittal, J. Sherry, S. Ratnasamy, and S. Shenker, "Low latency via redundancy," in *ACM CoNEXT*, pp. 283–294, 2013.
[5] J. Dean and S. Ghemawat, "MapReduce: simplified data processing on large clusters," *ACM Commun. Mag.*, vol. 51, pp. 107–113, Jan. 2008.
[6] G. Ananthanarayanan, A. Ghodsi, and I. S. S. Shenker, "Effective straggler mitigation: Attack of the clones," in *USENIX Conference on Networked Systems Design and Implementation*, pp. 185–198, 2013.
[7] 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.
[8] G. Joshi, Y. Liu, and E. Soljanin, "Coding for fast content download," *Allerton Conf. on Communication, Control and Computing*, pp. 326–333, Oct. 2012.
[9] 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.
[10] 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.
[11] R. Nelson and A. Tantawi, "Approximate analysis of fork/join synchronization in parallel queues," vol. 37, pp. 739–743, Jun. 1988.
[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] 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.
[15] 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.
[16] J. Cao and Y. Wang, "The nbuc and nwuc classes of life distributions," *Journal of Applied Probability*, pp. 473–479, 1991.
[17] G. Koole and R. Righter, "Resource allocation in grid computing," *Journal of Scheduling*, vol. 11, pp. 163–173, June 2008.
[18] Y. Kim, R. Righter, and R. Wolff, "Job replication on multiserver systems," *Advances in Applied Probability*, vol. 41, pp. 546–575, June 2009.
[19] N. Shah, K. Lee, and K. Ramchandran, "When do redundant requests reduce latency?," in *Allerton Conf. on Comm., Control and Computing*, Oct. 2013.
[20] 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.
[21] D. Wang, G. Joshi, and G. Wornell, "Efficient task replication for fast response times in parallel computation," *ACM SIGMETRICS short paper*, June 2014.
[22] 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.
[23] G. Joshi, E. Soljanin, and G. Wornell, "Queues with redundancy: Latency-cost analysis," *ACM SIGMETRICS Workshop on Mathematical Modeling and Analysis*, jun 2015.
[24] G. Joshi, E. Soljanin, and G. Wornell, "Efficient redundancy techniques for latency reduction in cloud systems," *arXiv:1508.03599 [cs.dc]*, Aug. 2015.
[25] M. Bagnoli and T. Bergstrom, "Log-concave probability and its applications," *Economic Theory*, vol. 26, no. 2, pp. pp. 445–469, 2005.
[26] A. M. Lee, and P. A. Loughton, "Queueing process associated with airline passenger check-in," *Operations Research Quarterly*, pp. 56–71, 1957.
[27] M. Harchol-Balter, *Performance Modeling and Design of Computer Systems: Queueing Theory in Action*. Cambridge University Press, 2013.