

On the Delay-Storage Trade-off in Content ¹ Download from Coded Distributed Storage Systems

Gauri Joshi, Yanpei Liu, *Student Member, IEEE*, Emina Soljanin, *Senior Member, IEEE*

Abstract

In this paper we study how coding in distributed storage reduces expected download time, in addition to providing reliability against disk failures. The expected download time is reduced because when a content file is encoded to add redundancy and distributed across multiple disks, reading only a subset of the disks is sufficient to reconstruct the content. For the same total storage used, coding exploits the diversity in storage better than simple replication, and hence gives faster download. We use a novel fork-join queuing framework to model multiple users requesting the content simultaneously, and derive bounds on the expected download time. Our system model and results are a novel generalization of the fork-join system that is studied in queueing theory literature. Our results demonstrate the fundamental trade-off between the expected download time and the amount of storage space. This trade-off can be used for design of the amount of redundancy required to meet the delay constraints on content delivery.

Index Terms

distributed storage, fork-join queues, MDS codes

This work was presented in part at the 50th Annual Allerton Conference on Communication, Control and Computing, Monticello IL, October 2012.

G. Joshi is with the Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology, Cambridge, MA. Y. Liu is with the Department of Electrical and Computer Engineering, University of Wisconsin Madison, Madison, WI. E. Soljanin is with Bell Labs, Alcatel-Lucent, Murray Hill, NJ. (E-mail: gauri@mit.edu, yliu73@wisc.edu, emina@research.bell-labs.com)

I. INTRODUCTION

Large-scale cloud storage and distributed file systems such as Amazon Elastic Block Store (EBS) [1] and Google File System (GoogleFS) [2] have become the backbone of many applications such as web searching, e-commerce, and cluster computing. In these distributed storage systems, the content files stored on a set of disks may be simultaneously requested by multiple users. The users have two major demands: reliable storage and fast content download. Content download time includes the time taken for a user to compete with the other users for access to the disks, and the time to acquire the data from the disks. Fast content download is important for delay-sensitive applications such as video streaming, VoIP, as well as collaborative tools like Dropbox [3] and Google Docs [4].

The authors in [2] point out that in large-scale distributed storage systems, disk failures are the norm and not the exception. To protect the data from disk failures, cloud storage providers today simply replicate content throughout the storage network over multiple disks. In addition to fault tolerance, replication makes the content quickly accessible since multiple users requesting a content can be directed to different replicas. However, replication consumes a large amount of storage space. In data centers that process massive data today, using more storage space implies higher expenditure on electricity, maintenance and repair, as well as the cost of leasing physical space.

Coding, which was originally developed for reliable communication in presence of noise, offers a more efficient way to store data in distributed systems. The main idea behind coding is to add redundancy so that a content, stored on a set of disks, can be reconstructed by reading a subset of these disks. Previous work shows that coding can achieve the same reliability against failures with lower storage space used. It also allows efficient replacement of disks that have to be removed due to failure or maintenance. We show that in addition to reliability and easy repair, coding also gives faster content download because we only have to wait for content download from a subset of the disks. Some preliminary results on the analysis of download time via queueing-theoretic modeling are presented in [5].

A. Previous Work

Research in coding for distributed storage was galvanized by the results reported in [6]. Prior to that work, literature on distributed storage recognized that, when compared with replication, coding can offer huge storage savings for the same reliability levels. But it was also argued that the benefits of coding are limited, and are outweighed by certain disadvantages and extra complexity. Namely, to provide reliability in multi-disk storage systems, when some disks fail, it must be possible to restore either the exact lost data or an equivalent reliability with minimal download from the remaining storage. This problem of efficient recovery from disk failures was addressed in some early work [7]. But in general, the cost of repair regeneration was considered much higher in coded than in replication systems [8], until [6] established existence and advantages of new regenerating codes. This work was then quickly followed, and the area is very active today (see e.g., [9]–[11] and references therein).

Only recently [12]–[14] was it realized that in addition to reliability, coding can guarantee the same level of content accessibility, but with lower storage than replication. In [12], the scenario that when there are multiple requests, all except one of them are blocked and the accessibility is measured in terms of blocking probability is considered. In [13], multiple requests are placed in a queue instead of blocking and the authors propose a scheduling scheme to map requests to servers (or disks) to minimize the waiting time. In [14], the authors give a combinatorial proof that flooding requests to all disks, instead of a subset of them gives the fastest download time. This result corroborates the system model we consider in this paper to model the distributed storage system and analyze its download time.

Using redundancy in coding for delay reduction has also been studied in the context of packet transmission in [15]–[17], and for some content retrieval scenarios in [18], [19]. Although they share some common spirit, they do not consider the effect of queueing of requests in coded distributed storage systems.

B. Our Contributions

In this paper we show that coding allows fast content download in addition to reliable storage. Since multiple users can simultaneously request the content, the download time includes the time to wait for access to the disks plus the time to read the data. When the content is coded and

distributed on multiple disks, it is sufficient to read it only from a subset of these disks in order to retrieve the content. We take a queuing-theoretic approach to study how coding the content in this way provides diversity in storage, and achieve a significant reduction in the download time. The analysis of download time leads us to an interesting trade-off between download time and storage space, which can be used to design the optimal level of redundancy in a distributed storage system. To the best of our knowledge, we are the first to propose the (n, k) fork-join system and find bounds on its mean response time, a novel generalization of the (n, n) fork-join system studied in queueing theory literature.

We consider that requests entering the system are assigned to multiple disks, where they enter local queues waiting for disks access. Note that this is in contrast to some existing works (e.g. [13], [14]) where requests wait in a centralized queue when all disks are busy. Our approach of immediate dispatching of requests to local queues is used by most server farms to facilitate fast acknowledgement response to customers [20]. Under this queueing model, we propose the (n, k) fork-join system, where each request is forked to n disks that store the coded content, and it exits the system when any k ($k \leq n$) disks are read. The (n, n) fork-join system in which all n disks have to be read has been extensively studied in queueing theory and operations research related literature [21]–[23]. Our analysis of download time can be seen as a generalization to the analysis of the (n, n) fork-join system.

The rest of the paper is organized as follows. In Section II, we present some preliminary concepts that are central to the results presented in the paper. In Section III, we analyze the expected download time of the (n, k) fork-join system and present the fundamental trade-off between expected download time and storage. These results were presented in part in [5]. In Section IV, we relax some simplifying assumptions, and present the delay-storage trade-off by considering some practical issues such as heavy-tailed and correlated service times of the disks. In Section V, we extend the analysis to distributed storage systems with a large number of disks. Such systems can be divided into groups of n disks each, where each group is an independent (n, k) fork-join system. Finally, Section VI concludes the paper and gives future research directions.

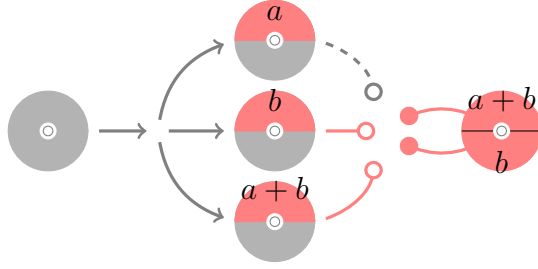


Fig. 1: Storage is 50% higher, but response time (per disk & overall) is reduced.

II. PRELIMINARY CONCEPTS

A. Reducing Delay using Coding

One natural way to reduce the download time is to replicate the content across n disks. Then if the user issues n download requests, one to each disk, it only needs to wait for the one of the requests to be served. This strategy gives a sharp reduction in download time, but at the cost of n times more storage space and the cost of processing multiple requests.

It is more efficient to use coding instead of replication. Consider that a content F of unit size is divided into k blocks of equal size. It is encoded to $n \geq k$ blocks using an (n, k) maximum distance separable (MDS) code, and the coded blocks are stored on an array of n disks. MDS codes have the property that any k out of the n blocks are sufficient to reconstruct the entire file. MDS codes have been suggested to provide reliability against disk failures. In this paper we show that, in addition to error-correction, we can exploit these codes to reduce the download time of the content.

The encoded blocks are stored on n different disks (one block per disk). Each incoming request is sent to all n disks, and the content can be recovered when any k out of n blocks are successfully downloaded. An illustrative example with $n = 3$ disks and $k = 2$ is shown in Fig. 1. The content F is split into equal blocks a and b , and stored on 3 disks as a , b , and $a \oplus b$, the exclusive-or of blocks a and b . Thus each disk stores content of half the size of file F . Downloads from any 2 disks jointly enable reconstruction of F .

B. Role of Order Statistics

The time taken to download a block of content is a random variable. If the block download times are independent and identically distributed (i.i.d.), the time to download any k out of n blocks is the k^{th} order statistic of the block download times. We now provide some background on order statistics of i.i.d. random variables. For a more complete treatment, please refer to [24]. Although in our system model the block download times are not i.i.d., this background on i.i.d. order statistics is a powerful tool for our analysis on the dependent case as shown in later sections.

Let X_1, X_2, \dots, X_n be i.i.d. random variables. Then, $X_{k,n}$, the k^{th} order statistic of X_i , $1 \leq i \leq n$, or the k^{th} smallest variable has the distribution,

$$f_{X_{k,n}}(x) = n \binom{n-1}{k-1} F_X(x)^{k-1} (1-F_X(x))^{n-k} f_X(x),$$

where f_X is the probability density function (PDF) and F_X is the cumulative distribution function (CDF) of X_i for all i . In particular, if X_i 's are exponential with mean $1/\mu$, then the expectation and variance of order statistic $X_{k,n}$ are given by,

$$\mathbb{E}[X_{k,n}] = \frac{1}{\mu} \sum_{i=1}^k \frac{1}{n-k+i} = \frac{1}{\mu} (H_n - H_{n-k}), \quad (1)$$

$$\mathbb{V}[X_{k,n}] = \frac{1}{\mu^2} \sum_{i=1}^k \frac{1}{(n-k+i)^2} = \frac{1}{\mu^2} (H_{n^2} - H_{(n-k)^2}), \quad (2)$$

where H_n and H_{n^2} are generalized harmonic numbers defined by

$$H_n = \sum_{j=1}^n \frac{1}{j} \quad \text{and} \quad H_{n^2} = \sum_{j=1}^n \frac{1}{j^2}. \quad (3)$$

We observe from (1) that for fixed n , $\mathbb{E}[X_{k,n}]$ decreases when k becomes smaller. This fact will help us understand the analysis of download time in Section III and Section V respectively.

C. Assignment Policies

In our distributed storage model, we divide the content into k blocks, we use $1/k$ units of space of each disk, and hence total storage space used is n/k units. This is unlike conventional replication-based storage solutions where n entire copies of content are stored on the n disks. In such systems, each incoming request can be assigned to any of the n disks. One such assignment

policies is the power-of- d assignment [20], [25]. For each incoming request, the power-of- d job assignment uniformly selects d nodes ($d \leq n$) and sends the request to the node with least work left among the d nodes. The amount of work left of a node can be the expected time taken for that node to become empty when there are no new arrivals or simply the number of jobs queued. When $d = n$, power-of- d reduces to the least-work-left (LWL) policy (or joint-the-shortest-queue (JSQ) if work is measured by the number of jobs). Power-of- d assignment has received much attention recently due to the prevailing popularity in large-scale parallel computing. In Section III and Section V, we compare these policies with our proposed distributed storage model.

III. THE (n, k) FORK-JOIN SYSTEM

We consider the scenario that users attempt to download the content from the distributed storage system where their requests are placed in a queue at each disk. In Section III-A we propose the (n, k) fork-join system to model the queueing of download requests, and derive theoretical bounds on the expected download time in Section III-B. This analysis leads us to the fundamental trade-off between download time and storage, which provides insights into the practical system design. Numerical and simulation results demonstrating this trade-off are presented in Section III-C.

A. System Model

We model the queueing of download requests at the disks using the (n, k) fork-join system which is defined as follows.

Definition 1 ((n, k) fork-join system). *An (n, k) fork-join system consists of n nodes. Every arriving job is divided into n tasks which enter first-come first-serve queues at each of the n nodes. The job departs the system when any k out of n tasks are served by their respective nodes. The remaining $n - k$ tasks abandon their queues and exit the system before completion of service.*

The (n, n) fork-join system, known in literature as fork-join queue, has been extensively studied in, e.g., [21]–[23]. However, the (n, k) generalization in Definition 1 above has not been previously studied to our best knowledge. Fig. 2 illustrates the $(3, 2)$ fork-join system

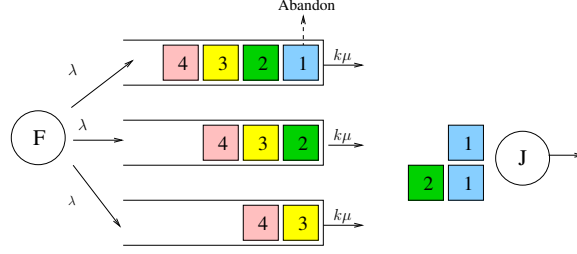


Fig. 2: Illustration of the $(3, 2)$ fork-join system. Since 2 out of 3 tasks of Job 1 are served, the third task abandons its queue and the job exits the system. Job 2 has to wait for one more task to be served.

corresponding to the coded distributed storage example shown in Fig. 1. Each download request, or a job is forked to the 3 nodes. When 2 out of 3 tasks are served, the third task abandons its queue and the job exits the system. For example, Job 1 is about to exit the system, while Job 2 is waiting for one more task to be served. The letters F and J denote the fork and join operations respectively.

We consider that arrival of download requests is Poisson with rate λ . Every request is forked to the n disks. The time taken to download one unit of data is exponential with mean $1/\mu$. Since, each disk stores $1/k$ units of data, consider that the service time for each node is exponentially distributed with mean $1/\mu'$ where $\mu' = k\mu$. Define the load factor $\rho' \triangleq \lambda/\mu'$. This model with an M/M/1 queue at every disk is sometimes referred to as a Flatto-Hahn-Wright (or FHW) model [26], [27] in fork-join queue literature. While most of our analytical results in Section III and Section V are for the FHW model, and we use simulations to study systems with M/G/1 queues at the disks in Section IV.

For the (n, n) fork-join system to be stable, [28] shows that the arrival rate λ must be less than μ' , the service rate of a node, which in our (n, n) system equals to $n\mu$. In Lemma 1 below, we show that $\lambda < n\mu$ is also a necessary condition for the stability of the (n, k) fork-join system for any $1 \leq k \leq n$.

Lemma 1 (Stability of (n, k) fork-join system). *For the (n, k) fork-join system to be stable, the*

rate of Poisson arrivals λ and the service rate $\mu' = k\mu$ per node must satisfy

$$\lambda < \frac{n\mu'}{k} = n\mu. \quad (4)$$

Proof: Tasks arrive at each queue at rate λ and are served at rate $\mu' = k\mu$. But when k out of the n tasks finish service, the remaining $n - k$ tasks abandon their queues. A task can be one of the abandoning tasks with probability $(n - k)/n$. Hence the effective arrival rate to each queue is λ minus the rate of abandonment $\lambda(n - k)/n$. Then the condition for stability of each queue is

$$\lambda - \frac{\lambda(n - k)}{n} < \mu', \quad (5)$$

which reduces to (4). ■

B. Bounds on the Mean Response Time

Our objective is to determine the expected download time, which we refer to as the mean response time $T_{(n,k)}$ of the (n, k) fork-join system. It is the expected time that a job spends in the system, from its arrival until k out of n of its tasks are served by their respective nodes. Previous works [21]–[23] have studied $T_{(n,n)}$, but it has not been found in closed form – only bounds are known. An exact expression for the mean response time is found only for the $(2, 2)$ fork-join system [22].

Since the n tasks are served by independent M/M/1 queues, intuition suggests that $T_{(n,k)}$ is the k^{th} order statistic of n exponential service times. However this is not true, which makes the analysis of $T_{(n,k)}$ challenging. The reason why the order statistics approach does not work is that when j nodes ($j < n$) finish serving their tasks they can start serving the tasks of the next job (cf. Fig. 2). As a result, the service time of a job depends on the departure time of previous jobs.

We now present upper and lower bounds on the mean response time $T_{(n,k)}$. The numerical results in Section III-C show that these bounds are fairly tight.

Theorem 1 (Upper Bound on Mean Response Time). *The mean response time $T_{(n,k)}$ of an (n, k)*

fork-join system satisfies

$$T_{(n,k)} \leq \frac{H_n - H_{n-k}}{\mu'} + \frac{\lambda[(H_{n^2} - H_{(n-k)^2}) + (H_n - H_{(n-k)})^2]}{2\mu'^2[1 - \rho'(H_n - H_{n-k})]}, \quad (6)$$

where λ is the request arrival rate, μ' is the service rate at each queue, $\rho' = \lambda/\mu'$ is the load factor, and the generalized harmonic numbers H_n and H_{n^2} are as given in (3). The bound is valid only when $\rho'(H_n - H_{n-k}) < 1$.

Proof: To find this upper bound, we use a model called the split-merge system, which is similar but easier to analyze than the fork-join system. In the (n, k) fork-join queueing model, after a node serves a task, it can start serving the next task in its queue. On the contrary, in the split-merge model, the n nodes are blocked until k of them finish service. Thus, the job departs all the queues at the same time. Due to this blocking of nodes, the mean response time of the (n, k) split-merge model is an upper bound on (and a pessimistic estimate of) $T_{(n,k)}$ for the (n, k) fork-join system.

The (n, k) split-merge system is equivalent to an M/G/1 queue where arrivals are Poisson with rate λ and service time is a random variable S distributed according to the k^{th} order statistic of the exponential distribution. The mean and variance of S are (cf. (1) and (2))

$$\mathbb{E}[S] = \frac{H_n - H_{n-k}}{\mu'} \quad \text{and} \quad \mathbb{V}[S] = \frac{H_{n^2} - H_{(n-k)^2}}{\mu'^2}. \quad (7)$$

The Pollaczek-Khinchin formula [29] gives the mean response time T of an M/G/1 queue in terms of the mean and variance of S as,

$$T = \mathbb{E}[S] + \frac{\lambda(\mathbb{E}[S]^2 + \mathbb{V}[S])}{2(1 - \lambda\mathbb{E}[S])}. \quad (8)$$

Substituting the values of $\mathbb{E}[S]$ and $\mathbb{V}[S]$ given by (7), we get the upper bound (6). Note that the Pollaczek-Khinchin formula is valid only when $\frac{1}{\lambda} > \mathbb{E}[S]$, the stability condition of the M/G/1 queue. Since $\mathbb{E}[S]$ increases with k , there exists a k_0 such that the M/G/1 queue is unstable for all $k \geq k_0$. The inequality $\frac{1}{\lambda} > \mathbb{E}[S]$ can be simplified to $\rho'(H_n - H_{n-k}) < 1$ which is the condition for validity of the upper bound given in Theorem 1. ■

Remark 1. For the (n, n) fork-join system, the authors in [22] find an upper bound on mean response time different from (6) derived above. To find the bound, they first prove that the response times of the n queues form a set of associated random variables [30]. Then they use the property of associated random variables that their expected maximum is less than that for independent variables with the same marginal distributions. However this approach used in [22] cannot be 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 $k < n$.

As a corollary to Theorem 1 above, we can get an exact expression for $T_{(n,1)}$, the mean response time of the $(n, 1)$ fork-join system. Recall that in the $(n, 1)$ fork-join system, the entire content is replicated on n disks, and we just have to wait for any one disk to serve the incoming request.

Corollary 1. The mean response time $T_{(n,1)}$ of the $(n, 1)$ fork-join system is given by

$$T_{(n,1)} = \frac{1}{n\mu - \lambda}, \quad (9)$$

where λ is the rate of Poisson arrivals and μ is the service rate.

Proof: In Theorem 1 we constructed the (n, k) split-merge system which always has worse response time than the corresponding (n, k) fork-join system. For the special case when $k = 1$, the split-merge system is equivalent to the fork-join system and gives the same response time. Substituting $k = 1$ and $\mu' = k\mu = \mu$ in (7) and (8) we get the result (9). ■

Theorem 2 (Lower Bound on Mean Response Time). The mean response time $T_{(n,k)}$ of an (n, k) fork-join queueing system satisfies

$$T_{(n,k)} \geq \frac{1}{\mu'} [H_n - H_{n-k} + \rho' (H_{n(n-\rho')} - H_{(n-k)(n-k-\rho')})], \quad (10)$$

where λ is the request arrival rate, μ' is the service rate at each queue, $\rho' = \lambda/\mu'$ is the load factor, and the generalized harmonic number $H_{n(n-\rho')}$ is given by

$$H_{n(n-\rho')} = \sum_{j=1}^n \frac{1}{j(j-\rho')}.$$

Proof: The lower bound in (10) is a generalization of the bound for the (n, n) fork-join system derived in [23]. 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.

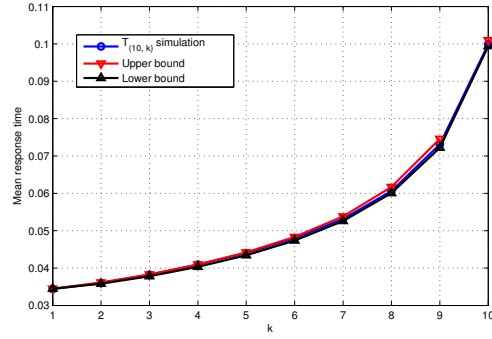
We now show that the service rate of a job in the j^{th} stage of processing is *at most* $(n - j)\mu'$. Consider two jobs B_1 and B_2 in the i^{th} and j^{th} stages of processing respectively. Let $i > j$, that is, B_1 has completed more tasks than B_2 . Job B_2 moves to the $(j + 1)^{\text{th}}$ stage 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, the time for a job to move from the j^{th} to $(j + 1)^{\text{th}}$ stage is lower bounded by $1/((n - j)\mu' - \lambda)$, the mean response time of an M/M/1 queue with arrival rate λ and service rate $(n - j)\mu'$. The total mean response time is the sum of the mean response times of each of the k stages of processing and is bounded below as

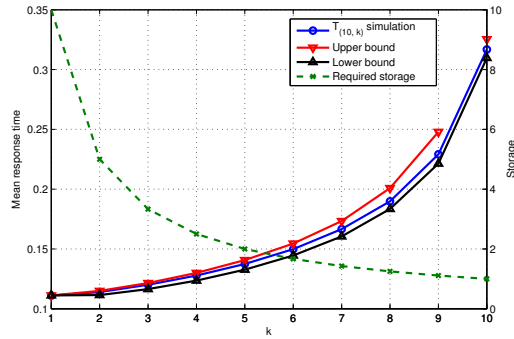
$$\begin{aligned} T_{(n,k)} &\geq \sum_{j=0}^{k-1} \frac{1}{(n-j)\mu' - \lambda}, = \frac{1}{\mu'} \sum_{j=0}^{k-1} \frac{1}{(n-j) - \rho'}, \\ &= \frac{1}{\mu'} \sum_{j=0}^{k-1} \left[\frac{1}{n-j} + \frac{\rho'}{(n-j)(n-j-\rho')} \right], \\ &= \frac{1}{\mu'} [H_n - H_{n-k} + \rho' (H_{n(n-\rho')} - H_{(n-k)(n-k-\rho')})]. \end{aligned}$$

■

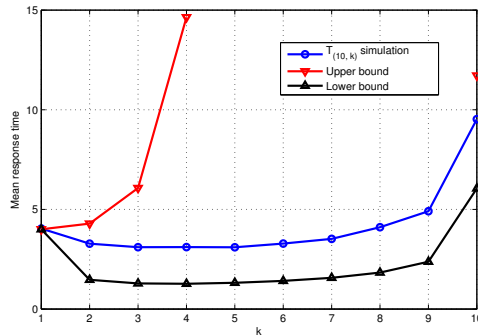
Hence, we have found lower and upper bounds on the mean response time $T_{(n,k)}$. In Fig. 3 we demonstrate how the tightness of the bounds changes with service rate μ . The figure shows the mean response time of a $(10, k)$ fork-join system versus k for service rates $\mu = 3, 1$ and $1/8$. Note that the upper bound for $k = n$ shown in the plot is $T_{(n,n)} \leq H_n/(n\mu - \lambda)$ as given in



(a) Arrival rate $\lambda = 1$ and service rate $\mu = 3$.



(b) Arrival rate $\lambda = 1$ and service rate $\mu = 1$.



(c) Arrival rate $\lambda = 1$ and service rate $\mu = 1/8$.

Fig. 3: Behavior of the mean response time $T_{(10,k)}$ as k increases (and total storage n/k decreases). The plot shows that the bounds on mean response time given by (6) and (10) are tight when the system is lightly loaded and become loose as μ decreases and/or k increases.

[22], instead of the bound in (6). The reason behind this substitution is explained in Remark 1.

We observe in Fig. 3 that the bounds become loose as k increases and/or μ decreases. In particular, the upper bound becomes loose because the blocking of queues in split-merge system becomes significant when k increases and/or μ decreases. For $\mu = 1/8$, the upper bound in (6) becomes invalid for $k \geq 5$ because the condition $\rho'(H_n - H_{n-k}) < 1$ is violated. Similarly, the lower bound becomes loose with increasing k and decreasing μ because the difference between the actual service rate in the j^{th} stage of processing, and its bound $(n - j)\mu'$ increases. When $k = 1$, the bounds coincide and give $T_{(n,1)} = 1/(n\mu - \lambda)$.

C. Download Time vs. Storage Space Trade-off

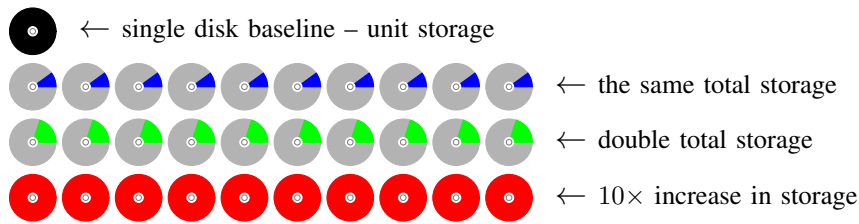
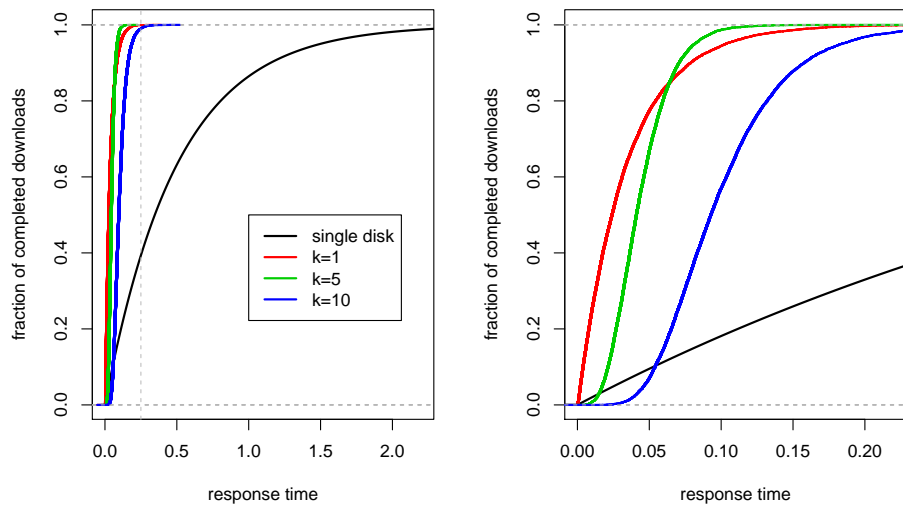


Fig. 4: CDFs of the response time of $(10, k)$ fork-join systems, and the required storage

In this section we present numerical results demonstrating the fundamental trade-off between storage and response time of the (n, k) fork-join system. We also compare the response time

of the (n, k) fork-join system to the power-of- d and LWL assignment policies introduced in Section II-C.

The expected download time of the file can be reduced in two ways 1) by increasing the total storage, or the storage expansion n/k per file, and 2) by increasing the number n of disks used for file storage. Both the total storage and the number of disks could be a limiting factor in practice. We first address the scenario where the number of disks n is kept constant, but the storage expansion changes from 1 to n as we choose k from 1 to n . We then study the scenario where the storage expansion factor n/k is kept constant, but the number of disks varies.

1) *Flexible Storage Expansion & Fixed Number of Disks:* Fig. 3 is a plot of mean response versus k for a fixed number of disks n . Note that as we increase k , the total storage n/k used decreases as shown in Fig. 3b. When we increase k , two factors affect the mean response time $T_{(n,k)}$ in opposite ways: 1) As k increases the storage per disk reduces which reduces mean response time. 2) With higher k we have to wait for more nodes to finish service for the job to exit the system. Hence we lose the diversity benefit of coding, which results in an increase in the mean response time.

In Fig. 3 we observe that when $\mu = 1$ or 3, the second factor dominates causing the mean response time $T_{(n,k)}$ to strictly increase with k . At lower service rate $\mu = \frac{1}{8}$ shown in Fig. 3c, the mean response time first decreases, and then increases with k . At small k (e.g. $k = 1$), the per-node service time $1/k\mu$ becomes large, outweighing the benefit of waiting for just k nodes to finish service. At large k , waiting for many nodes to response outweighs the fast $1/k\mu$ service time. Due to this phenomenon, there is an optimal k that minimizes the mean response time.

In addition to small mean response time, ensuring quality-of-service to the user may also require that the probability of exceeding some maximum tolerable response time to be small. Thus, we study the cumulative distribution function (CDF) of the response time for different values of k for a fixed n . In Fig. 4 we plot the CDF of the response time with $k = 1, 2, 5, 10$ for fixed $n = 10$. The arrival rate and service rate are $\lambda = 1$ and $\mu = 3$ as defined earlier. For $k = 1$, the CDF is represents the minimum of n exponential random variables, which is also exponentially distributed.

The CDF plot can be used to design a storage system that gives probabilistic bounds on the response time. For example, if we wish to keep the response time below 0.1 seconds with

probability at least 0.75, then the CDF plot shows that $k = 5, 10$ satisfy this requirement but $k = 1$ does not. The plot also shows that at 0.4 seconds, 100% of requests are complete in all fork-join systems, but only 50% are complete in the single-disk case.

2) *Flexible Number of Disks & Fixed Storage Expansion* : Next, we take a different viewpoint and analyze the benefit of spreading the content across more disks while using the same total storage space. Fig. 5 plots the bounds (6) and (10) on the mean response time $T_{(n,k)}$ versus k while keeping constant code rate $k/n = 1/2$, for the (n, k) fork-join system with $\lambda = 1$ and three different values of μ . For these parameter values the bounds are tight, and can be used for

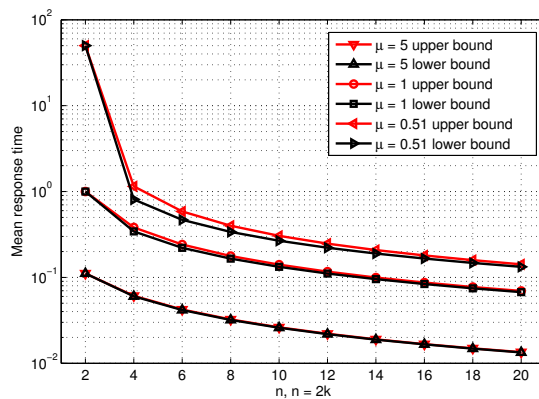


Fig. 5: Mean response time upper and lower bounds on the mean response time $T_{(n,n/2)}$ for $\lambda = 1$ and three different service rates μ . Due to the diversity advantage of more disks, $T_{(n,n/2)}$ reduces with n .

analysis in place of simulations.

We observe that the mean response time $T_{(n,k)}$ reduces as k increases because we get the diversity advantage of having more disks. The reduction in $T_{(n,k)}$ happens at the higher rate for small values of k and μ . For heavy-tailed distributions (e.g. Pareto, cf. Sec. IV), the benefit that comes from diversity is even larger.

$T_{(n,k)}$ approaches zero as $n \rightarrow \infty$ for a fixed storage expansion n/k . This is because we assumed that service rate of a single disk is $k\mu$ since the $1/k$ units of the content F is stored on one disk. However, in practice the mean service time $1/k\mu$ will not go zero as reading each disk will need some non-zero setup time in completing each task, irrespective of the amount of

data read from the disk. In Section IV we will see how this setup time affects the delay-storage trade-off.

In order to understand the response time better, we plot in Fig. 6 the CDF for different values of k for a fixed ratio $k/n = 1/2$. Again we observe that the diversity of increasing number of disks n helps to reduce the response time.

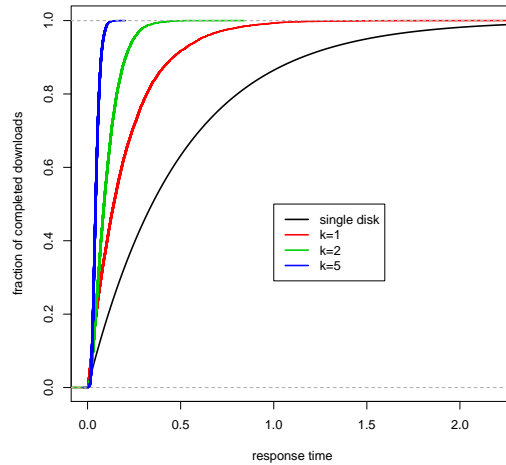


Fig. 6: CDFs of the response time of $(n, k = n/2)$ fork-join systems, and the required storage

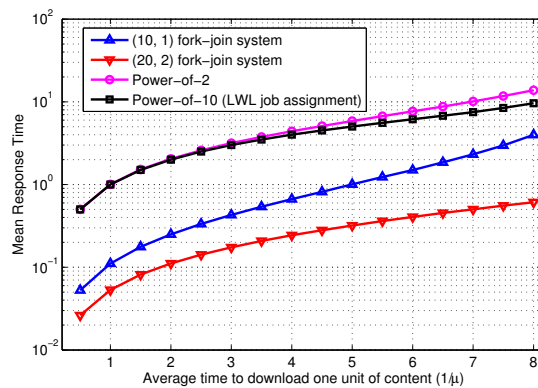


Fig. 7: For $\lambda = 1$ and the same amount of total storage used (10 units), the fork-join system has lower mean response time than the corresponding power-of- d and LWL assignment policies.

3) *Comparison with Power-of- d Assignment:* We now compare the mean response time of the (n, k) fork-join system with power-of- d and least-work-left (LWL) job assignment introduced in Section II. Recall that for each incoming request, the power-of- d policy assigns a request to the node with the least-work-left from among d uniformly selected nodes. Fig. 7 is a plot of the mean response time versus $1/\mu$, the average time taken to download one unit of content. It compares the (n, k) fork-join system which uses n/k units of total storage with the power-of- d and LWL assignment policies with the entire content (one unit) replicated on the n/k disks. Thus, all the systems shown in Fig. 7 use the same total storage space $n/k = 10$ units.

We observe in Fig. 7 that the fork-join system outperforms the power-of- d and LWL assignment policies. This is because as we saw in Fig. 5, when we increase n and k while keeping the ratio n/k (the total storage) fixed, the mean response time of the (n, k) fork-join system decreases. That is, the diversity advantage dominates over the slowdown due to waiting for more nodes to finish service. Thus, for large enough n , the (n, k) fork-join system outperforms the corresponding power-of- d scheme that uses the same storage space n/k units.

There are other practical issues that are not considered in Fig. 7. For instance, in the (n, k) fork-join system there are communication costs associated with forking jobs to n nodes and costs of decoding the MDS coded blocks. On the other hand, the power-of- d assignment system requires constant feedback from the nodes to determine the work left at each node.

IV. GENERALIZING THE SERVICE DISTRIBUTION

The theoretical analysis and numerical results so far assumed a specific service time distribution at each node – we considered the exponential distributions. In this section we present some results by generalizing the service time distribution. In Section IV-A we extend the upper bound to general service time distributions. We present numerical results for heavy-tailed and correlated service times in Section IV-B and Section IV-C respectively.

A. General Service Time Distribution

In several practical scenarios the service distribution is unknown. We present an upper bound on the mean response time for such cases, only using the mean and the variance of the service

distribution. Let X_1, X_2, \dots, X_n be the i.i.d random variables representing the service times of the n nodes, with expectation $E[X_i] = \frac{1}{\mu'}$ and variance $V[X_i] = \sigma^2$ for all i .

Theorem 3 (Upper Bound with General Service Time). *The mean response time $T_{(n,k)}$ of an (n, k) fork-join system with general service time X such that $E[X] = \frac{1}{\mu'}$ and $V[X] = \sigma^2$ satisfies*

$$T_{(n,k)} \leq \frac{1}{\mu'} + \sigma \sqrt{\frac{k-1}{n-k+1}} + \frac{\lambda \left[\left(\frac{1}{\mu'} + \sigma \sqrt{\frac{k-1}{n-k+1}} \right)^2 + \sigma^2 C(n, k) \right]}{2 \left[1 - \lambda \left(\frac{1}{\mu'} + \sigma \sqrt{\frac{k-1}{n-k+1}} \right) \right]}. \quad (11)$$

Proof: The proof follows from Theorem 1 where the upper bound can be calculated using (n, k) split-merge system and Pollaczek-Khinchin formula (8). Unlike the exponential distribution, we do not have an exact expression for S , i.e., the k^{th} order statistic of the service times X_1, X_2, \dots, X_n . Instead, we use the following upper bounds on the expectation and variance of S derived in [31] and [32].

$$E[S] \leq \frac{1}{\mu'} + \sigma \sqrt{\frac{k-1}{n-k+1}}, \quad (12)$$

$$V[S] \leq C(n, k) \sigma^2. \quad (13)$$

The proof of (12) involves Jensen's inequality and Cauchy-Schwarz inequality. For details please refer to [31]. The constant $C(n, k)$ depends only on n and k , and can be found in the table in [32]. Holding n constant, $C(n, k)$ decreases as k increases. The proof of (13) can be found in [32].

Note that (8) strictly increases as either $E[S]$ or $V[S]$ increases. Thus, we can substitute the upper bounds in it to obtain the upper bound on mean response time (11). ■

Regarding the lower bound, we note that our proof in Theorem 2 cannot be extended to this general service time setting. The proof requires memoryless property of the service time, which does not necessary hold in the general service time case.

B. Heavy-tailed Service Time

In many practical systems the service time has a heavy-tail distribution, which means that there is a larger probability of getting very large values. More formally, a random variable

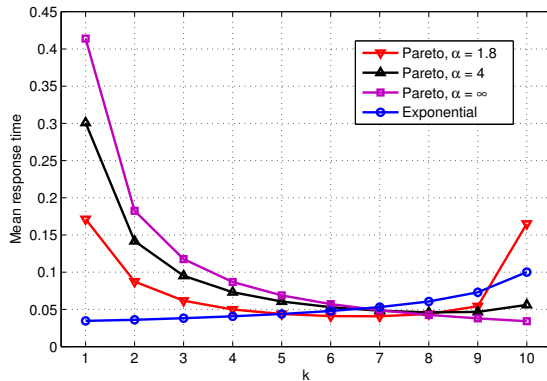


Fig. 8: Mean response time $T_{(10,k)}$ of different service time distributions. $\lambda = 1$ and $\mu = 3$. For more heavy-tailed (smaller α) distributions, the increase in mean response time with k becomes dominant since we have to wait for more nodes to finish service.

X is said to be heavy-tail distribution if its tail probability is not exponentially bounded and $\lim_{x \rightarrow \infty} e^{\beta x} \Pr(X > x) = \infty$ for all $\beta > 0$. We consider the Pareto distribution which has been widely used to model heavy-tailed jobs in existing literature (see for example [33], [34]). The Pareto distribution is parametrized by scale parameter x_m and shape parameter α and its cumulative distribution function is given by,

$$F_X(x) = \begin{cases} 1 - \left(\frac{x_m}{x}\right)^\alpha & \text{for } x \geq x_m \\ 0 & \text{for } x < x_m \end{cases} \quad (14)$$

A smaller value of α implies a heavier tail. In particular, when $\alpha = \infty$ the service time becomes deterministic and when $\alpha \leq 1$ the service time becomes infinite. In [33] Pareto distribution with $\alpha = 1.1$ was reported for the sizes of files requested from websites.

In Fig. 8 we plot the mean response time $T_{(n,k)}$ versus k for $n = 10$ disks, for arrival rate $\lambda = 1$ and service rate $\mu = 3$ for the exponential and Pareto service distributions. Each disk stores $1/k$ units of data and thus the service rate of each individual queue is $\mu' = k\mu$. For a given k , all distributions have the same mean service time $1/k\mu$. We observe that as the distribution becomes more heavy-tailed (smaller α), waiting for more nodes (larger k) to finish results in an increase in mean response time which outweighs the decrease caused by smaller service time $1/k\mu$. For smaller α , the optimal k decreases because the increase in mean response time for

larger k is more dominant.

C. Correlated Service Times

Thus far we have considered that the n tasks of a job have independent service times. We now analyze how the correlation between service times affects the mean response time of the fork-join system. In practice the correlation between service times could be because the service time is proportional to the size of the file being downloaded. We model the correlation by considering that the service time of each task is $\delta X_d + (1 - \delta)X_{r,i}$, a weighted sum of two independent exponential random variables X_d and $X_{r,i}$ both with mean $1/k\mu$. The variable X_d is fixed across the n queues, and $X_{r,i}$ is the independent for the queues $1 \leq i \leq n$. The weight δ represents the degree of correlation between the service times of the n queues. When $\delta = 0$, the system is identical to the original (n, k) fork-join system analyzed in Section III. The mean response time $T'_{(n,k)}$ of the (n, k) fork-join system with service time distribution as described above is,

$$\begin{aligned} T'_{(n,k)} &= \delta \mathbb{E}[X_d] + (1 - \delta)T_{(n,k)}, \\ &= \frac{\delta}{k\mu} + (1 - \delta)T_{(n,k)}, \end{aligned} \tag{15}$$

where in $T_{(n,k)}$ is the response time with independent exponential service times analyzed in Section III. Fig. 9 shows the trade-off between mean response time and k for weight $\delta = 0, 0.5$, and 1. When δ is 0, coding provides diversity in this regime and gives faster response time for smaller k as we already observed in Fig. 3. As the correlation between service times increases we lose the diversity advantage provided by coding and do not get fast response for small k . Note that for $\delta = 1$, there is no diversity advantage and the decrease in response time with k is only because of the fact that each disk stores $1/k$ units of data.

V. THE (m, n, k) FORK-JOIN SYSTEM

In a distributed storage with a large number of disks m , having an (m, k) fork-join system would involve large signaling overhead of forking the request to all the m disks, and high decoding complexity. The decoding complexity is high even with small k because it depends on the field size, which is a function of m in standard codes such as Reed-Solomon codes. Hence, we propose a system where we divide the m disks into $g = m/n$ groups of n disks

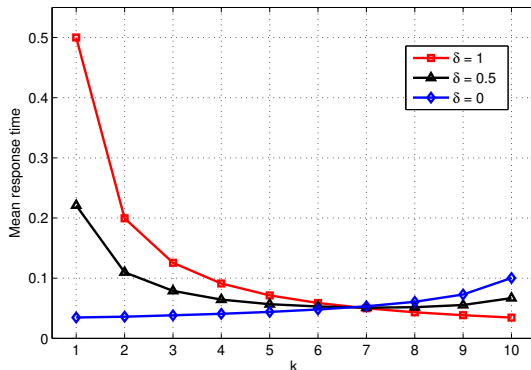


Fig. 9: Mean response time of $T'_{(n,k)}$ of $(10, k)$ fork-join systems with job dependent service time distribution for $\lambda = 1$, $\mu = 3$, and $\delta = 0, 0.5, 1$. As δ increases, the service times are more correlated and we lose the diversity advantage of coding.

each, which act as independent (n, k) fork-join systems. In Section V-A we give the system model and analyze the mean response time of the (m, n, k) fork-join system. In Section V-B we present numerical results comparing the mean response time with different policies of assigning an incoming request to one of the groups.

A. Analysis of Response Time

Consider a distributed storage system with m disks. We divide them into $g = m/n$ groups of n disks each as shown in Fig. 10. We refer to this system as the (m, n, k) fork-join system, formally defined as follows.

Definition 2 (The (m, n, k) fork-join system). *An (m, n, k) fork-join system consists of $m \geq n$ disks partitioned into $g = m/n$ groups with n disks each. An incoming download request is assigned to one of the g groups according to some policy (e.g., uniformly at random). Each group behaves as an independent (n, k) fork-join system described in Definition 1.*

We can extend Lemma 1 to find a necessary condition for the stability of the (m, n, k) fork-join system, in terms of the arrival rate λ_i to each group i , for $1 \leq i \leq g$.

Lemma 2 (Stability of (m, n, k) fork-join system). *For the (m, n, k) fork-join system to be*

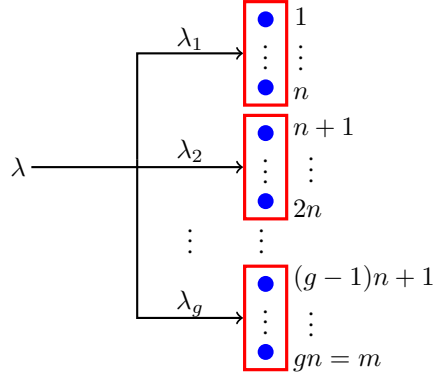


Fig. 10: The (m, n, k) fork-join system with incoming service requests split among $g = m/n$ fork-join systems.

stable, the rate of arrival of requests λ_i to group i and the service rate $\mu' = k\mu$ per node must satisfy

$$\lambda_i < n\mu, \quad \forall 1 \leq i \leq g. \quad (16)$$

Proof: Since each group behaves as an independent (n, k) fork-join system we can apply the condition of stability in Lemma 1 with λ replaced by λ_i . The result follows from this. ■

The response time of the (m, n, k) fork-join system depends on the policy used to assign an incoming request to one of the groups. Under the uniform job assignment policy, each incoming request is assigned to a group chosen uniformly at random from the g groups. The Poisson arrival rate to each group is then reduced to λ/g , and each group is an independent (n, k) fork-join system. Therefore, we can extend the bounds in Theorem 1 and Theorem 2 to the mean response time of the (m, n, k) fork-join system as follows.

Corollary 2. *The response time $T_{(m,n,k)}$ of an (m, n, k) fork-join system with uniform group assignment is bounded by (6) and (10) with λ replaced by λ/g .*

B. Numerical Results

To reduce the decoding complexity and signaling overhead, an (m, n, k) fork-join system with smaller n , and thus more groups $g = m/n$, is preferred. However, reducing n reduces the

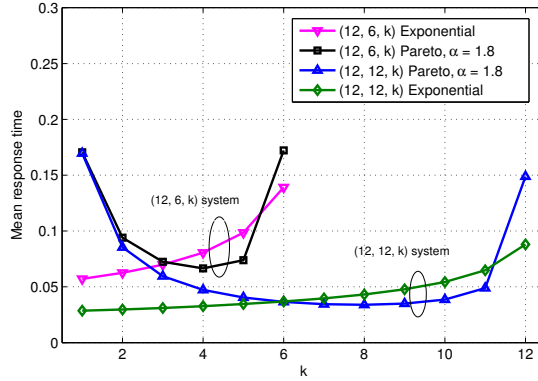


Fig. 11: Mean response time $T_{(12,n,k)}$ with the exponential and Pareto service distributions, and parameters $\lambda = 1$ and $\mu = 3$. Given m and we would like to find the smallest n and largest k that can achieve a given target response time.

diversity advantage which could give higher expected download time (cf. Fig. 6). Thus, there is a delay-complexity trade-off when we vary the number of groups g . Moreover, the content has to be replicated at all groups to which its request can be directed. Thus, having a large number of groups, also means increased storage space.

In Fig. 11 we plot the mean response time for $(12, n, k)$ system and uniform group assignment with exponential and Pareto service times. Given the number of disks m , we would like to find the smallest n , and largest k that can achieve a given target response time. Smaller n means there are less disks per group, and hence less signaling overhead of forking a request to the disks in a group. Larger k is desirable because the total storage space used is m/k units. For exponential service distribution, Fig. 11 shows that diversity of having a large n , or smaller k always gives lower response time. But this monotonicity does not hold for the Pareto service time distribution. For example, the $(12, 6, 3)$ fork-join system with $n = 6$ disks per group and $12/3 = 4$ units of storage used, gives lower response time than the $(12, 12, 2)$ fork-join system with $n = 12$ disks per group and total storage $12/2 = 6$ units of storage used.

We now study the response time of the (m, n, k) fork-join system under three different group assignment policies – the uniform job assignment policy, where each incoming request is assigned to a group chosen uniformly at random from the g groups, and the power-of- d and least-work-left

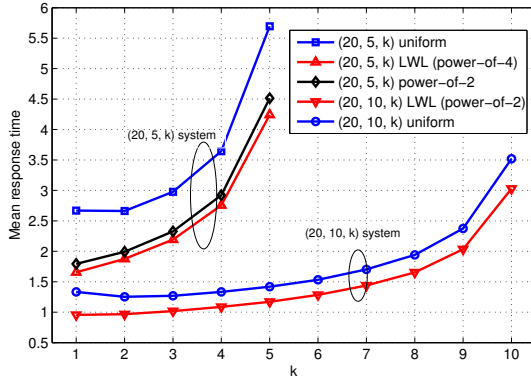


Fig. 12: Mean response time of $(20, n, k)$ systems, with $\lambda = 1$ and $\mu = 1/8$ for different group assignment policies. The power-of-2 and LWL assignment give faster response time than uniform assignment.

(LWL) policies introduced in Section II-C.

In Fig. 12 we show a comparison of the response time of the $(20, n, k)$ fork-join system with the uniform, power-of- d and LWL group assignment policies. Request arrival are Poisson with rate $\lambda = 1$ and service times are exponential with rate $\mu = 1/8$. As expected, the power-of- d assignments give lower response time than the uniform assignment but it is at the cost of receiving feedback about the amount of work left at each node. We again note that power-of-2 policy is only slightly worse than the LWL policy (cf. Fig. 7). The simulation suggests power-of- d group assignment is a strategy worth considering in actual implementations.

VI. CONCLUDING REMARKS

A. Major Implications

In this paper we show how coding in distributed storage systems, which has been used to provide reliability against disk failures, also reduces the content download time. We consider that content is divided into k blocks, and stored on $n > k$ disks or nodes in a network. The redundancy is added using an (n, k) maximum distance separable (MDS) code, which allows content reconstruction by reading any k of the n disks. Since the download time from each disk is random, waiting for only k out of n disks reduces overall download time significantly.

We take a queueing-theoretic approach to model multiple users requesting the content simultaneously. We propose the (n, k) fork-join system model where each request is forked to queues at the n disks. This is a novel generalization of the (n, n) fork-join system studied in queueing theory literature. We analytically derive upper and lower bounds on the expected download time and show that they are fairly tight. To the best of our knowledge, we are the first to propose the (n, k) fork-join system and find bounds on its mean response time. We also extend this analysis to distributed systems with large number of disks, that can be divided into many (n, k) fork-join systems.

Our results demonstrate the fundamental trade-off between the download time and the amount of storage space. This trade-off can be used for design of the amount of redundancy required to meet the delay constraints of content delivery. We observe that the optimal operating point varies with the service distribution of the time to read each disk. We present theoretical results for the exponential distribution, and simulation results for the heavy-tailed Pareto distribution.

B. Future Perspectives

Although, we focus on distributed storage here, the results in this paper can be extended to computing systems such as MapReduce [35] as well as content access networks [16], [36].

There are some practical issues affecting the download time that are not considered in this paper and could be addressed in future work. For instance, the signaling overhead of forking the request to n disks, and the complexity of decoding the content increases with n . In practical storage systems, adding redundancy in storage not only requires extra capital investment in storage and networking but also consumes more energy [37]. It would be interesting to study the fundamental trade-off between power consumption and quality-of-service. Finally, note that in this paper we focus on the *read* operation in a storage system. However in practical systems requests entering the system consist of both *read* and *write* operations – we leave the investigation of the *write* operation for future work.

REFERENCES

- [1] Amazon EBS, <http://aws.amazon.com/ebs/>.
- [2] S. Ghemawat, H. Gobioff, and S.-T. Leung, “The Google file system,” in *ACM SIGOPS Op. Sys. Rev.*, vol. 37, no. 5, 2003, pp. 29–43.

- [3] Dropbox, <http://www.dropbox.com/>.
- [4] Google Docs, <http://docs.google.com/>.
- [5] G. Joshi, Y. Liu, and E. Soljanin, "Coding for fast content download," *Allerton Conf. on Commun. Control and Comput.*, pp. 326–333, Oct. 2012.
- [6] A. G. Dimakis, P. B. Godfrey, M. Wainwright, and K. Ramchandran, "Network coding for distributed storage systems," *Proc. of IEEE INFOCOM*, pp. 2000–2008, May 2007.
- [7] M. Blaum, J. Brady, J. Bruck, and J. Menon, "EVENODD: an efficient scheme for tolerating double disk failures in RAID architectures," *IEEE Transactions on Computers*, vol. 44, no. 2, pp. 192–202, 1995.
- [8] R. Rodrigues and B. Liskov, "High availability in DHTs: Erasure coding vs. replication," *Int. Workshop Peer-to-Peer Sys.*, pp. 226–239, Feb. 2005.
- [9] K. V. Rashmi, N. B. Shah, P. V. Kumar, and K. Ramchandran, "Explicit construction of optimal exact regenerating codes for distributed storage," *Allerton Conf. on Commun. Control and Comput.*, pp. 1243 – 1249, Sep. 2009.
- [10] N. B. Shah, K. V. Rashmi, P. V. Kumar, and K. Ramchandran, "Interference alignment in regenerating codes for distributed storage: necessity and code constructions," *IEEE Trans. Inform. Theory*, vol. 58, pp. 2134–2158, Apr. 2012.
- [11] I. Tamo, Z. Wang and J. Bruck, "Zigzag Codes: MDS Array Codes With Optimal Rebuilding," *IEEE Transactions on Information Theory*, vol. 59, no. 3, pp. 1597–1616, 2013.
- [12] U. Ferner, M. Médard, and E. Soljanin, "Toward sustainable networking: Storage area networks with network coding," *Allerton Conf. on Commun. Control and Comput.*, pp. 517–524, Oct. 2012.
- [13] L. Huang, S. Pawar, H. Zhang, and Kannan Ramchandran, "Codes can reduce queuing delay in data centers," *Proc. Int. Symp. Inform. Theory*, pp. 2766–2770, Jul. 2012.
- [14] N. Shah, K. Lee, and K. Ramchandran, "The MDS queue: analyzing latency performance of codes and redundant requests," Tech. Rep. arXiv:1211.5405, Nov. 2012.
- [15] G. Kabatiansky, E. Krouk and S. Semenov, *Error correcting coding and security for data networks: analysis of the superchannel concept*, 1st ed. Wiley, Mar. 2005, ch. 7.
- [16] N. F. Maxemchuk, "Dispersivity routing in high-speed networks," *Compu. Networks and ISDN Sys.*, vol. 25, pp. 645–661, Jan. 1993.
- [17] Y. Liu, J. Yang, and S. C. Draper, "Exploiting route diversity in multi-packet transmission using mutual information accumulation," *Allerton Conf. on Commun. Control and Comput.*, pp. 1793–1800, Sep. 2011.
- [18] L. Xu, "Highly available distributed storage systems," Ph.D. dissertation, California Institute of Technology, 1998.
- [19] E. Soljanin, "Reducing delay with coding in (mobile) multi-agent information transfer," *Allerton Conf. on Commun. Control and Comput.*, pp. 1428–1433, Sep. 2010.
- [20] M. Harchol-Balter, *Performance Modeling and Design of Computer Systems: Queuing Theory in Action*. Cambridge University Press, 2013.
- [21] C. Kim and A. K. Agrawala, "Analysis of the fork-join queue," *IEEE Trans. Comput.*, vol. 38, no. 2, pp. 250–255, Feb. 1989.
- [22] R. Nelson and A. Tantawi, "Approximate analysis of fork/join synchronization in parallel queues," *IEEE Trans. Comput.*, vol. 37, no. 6, pp. 739–743, Jun. 1988.
- [23] E. Varki, A. Merchant and H. Chen, "The M/M/1 fork-join queue with variable sub-tasks," *unpublished, available online*.
- [24] S. Ross, *A first course in probability*, 6th ed. Prentice Hall, 2002, ch. 6.6, p. 273.

- [25] M. Mitzenmacher, “The power of two choices in randomized load balancing,” Ph.D. dissertation, University of California Berkeley, CA, 1996.
- [26] 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.
- [27] P. E. Wright, “Two parallel processors with coupled inputs,” *Advances in applied probability*, pp. 986–1007, 1992.
- [28] P. Konstantopoulos and J. Walrand, “Stationarity and stability of fork-join networks,” *J. Appl. Prob.*, vol. 26, pp. 604–614, Sep. 1989.
- [29] H. C. Tijms, *A first course in stochastic models*, 2nd ed. Wiley, 2003, ch. 2.5, p. 58.
- [30] J. Esary, F. Proschan and D. Walkup, “Association of random variables, with applications,” *Annals of Math. Stat.*, vol. 38, no. 5, pp. 1466–1474, Oct. 1967.
- [31] B. C. Arnold and R. A. Groeneveld, “Bounds on expectations of linear systematic statistics based on dependent samples,” *Annals of Stat.*, vol. 7, pp. 220–223, Oct. 1979.
- [32] N. Papadatos, “Maximum variance of order statistics,” *Ann. Inst. Statist. Math*, vol. 47, pp. 185–193, 1995.
- [33] M. E. Crovella and A. Bestavros, “Self-similarity in World Wide Web traffic: evidence and possible causes,” *IEEE/ACM Trans. Networking*, pp. 835–846, Dec. 1997.
- [34] M. Faloutsos, P. Faloutsos, and C. Faloutsos, “On power-law relationships of the Internet topology,” *Proc. ACM SIGCOMM*, pp. 251–262, 1999.
- [35] J. Dean and S. Ghemawat, “MapReduce: simplified data processing on large clusters,” *Comm. of ACM*, vol. 51, pp. 107–113, Jan. 2008.
- [36] A. Vulimiri, O. Michel, P. B. Godfrey, and S. Shenker, “More is less: reducing latency via redundancy,” *Proc. ACM HotNets*, pp. 13–18, 2012.
- [37] T. Bostoen, S. Mullender, and Y. Berbers, “Power-reduction techniques for data-center storage systems,” *ACM Comput. Surveys*, vol. 45, no. 3, 2013.