Mean-Field-Analysis of Coding versus Replication in Cloud Storage Systems

Bin Li
Coordinate Science Lab
University of Illinois at Urbana-Champaign
Urbana, IL 61801, USA
Email: lib@illinois.edu

Aditya Ramamoorthy
Department of ECE
Iowa State University
Ames, IA 50011, USA
Email: adityar@iastate.edu

R. Srikant
Coordinate Science Lab, Dept. of ECE
University of Illinois at Urbana-Champaign
Urbana, IL 61801, USA
Email: rsrikant@illinois.edu

***Abstract*—We study cloud-storage systems with a very large number of files stored in a very large number of servers. In such systems, files are either replicated or coded to ensure reliability, i.e., file recovery from server failures. This redundancy in storage can further be exploited to improve system performance (mean file access delay) through appropriate load-balancing (routing) schemes. However, it is unclear whether coding or replication is better from a system performance perspective since the corresponding queueing analysis of such systems is, in general, quite difficult except for the trivial case when the system load asymptotically tends to zero. Here, we study the more difficult case where the system load is not asymptotically zero. Using the fact that the system size is large, we obtain a mean-field limit for the steady-state distribution of the number of file access requests waiting at each server. We then use the mean-field limit to show that, for a given storage capacity per file, coding strictly outperforms replication at all traffic loads while improving reliability. Further, the factor by which the performance improves in the heavy-traffic is at least as large as in the light-traffic case. Finally, we validate these results through extensive simulations.**

I. Introduction

Data centers with massive numbers of servers are used by many modern companies to serve their storage and computational needs. In this paper, we focus on the storage component of data centers. Consider a company like Facebook which stores a very large number of files, such as pictures, videos, etc., in a very large number of servers. Requests for downloading files arrive at the server, and the goal is to serve these requests with as little delay as possible. Additionally, for reliability purposes, each file is stored in multiple servers, using either simple replication or coding, to ensure that data is not lost even when some servers suffer from failures. The goal of this paper is to understand how this redundancy can be exploited to reduce the mean file access delay. In particular, we are interested in understanding whether coding always outperforms replication in terms of mean file access delay, under the same storage requirements.

To illustrate the difference between coding and replication, let us first consider the replication scheme. Suppose that each file is replicated in two servers, and assume that the time to download a file from a server is exponentially distributed with mean 1 and is independent across servers. Suppose that the load-balancing policy is to route an arriving request to server with the smallest queue length (i.e., the server with the smallest number of waiting requests). If the arrival rate of file download requests is very small, then the queue lengths (i.e., the number of requests awaiting service) at each server will be close to zero and therefore, an arriving request can be routed at random to any server containing the file. In this case, it is clear that the mean file access delay is just 1.

Next, let us consider the coding case. In particular, assume that the file is coded into 4 chunks, where the size of each chunk is half the size of the original file, and further the code is such that the file can be recovered from any two chunks. This can be achieved via Maximum Distance Separable (MDS) codes (e.g., [1]) with parameters $(4, 2)$, where the file is partitioned into two equal-size chunks A_1 and A_2, and the coded chunks A_1, A_2, $A_1 + A_2$ and $A_1 + 2A_2$ are stored in 4 different servers, respectively. Since each chunk is half the size of the original file, we assume that the amount of time required to download a chunk from a server is exponential with mean $1/2$. The natural load-balancing policy in this case is to choose the two least loaded of the four servers containing the file, and route an arriving request for the file to these two servers. Again, if the arrival rate of file download requests is close to zero, then all queue lengths will be close to zero and each arriving request can be routed to any two servers containing the file. Since we need both servers to complete serving the chunks that they contain, the mean file access delay is given by $\mathbb{E}[\max(X_1, X_2)]$, where X_1 and X_2 are i.i.d. exponential random variables with mean $1/2$. A straightforward calculation shows that this delay is equal to 0.75. Thus, it is quite clear that the mean file access delay is improved by 25% under coding compared with replication when the arrival rate is asymptotically negligible. However, it is unclear whether such a result extends to the case of non-zero request arrival rates. In such a case, queueing effects cannot be ignored. This poses significant challenges for the delay analysis. The main purpose of this paper is to address this open and difficult problem. Our contributions in this work can be summarized as follows:

• We first present a model of storage, routing, and file access in very large data centers. The interesting aspect of the model is that individual files become irrelevant, and the system can be viewed as a queueing model with a very large number of servers, thus facilitating the so-called mean-field analysis.

• Next, we carry out the mean-field analysis of the queueing system under both coding and replication, and derive their analytical expressions whose solutions yield the steady-state queue length distribution of each queue.

• Then, we utilize the mean-field-limit to show that coding strictly outperforms replication in terms of mean file access delay under the same storage requirements. We further characterize the improvement factor in the heavy-traffic regime, which is at least as large as that in the light-traffic regime. To the best of our knowledge, this is the first analytical result in the area of mean-field analysis that deals with the expected job delay rather than the expected task delay, where a job corresponds to a file access request containing a certain number of tasks (chunk downloading requests) depending on the coding scheme.

• Finally, we perform extensive simulations to validate our results, where we also study various service distributions, and more than one load-balancing scheme.

A. Related Work

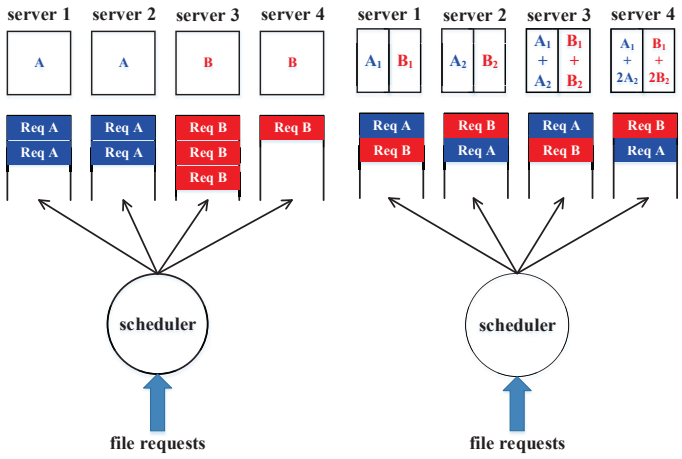
Delay reduction via coding in cloud storage systems: The main goal of a cloud storage system is to provide high data reliability and fast file access. Recently, much work has gone into the design of algorithms that speed up the file access in cloud storage systems. For example, papers (e.g., [2]–[4]) have performed simulation or testbed experiments to compare the delay performance of different coding schemes. Some other works investigated the file access delay performance analytically. For example, the authors in [5] showed that the MDS code has a smaller mean file access delay than the simple file replication. In [6], the authors provided delay bounds under the MDS code. Papers (e.g., [4], [7]–[13]) studied the delay performance of redundant requests in various settings. To the best of our knowledge, none of these works are able to characterize or analytically bound the performance improvement under coding compared with replication. Using the fact that the system size is large, we are able to obtain lower bounds on the performance improvement due to coding.

Load-balancing in the large-system limit: A load-balancing algorithm distributes arriving jobs across servers with the goal of minimizing queueing delays. The analysis of load-balancing algorithms in any finite systems is quite challenging in general. References [14] and [15] first considered the celebrated power-of-d-choices ($d \geq 2$) load-balancing algorithm in the large-system limit, where each arriving job is forwarded to the shortest d randomly sampled queues. There has been a considerable amount of recent work following the results in [14] and [15] studying various different load-balancing schemes with different amounts of overhead (e.g., [16]–[19]). But, to the best of our knowledge, none of the previous papers have studied the joint performance of load balancing and storage schemes in the large-system limit.

II. System Model

File storage scheme: We consider a cloud storage system with L servers, each of which stores a very large number of different types of files. Each file is stored using the Maximum Distance Separable (MDS) code with parameters (n, k) (see [1]), i.e., each file is encoded into n chunks with equal size stored at different servers, one for each server, and any k out of the n chunks are sufficient to recover the entire file. Since the storage space consumed at each server is $1/k$ of the size of the file, we assume that the time required for downloading data chunks are i.i.d. exponentially distributed[1] with mean of $1/k$. Note that the $(n, 1)$ code corresponds to the replication case, where each file is replicated at n different servers and thus we can download the desired file from any one of these n servers with exponential downloading time with mean 1.

Fig. 1(a) shows a small portion of the large storage system with $(2, 1)$ code, where file A is stored in servers 1 and 2, and file B is stored in servers 3 and 4. In order to download the file A, the scheduler can forward the file access request to either server 1 or server 2. Fig. 1(b) shows a part of the $(4, 2)$ coded system, where file A is divided into two equal-size halves A_1 and A_2, and the coded chunks A_1, A_2, A_1+A_2, and A_1+2A_2 are stored in four different servers, respectively. In order to access file A, the scheduler needs to forward the file download request to any two of four servers. File A is obtained only when these two download requests are processed, i.e., when we receive two chunks of file A from two different servers.



(a) $(2, 1)$ code (b) $(4, 2)$ code

Fig. 1: A small portion of a storage system. The letter inside the server box corresponds to the file it stores. Each server maintains a queue for download requests for the files it stores.

Arrival process: Recall that each file is stored in n servers. Thus, there are a total of $\binom{L}{n}$ subsets of servers where a file could be stored. We assume that there are $I = \Omega(L \log L)$ files in the system and I is an increasing function of L. These I files are stored such that the load on each server is approximately the same. Thus, we can model the arrival process as follows: we assume that the arrival process of file download requests is

[1]For the storage scheme with (n, k) code, the mean file access delay of the load-balancing scheme we consider is smaller under the positively correlated assumption than the i.i.d. assumption (cf. Section IV). In this sense, we try to characterize the mean delay performance of a particular storage scheme in the worse scenario.

Poisson with total arrival rate of $L\lambda$, where $\lambda \in (0,1)$. Further, each arrival requests a file uniformly at random from I files. Due to the property of the Poisson processes, this ensures that the load of any subset of servers of size n is independent with the same arrival rate.

Load-balancing algorithm: We assume that each server maintains a queue for file download requests that desire to download the chunks stored at the server, and processes these requests in the First-In-First-Out (FIFO) manner. Due to the MDS storage coding scheme, any k out of the n chunks are enough to obtain the entire file. Therefore, a natural load-balancing scheme is to forward an incoming file downloading request to the k least-loaded servers among n servers containing the file. In queueing theory jargon, upon job (file download request) arrival consisting of k tasks (data chunk retrieval request), forward these k tasks to the k least-loaded servers among n servers that can process this incoming job, one for each server. Each task processing time (chunk downloading time) follows exponential distribution with mean $1/k$. This load-balancing scheme is similar to the well-known *Batch Sampling* (BS) (e.g., [18], [20]). The main difference lies in that our considered load-balancing scheme uniformly selects one location containing n servers among I rather than $\binom{L}{n}$ different locations upon each job arrival, since there are a total of I files in the cloud storage system. Nevertheless, we still refer our load-balancing scheme as Batch Sampling in the rest of the paper.

Another popular load balancing scheme that has attracted much attention recently is called Redundant Request with Killing (RRK) (e.g., [7], [11], [13]). Under RRK, a request is sent to all servers where a file is stored, and when any k of these are served, the rest of the requests are killed. While this scheme is known to perform better than BS policy, it is under the assumption that the service time distributions are independent across servers. Later, in the simulations section, we show that the performance of RRK can be quite bad when service times across servers are correlated. For example, when a file is stored in equal-sized chunks across multiple servers, all requests for these chunks may have highly correlated service times. Thus, for our storage system, RRK has poor performance, so we do not study it here. On the other hand, in Section IV, we will show that the performance of the BS scheme is worst under the assumption that the service times are independent across different servers. Hence, we study the system under this assumption in this paper.

Finally, we make a comment on the scenario that is being modeled in our paper and some of the other prior works (e.g., [4], [7], [8], [10]–[12]). Our work views the problem from the point of view of storage service provider. On the other hand, the previous works (e.g., [4], [7], [8], [10]–[12]) view the problem from the point of view of a customer who uses a cloud storage system. Thus, in these other works, the service time of a file is a complicated function of one's own file size, the storage server's speed and the service provided to other customers. Thus, their assumptions regarding service times can be quite different from ours.

Goal: It is quite obvious that coding can significantly improve system reliability compared with replication. In this paper, we would like to investigate whether coding also reduces file access delay under BS load-balancing algorithm. While we derive queue length distributions for general (n,k) codes, we mainly compare the mean file access delays of (nk,k) and $(n,1)$ (replication) codes[2], both of which have the same storage requirements, where $k \geq 2$. Here, it is worth pointing out that none of existing works rigorously deal with the important and analytically hard problem of characterizing the mean job delay performance of the load-balancing schemes.

Let $\overline{W}^{(n,k)}$ be the mean file access delay under the (n,k) code. We first consider a trivial case, where the file request arrival rate is close to zero (also referred as the light-traffic regime). In such a case, queue lengths under both (nk,k) and $(n,1)$ codes are close to zero and thus the queueing effect can be ignored. Therefore, it is obvious that $\overline{W}^{(n,1)} = 1$ under the replication scheme.

Under the (nk,k) code, we need to download k chunks from k different servers to recover the entire file, and thus

$$\overline{W}^{(nk,k)} = \mathbb{E}\left[\max_{i=1,2,\cdots,k} X_i\right],$$

where $X_i, \forall i$, are i.i.d. with exponential distribution with mean $1/k$. According to [21], we have

$$\overline{W}^{(nk,k)} = \frac{H(k)}{k},$$

where $H(m) \triangleq \sum_{l=1}^{m} 1/l$ denotes m^{th} harmonic number. Thus, the (nk,k) code reduces delay by $100(1-H(k)/k)\%$ compared with the $(n,1)$ code in the light-traffic regime. In order to get a sense of how much delay improvement in this case, we plot the delay improvement percentage $100(1-H(k)/k)\%$ as a function of k. From Fig. 2, we can observe that the delay improvement is 25% when $k=2$, 38.89% when $k=3$, and the improvement becomes marginal as k further increases.

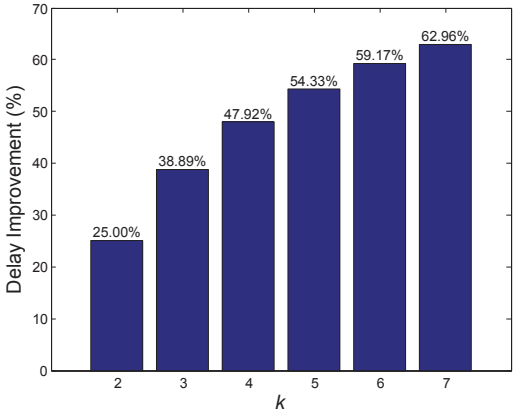


Fig. 2: Delay improvement under the (nk,k) code in the light-traffic regime (i.e., $\lambda \downarrow 0$)

[2]If files are stored using $(n,1)$ code such that arrival loads on each server are the same, then these files can also be stored using (nk,k) code to guarantee that arrival loads on each server are the same.

This interesting observation raises the following two natural questions in the non-zero arrival rate case where the queueing effect cannot be ignored: (i) does the (nk, k) code always outperform the $(n, 1)$ code in terms of mean file access delay? (ii) if it does, then how much performance improvement can it achieve? The goal of this paper is to address these two open questions. In particular, we show that the (nk, k) code always outperforms the $(n, 1)$ code in terms of mean file access delay at all traffic loads, and the improvement factor in the heavy-traffic regime is at least as large as in the light-traffic regime.

III. Mean-Field Analysis

In this section, we will use mean-field analysis to study the mean file access delay performance under the (n, k) code. The underlying assumptions behind the mean-field analysis are validated through simulations in Section IV.

Let $Q_l^{(L)}(t)$ be the length of the l^{th} queue at time t in a system with L queues. It is easy to check that the queue-length process $\{\mathbf{Q}^{(L)}(t)\}_{t\geq 0}$ is an irreducible and nonexplosive Markov chain. The following proposition further states that this Markov chain is positive recurrent and hence has a unique steady-state distribution.

Proposition 1: The Markov chain $\{\mathbf{Q}^{(L)}(t)\}_{t\geq 0}$ is positive recurrent. Moreover, the mean steady-state queue-length is finite, i.e.,

$$\mathbb{E}\left[\sum_{l=1}^{L} \widetilde{Q}_l^{(L)}\right] \leq \frac{L(1+\lambda)}{2(1-\lambda)}, \tag{1}$$

where $\widetilde{Q}_l^{(L)}$ is steady-state queue length of the l^{th} queue.

Proof: We first consider a quadratic Lyapunov function and study its conditional expected drift. Then, the desired result follows from the Foster-Lyapunov theorem. Please see our technical report [22] for details. ∎

Due to the symmetry, all queues have the same steady-state distribution. Let $\{\pi_m^{(L)}\}_{m\geq 0}$ be the steady-state queue-length distribution of one queue, where $\pi_m^{(L)}$ denotes the probability that queue-length is exactly equal to m. Let $s_m^{(L)} \triangleq \sum_{j=m}^{\infty} \pi_m^{(L)}$ be the probability that queue-length is at least m. Note that $s_0^{(L)} = 1$ and $s_m^{(L)}$ is non-increasing with respect to m, i.e., $1 = s_0^{(L)} \geq s_1^{(L)} \geq s_2^{(L)} \geq \cdots \geq 0$. In addition, we have $\sum_{j=m}^{\infty} s_j^{(L)} < \infty, \forall m = 1, 2, \cdots$. Indeed, according to Proposition 1, we have $\sum_{j=m}^{\infty} s_j^{(L)} \leq \sum_{j=1}^{\infty} s_j^{(L)} = \mathbb{E}\left[\widetilde{Q}_l^{(L)}\right] < \infty, \forall m \geq 1$, where we use the fact that $\mathbb{E}[Z] = \sum_{m=1}^{\infty} \Pr\{Z \geq m\}$ for any non-negative integer-valued random variable Z.

In this paper, our goal is to investigate the mean file access delay performance under the (n, k) code. In order to evaluate it accurately, it is important to obtain the queue-length distribution, i.e., the distribution of number of waiting download requests (queue-length) at each queue. However, queue lengths are correlated across queues and their distribution is hard to obtain in a system with finite number of queues. Fortunately, such correlations among queues become weaker and weaker as the number of servers increases. Therefore, we assume that any fixed number of queues become independent of each other as the number of servers goes to infinity, i.e., $L \to \infty$, where the queue-length distribution can be exactly characterized. Such an analysis in the large-system limit is commonly referred as *mean-field analysis*. In addition, a cloud storage system typically contains a very large number of servers, and therefore the mean-field analysis is accurate enough, as will be demonstrated in Section IV via simulations.

A. Steady-State Queue Length Distribution

In this subsection, we obtain the queue-length distribution under the (n, k) code in the large-system limit, i.e., $L \to \infty$.

Recall that all queues have the same steady-state distribution because of symmetry. Let $\overline{Q}^{(n,k)}$ be a random variable with the same distribution as the steady-state distribution of the queue-length under the (n, k) code in the large-system limit. Let $\overline{\pi}_m \triangleq \Pr\{\overline{Q}^{(n,k)} = m\}$ be the steady-state probability that queue length is equal to m in the large-system limit, where $m = 0, 1, 2, \cdots$. Under the (n, k) code, whenever there is an arriving file access request, we forward these tasks to the k least-loaded servers among n servers containing the file, one for each server. Note that the time required for downloading the chunks are i.i.d. with exponential distribution with mean $1/k$. We assume that n servers containing the file requested by the incoming job have independent queue-length distributions. Note that the queue-length of each server increases or decreases at most by one. Each queue forms an independent Markov chain, as shown in Figure 3.

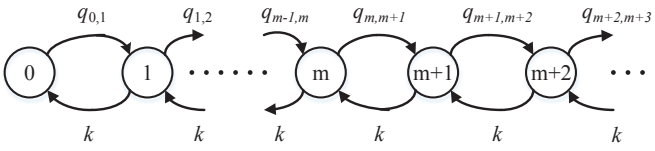


Fig. 3: The queue-length Markov chain of a single server in the large-system limit

According to the local balance equation, we have

$$\overline{\pi}_m q_{m,m+1} = k\overline{\pi}_{m+1}. \tag{2}$$

Therefore, in order to characterize the steady-state distribution $\{\overline{\pi}_m\}_{m\geq 0}$ in the large-system limit, we need to first obtain the transition rate $q_{m,m+1}$ when a file access request (job) arrives to a server with queue-length of m. Consider a particular server with queue-length of m. Its queue-length increases by 1 only when there is an arrival job and this server is one of the k least-loaded server among n servers containing the file that an incoming job requests. Note that $\overline{\pi}_m$ can also be interpreted as the fraction of servers with queue-length exactly equal to m in the large-system limit, which simply follows from the Strong Law of Large Numbers. Hence, $L\overline{\pi}_m$ is the average number of servers with queue-length of m and $L\overline{\pi}_m q_{m,m+1}\Delta$ is the average number of these servers that become of size $m + 1$ due to an arrival

in a small time interval Δ, which can also be represented as $L\lambda\Delta\sum_{i=1}^{k}\Pr\{\overline{Q}_{(i)}^{(n,k)}=m\}$. Thus, we have

$$\overline{\pi}_m q_{m,m+1}=\lambda\sum_{i=1}^{k}\Pr\{\overline{Q}_{(i)}^{(n,k)}=m\}, \tag{3}$$

where $\overline{Q}_{(i)}^{(n,k)}$ is the i^{th} smallest queue-length among n servers containing the file requested by the incoming job, i.e.,

$$\overline{Q}_{(1)}^{(n,k)}\le\overline{Q}_{(2)}^{(n,k)}\le\cdots\le\overline{Q}_{(i)}^{(n,k)}\le\cdots\le\overline{Q}_{(n)}^{(n,k)}.$$

The next lemma gives the exact expression for $\sum_{i=1}^{k}\Pr\{\overline{Q}_{(i)}^{(n,k)}=m\}$. Let $\overline{s}_m\triangleq\sum_{j=m}^{\infty}\overline{\pi}_j$ denote the steady-state probability that queue-length is at least m in the large-system limit.

Lemma 1: The term $\sum_{i=1}^{k}\Pr\{\overline{Q}_{(i)}^{(n,k)}=m\}$ can be expressed as follows:

$$\sum_{i=1}^{k}\Pr\{\overline{Q}_{(i)}^{(n,k)}=m\}=f^{(n,k)}(\overline{s}_m)-f^{(n,k)}(\overline{s}_{m+1}), \tag{4}$$

where $f^{(n,k)}(x)\triangleq\sum_{l=1}^{k}\binom{n}{n-k+l}\binom{n-k+l-2}{l-1}(-1)^{l-1}x^{n-k+l}$, $x\in[0,1]$.

Proof: We first simplify the expression of $\Pr\{\overline{Q}_{(i)}\ge m\}$ by using the mean-field assumption, and then derive the expression for $\sum_{i=1}^{k}\Pr\{\overline{Q}_{(i)}\ge m\}$ through a little bit complicated algebraic operations. Please see our technical report [22] for details. ∎

For example, $f^{(n,1)}(x)=x^n$ and $f^{(n,2)}(x)=nx^{n-1}-(n-2)x^n$. In general, the function $f^{(n,k)}(x)$ is quite complicated. However, it has several nice properties, which play an important role in later analysis.

Lemma 2: The function $f^{(n,k)}(x)$ (cf. Lemma 1) has the following three properties:

(i) $f^{(n,k)}(x)$ is strictly increasing, differentiable and convex on the interval $[0,1]$;
(ii) $f^{(n,k)}(0)=0$ and $f^{(n,k)}(1)=k$;
(iii) $f^{(n,k)}(x)$ has a bounded derivative, i.e.,

$$0\le\left(f^{(n,k)}(x)\right)'\le n,\quad\forall x\in[0,1].$$

Proof: We consider the first and second derivatives of the function $f^{(n,k)}(x)$, and utilize the subset-of-a-subset identity. Please see our technical report [22] for the proof. ∎

Fig. 4 sketches the graph of the function $f^{(n,k)}(x)$. We are now ready to characterize the steady-state queue-length distribution in the large-system limit.

Proposition 2: The steady-state queue-length distribution of a single server under the (n,k) code in the large-system limit is unique and can be characterized as follows:

$$\begin{cases}\overline{s}_{m+1}=\lambda f^{(n,k)}(\overline{s}_m)/k & \text{for } m=0,1,2,\cdots;\\ \overline{s}_0=1 & .\end{cases} \tag{5}$$

Proof: According to (2), (3) and Lemma 1, we have

$$\lambda\left(f^{(n,k)}(\overline{s}_m)-f^{(n,k)}(\overline{s}_{m+1})\right)=k(\overline{s}_{m+1}-\overline{s}_{m+2}), \tag{6}$$

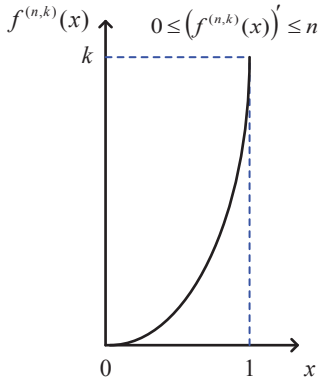


Fig. 4: The graph of the function $f^{(n,k)}(x)$

for any $m=0,1,2,\cdots$. Clearly if $\lambda f^{(n,k)}(\overline{s}_m)/k=\overline{s}_{m+1}$, then equation (6) holds. According to Lemma 2, the function $\lambda f^{(n,k)}(x)/k$ has a bounded derivative and thus it is Lipschitz. Also, $\lambda f^{(n,k)}(x)/k\in[0,1]$ since $f^{(n,k)}(x)\le k$ for all $x\in[0,1]$ and $\lambda\in(0,1)$. Therefore, the function $\lambda f^{(n,k)}(x)/k$ maps the convex and compact set $[0,1]$ to itself, and hence, according to the Schauder fixed point theorem, there exists a fixed point for the system of equations $\lambda f(\overline{s}_m)/k=\overline{s}_{m+1},\forall m\ge0$.

Next, we will show that this fixed point is unique. First, we note that

$$\overline{s}_{m+1}=\frac{\lambda}{k}f^{(n,k)}(\overline{s}_m)\overset{(a)}{\le}\lambda\overline{s}_m^{n/k}\overset{(b)}{\le}\lambda\overline{s}_m, \tag{7}$$

where the step (a) utilizes the inequality $f^{(n,k)}(x)\le kx^{n/k}$ for any $x\ge0$ (see Lemma 5 in our technical report [22]), and step (b) is true since $n>k$ and $0\le\overline{s}_m\le1$. Inequality (7) directly implies $\sum_{j=m}^{\infty}\overline{s}_j<\infty$. Hence, we have $\sum_{j=m}^{\infty}f^{(n,k)}(\overline{s}_j)<\infty$. Indeed,

$$\sum_{j=m}^{\infty}f^{(n,k)}(\overline{s}_j)\overset{(a)}{=}\sum_{j=m}^{\infty}\left(f^{(n,k)}(z_j)\right)'\overline{s}_j\le n\sum_{j=m}^{\infty}\overline{s}_j<\infty,$$

where the step (a) uses the fact that $f^{(n,k)}(\overline{s}_j)-f^{(n,k)}(0)=\left(f^{(n,k)}(z_j)\right)'\overline{s}_j$ for some $z_j\in[0,\overline{s}_j]$ according to the Mean-Value Theorem and the fact that $f^{(n,k)}(0)=0$; step (b) uses bounded derivative property of the function $f^{(n,k)}(x)$ (cf. Lemma 2). Therefore, by summing (6) over all $m\ge0$, we obtain $\overline{s}_1=\frac{\lambda}{k}f^{(n,k)}(\overline{s}_0)$. The uniqueness of the fixed point then follows from (6) by mathematical induction. ∎

Proposition 2 provides an iterative formula for exactly calculating the steady-state queue-length distribution under the (n,k) code. For example, under the $(n,1)$ code, i.e., power of n choices, according to Proposition 2, we have $\overline{s}_{m+1}=\lambda\overline{s}_m^n$ for all $m\ge0$ and $\overline{s}_0=1$, which implies that $\overline{s}_m=\lambda^{\frac{n^m-1}{n-1}}$. This exactly matches the results in [15] and [14]. Under the $(n,2)$ code, we have $\overline{s}_{m+1}=\frac{\lambda}{2}(n\overline{s}_m^{n-1}-(n-2)\overline{s}_m^n)$ for all $m\ge0$ and $\overline{s}_0=1$ from the Proposition 2.

We are now ready to evaluate the mean file access delay.

B. Mean File Access Delay Analysis

In this subsection, we analyze the mean file access delay performance under coding by using its steady-state queue-length distribution in the large-system limit (cf. Proposition 2). In particular, we characterize the delay improvement between (nk,k) and $(n,1)$ codes, both of which have the same storage requirements.

Proposition 3: (i) The mean file access delay under the (nk,k) code is at least $(1-H(k)/k)$ smaller than that under the $(n,1)$ code for any arrival rate $\lambda \in (0,1)$, i.e.,

$$\overline{W}^{(nk,k)} - \overline{W}^{(n,1)} \leq -\left(1-\frac{H(k)}{k}\right). \tag{8}$$

(ii) In the light-traffic regime (i.e., $\lambda \downarrow 0$), the mean file access delay under the (nk,k) code improves $100\,(1-H(k)/k)\,\%$ compared with the $(n,1)$ code, i.e.,

$$\lim_{\lambda \downarrow 0} \frac{\overline{W}^{(nk,k)} - \overline{W}^{(n,1)}}{\overline{W}^{(n,1)}} = -\left(1-\frac{H(k)}{k}\right). \tag{9}$$

In the heavy-traffic regime (i.e., $\lambda \uparrow 1$), the mean file access delay improvement under the (nk,k) code is at least $100\,(1-H(k)/k)\,\%$ compared with the $(n,1)$ code, i.e.,

$$\lim_{\lambda \uparrow 1} \frac{\overline{W}^{(nk,k)} - \overline{W}^{(n,1)}}{\overline{W}^{(n,1)}} \leq -\left(1-\frac{H(k)}{k}\right). \tag{10}$$

Remark: Our analysis shows that the (nk,k) code strictly outperforms the replication code at all traffic loads and its delay improvement in the heavy-traffic regime is at least as large as in the light-traffic regime. However, simulations in Section IV indicate that the performance improvement in heavy-traffic is even better.

Proof: Recall that under the (n,k) code, each job (file download request) contains k i.i.d. tasks (chunk download request) with exponential downloading time distribution with mean $1/k$. Upon job arrival, we forward its k tasks to the least-loaded k servers among n servers containing the file that the job request. Since a job is complete only when these k tasks are processed, if the queue lengths of these n servers are $\widehat{Q}_{(i)}^{(n,k)}$, $\forall i = 1,2,\cdots,n$ when a job arrives, then this job experiences a delay equal to

$$\max_{i=1,2,\cdots,k} \sum_{j=1}^{\widehat{Q}_{(i)}^{(n,k)}+1} X_j^{(k,i)}, \tag{11}$$

where $X_j^{(k,i)}$, $\forall i,j$, are i.i.d. exponential random variables with mean $1/k$, and $\widehat{Q}_{(i)}^{(n,k)}$ is the i^{th} smallest queue-length among n servers seen by an incoming job, i.e., $\widehat{Q}_{(1)}^{(n,k)} \leq \widehat{Q}_{(2)}^{(n,k)} \leq \cdots \leq \widehat{Q}_{(n)}^{(n,k)}$.

Note that (11) is true since the remaining service time for the task in service is still exponential. We also note that $\widehat{Q}_{(i)}^{(n,k)}$, $\forall i = 1,2,\cdots,n$ and $X_j^{(k,i)}, \forall i,j$, are independent. Therefore, the mean job delay $\overline{W}^{(n,k)}$ can be written as follows:

$$\overline{W}^{(n,k)} = \mathbb{E}\left[\max_{i=1,2,\cdots,k} \sum_{j=1}^{\widehat{Q}_{(i)}^{(n,k)}+1} X_j^{(k,i)}\right]. \tag{12}$$

Next, we compare the mean job delay under (nk,k) and $(n,1)$ codes.

$$\begin{aligned}
\overline{W}^{(nk,k)} &= \mathbb{E}\left[\max_{i=1,2,\cdots,k} \sum_{j=1}^{\widehat{Q}_{(i)}^{(nk,k)}+1} X_j^{(k,i)}\right] \\
&\overset{(a)}{\leq} \mathbb{E}\Bigg[\max\Bigg\{\sum_{j=1}^{\widehat{Q}_{(1)}^{(nk,k)}+1} X_j^{(k,1)}, \\
&\qquad \max_{i=2,\cdots,k} \sum_{j=1}^{\widehat{Q}_{(1)}^{(nk,k)}+1} X_j^{(k,i)} + \max_{i=2,\cdots,k} \sum_{j=\widehat{Q}_{(1)}^{(nk,k)}+2}^{\widehat{Q}_{(i)}^{(nk,k)}+1} X_j^{(k,i)}\Bigg\}\Bigg] \\
&\overset{(b)}{\leq} \mathbb{E}\left[\max_{i=1,2,\cdots,k} \sum_{j=1}^{\widehat{Q}_{(1)}^{(nk,k)}+1} X_j^{(k,i)}\right] \\
&\qquad + \mathbb{E}\left[\max_{i=2,\cdots,k} \sum_{j=\widehat{Q}_{(1)}^{(nk,k)}+2}^{\widehat{Q}_{(i)}^{(nk,k)}+1} X_j^{(k,i)}\right],
\end{aligned} \tag{13}$$

where the step (a) utilizes the fact that $\widehat{Q}_{(1)}^{(nk,k)} \leq \widehat{Q}_{(2)}^{(nk,k)} \leq \cdots \leq \widehat{Q}_{(k)}^{(nk,k)}$, and follows from the fact that $\max_i (x_i + y_i) \leq \max_i x_i + \max_i y_i$, for any non-negative real numbers x_i and y_i; step (b) utilizes the fact that $\max\{x, y+z\} \leq \max\{x,y\} + z$, for any non-negative real numbers x, y and z. By repeating steps in deriving (13) on the term

$$\mathbb{E}\left[\max_{i=2,\cdots,k} \sum_{j=\widehat{Q}_{(1)}^{(nk,k)}+2}^{\widehat{Q}_{(i)}^{(nk,k)}+1} X_j^{(k,i)}\right],$$

we obtain

$$\begin{aligned}
\overline{W}^{(nk,k)} \leq& \mathbb{E}\left[\max_{i=1,2,\cdots,k} \sum_{j=1}^{\widehat{Q}_{(1)}^{(nk,k)}+1} X_j^{(k,i)}\right] \\
&+ \sum_{l=2}^{k} \mathbb{E}\left[\max_{i=l,l+1,\cdots,k} \sum_{j=\widehat{Q}_{(l-1)}^{(nk,k)}+2}^{\widehat{Q}_{(l)}^{(nk,k)}+1} X_j^{(k,i)}\right] \\
\leq& \mathbb{E}\left[\sum_{j=1}^{\widehat{Q}_{(1)}^{(nk,k)}+1} \max_{i=1,2,\cdots,k} X_j^{(k,i)}\right] \\
&+ \sum_{l=2}^{k} \mathbb{E}\left[\sum_{j=\widehat{Q}_{(l-1)}^{(nk,k)}+2}^{\widehat{Q}_{(l)}^{(nk,k)}+1} \max_{i=l,l+1,\cdots,k} X_j^{(k,i)}\right],
\end{aligned} \tag{14}$$

where the last step follows from the fact that

$$\max_{i=1,2,\cdots,a}\sum_{j=1}^{b}x_j^{(i)}\leq\sum_{j=1}^{b}\max_{i=1,2,\cdots,a}x_j^{(i)}$$

holds for any positive integers a, b, and non-negative real numbers $x_j^{(i)}$, $\forall i=1,2,\cdots,a,\forall j=1,2,\cdots,b$.

Since $X_j^{(k,i)}$ are i.i.d. exponential random variables, according to [21], we have

$$\mathbb{E}\left[\max_{i=1,2,\cdots,m}X_j^{(k,i)}\right]=\frac{1}{k}H(m),\tag{15}$$

where we recall that $H(m)\triangleq\sum_{i=1}^{m}1/i$ is the m^{th} harmonic number. Note that since $X_j^{(k,i)},i=l,l+1,\cdots,k$, are i.i.d. and independent from $\widehat{Q}_{(l)}^{(nk,k)}$, by utilizing (15), inequality (14) becomes

$$\begin{aligned}\overline{W}^{(nk,k)}\leq&\frac{1}{k}\Bigg(\left(1+\mathbb{E}\left[\widehat{Q}_{(1)}^{(nk,k)}\right]\right)H(k)\\&+\sum_{l=2}^{k}\left(\mathbb{E}\left[\widehat{Q}_{(l)}^{(nk,k)}\right]-\mathbb{E}\left[\widehat{Q}_{(l-1)}^{(nk,k)}\right]\right)H(k-l+1)\Bigg)\\=&\frac{1}{k}\left(H(k)+\sum_{l=1}^{k}\frac{1}{k-l+1}\mathbb{E}\left[\overline{Q}_{(l)}^{(nk,k)}\right]\right),\end{aligned}\tag{16}$$

where we recall that $\overline{Q}_{(l)}^{(nk,k)}$ is the l^{th} smallest steady-state queue-length among nk servers, and the last step follows from PASTA property since the arrival process to any subset of queues of size nk is a Poisson process under the (nk,k) coding scheme.

On the other hand, the mean delay under the $(n,1)$ code can be written as follows:

$$\begin{aligned}\overline{W}^{(n,1)}=&\mathbb{E}\left[\sum_{j=1}^{\widehat{Q}_{(1)}^{(n,1)}+1}X_j^{(1,1)}\right]\\\overset{(a)}{=}&\mathbb{E}\left[\widehat{Q}_{(1)}^{(n,1)}\right]+1\\\overset{(b)}{=}&\mathbb{E}\left[\overline{Q}_{(1)}^{(n,1)}\right]+1,\end{aligned}\tag{17}$$

where the step (a) follows from the fact that $\widehat{Q}_{(1)}^{(n,1)}$ and $X_j^{(1,1)},\forall j$, are independent; step (b) follows from the PASTA property since the arrival process to any subset of queues of size n is a Poisson process under the $(n,1)$ coding scheme.

By using (16) and (17), we have

$$\begin{aligned}&\overline{W}^{(nk,k)}-\overline{W}^{(n,1)}\\\leq&-\left(1-\frac{H(k)}{k}\right)\\&+\frac{1}{k}\sum_{l=1}^{k}\frac{1}{k-l+1}\mathbb{E}\left[\overline{Q}_{(l)}^{(nk,k)}\right]-\mathbb{E}\left[\overline{Q}_{(1)}^{(n,1)}\right].\end{aligned}\tag{18}$$

Note that

$$\begin{aligned}&\frac{1}{k}\sum_{l=1}^{k}\frac{1}{k-l+1}\mathbb{E}\left[\overline{Q}_{(l)}^{(nk,k)}\right]\\\leq&\frac{1}{k}\sum_{l=1}^{k}\mathbb{E}\left[\overline{Q}_{(l)}^{(nk,k)}\right]\leq\mathbb{E}\left[\overline{Q}_{(1)}^{(n,1)}\right],\end{aligned}\tag{19}$$

where the last step utilizes Lemma 3. By substituting (19) into (18), we have (8).

Lemma 3: The average queue-length of k shortest queues among nk servers under the (nk,k) code is not greater than the queue-length of the shortest queue among n servers under the $(n,1)$ code, i.e.,

$$\frac{1}{k}\sum_{i=1}^{k}\mathbb{E}\left[\overline{Q}_{(i)}^{(nk,k)}\right]\leq\mathbb{E}\left[\overline{Q}_{(1)}^{(n,1)}\right].\tag{20}$$

The proof of Lemma 3 is available in technical report [22].

The mean job delay improvement under the (nk,k) code compared with the $(n,1)$ code in the light-traffic regime directly follows from the discussions in Section II. Next, we will investigate the mean job delay improvement in the heavy-traffic regime, i.e., $\lambda\uparrow1$. According to (18), we have

$$\begin{aligned}&\frac{\overline{W}^{(nk,k)}-\overline{W}^{(n,1)}}{\overline{W}^{(n,1)}}\\\leq&-\left(1-\frac{1}{k}H(k)\right)\\&+\frac{1}{k}\frac{\sum_{l=1}^{k}\frac{1}{k-l+1}\mathbb{E}\left[\overline{Q}_{(l)}^{(nk,k)}\right]-H(k)\mathbb{E}\left[\overline{Q}_{(1)}^{(n,1)}\right]}{1+\mathbb{E}\left[\overline{Q}_{(1)}^{(n,1)}\right]},\end{aligned}\tag{21}$$

which implies

$$\begin{aligned}&\lim_{\lambda\uparrow1}\frac{\overline{W}^{(nk,k)}-\overline{W}^{(n,1)}}{\overline{W}^{(n,1)}}\\\leq&-\left(1-\frac{1}{k}H(k)\right)+\frac{1}{k}\lim_{\lambda\uparrow1}\frac{\frac{\sum_{l=1}^{k}\frac{1}{k-l+1}\mathbb{E}\left[\overline{Q}_{(l)}^{(nk,k)}\right]}{\mathbb{E}\left[\overline{Q}_{(1)}^{(n,1)}\right]}-H(k)}{\frac{1}{\mathbb{E}\left[\overline{Q}_{(1)}^{(n,1)}\right]}+1}.\end{aligned}$$

By utilizing Lemma 4, we have the desired result.

Lemma 4: (i) The mean queue-length of the shortest queue among n servers under the $(n,1)$ code in the heavy-traffic regime satisfies

$$\lim_{\lambda\uparrow1}\frac{\mathbb{E}\left[\overline{Q}_{(1)}^{(n,1)}\right]}{-\log(1-\lambda)}=\frac{1}{\log n};\tag{22}$$

(ii) The mean queue lengths of the k shortest queues among n servers under the (nk,k) code satisfy

$$\lim_{\lambda\uparrow1}\frac{\sum_{i=1}^{k}\frac{1}{k-i+1}\mathbb{E}\left[\overline{Q}_{(i)}^{(nk,k)}\right]}{\mathbb{E}\left[\overline{Q}_{(1)}^{(n,1)}\right]}\leq H(k).\tag{23}$$

The proof of Lemma 4 is available in technical report [22]. ■

IV. Simulation Results

In this section, we provide simulation results to compare the mean file access delay performance between coding and replication in the system with $L = 1,000$ servers and $I = 1,000,000$ files. In particular, we first verify the accuracy of the mean-field analysis and then investigate the delay improvement under coding. Then, we evaluate the impact of correlation of the chunk downloading time on the mean delay performance for two different load-balancing algorithms.

A. Validation of the Mean-Field Analysis

In this subsection, we first validate the accuracy of the mean-field analysis, and then illustrate the differences in mean file access delay performance between coding and replication, where we assume that the chunk downloading time follows exponential distribution. Given the queue-length distribution (cf. Proposition 2), we are able to calculate the mean file access delay under the (n, k) code according to (12) through Monte Carlo methods. In particular, at each time slot, generate n i.i.d. queue-length random variables according to its steady-state probability distribution in the large-system limit (cf. Proposition 2), then pick k smallest ones and calculate the delay through (12). Then, the time-average delay can be regarded as the mean delay. The markers in Fig. 5 (corresponding to theoretical results) were obtained in this manner, whereas the simulation results were obtained via an event-driven simulation of the whole system.

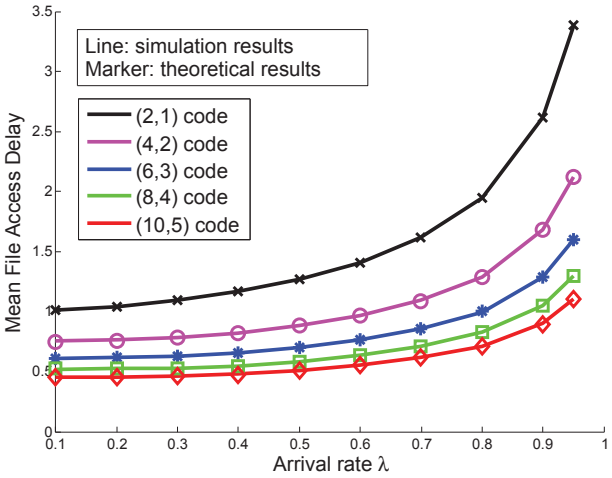
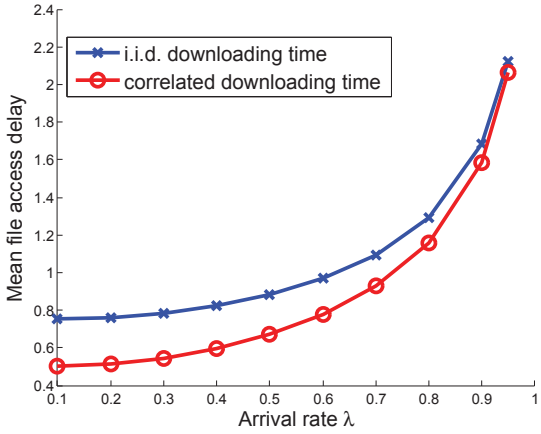
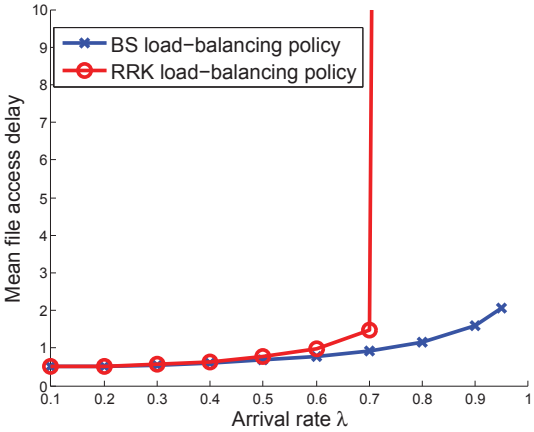


Fig. 5: Exponential downloading time

From Fig. 5, we first observe that the simulation results match the theoretical results very well under different coding schemes, which validates the accuracy of the mean-field analysis in the system even with $1,000$ servers. In addition, Fig. 5 shows the mean file access delay performance under the (nk, k) code, where $k = 1, 2, 3, 4, 5$. Recall that $k = 1$ corresponds to the replication code. We can see from Fig. 5 that the mean file access delay performance improves as k increases, where the delay improvement is most significant from $k = 1$ to $k = 2$. This is also expected from our theoretical analysis. In addition, for a fixed storage coding scheme, its delay improvement compared with the replication code increases as the arrival rate λ increases. We also consider another downloading time distribution in our technical report [22] and have similar observations.



(a) BS load-balancing scheme



(b) Correlated downloading time

Fig. 6: The impact of correlated downloading time on the delay performance of $(4, 2)$ code

B. Impact of Correlated Downloading Time Distribution

In this subsection, we consider another popular load-balancing scheme, called Redundant Request with Killing (RRK), under the storage scheme with (n, k) code. Recall that under the RRK load-balancing scheme, upon a file access request arrival, it forwards n requests to n servers containing the file and the entire file is obtained once k out of n downloading requests are processed.

Here, we consider both i.i.d. and correlated downloading time cases. In the case with i.i.d. downloading time, the time required for downloading data chunks are i.i.d. with exponential with mean $1/k$. In the case with correlated downloading time, the time required for downloading chunks associated with a file are exactly the same and follows exponential distribution with mean $1/k$.

Fig. 6 studies the impact of correlations on delay performance of the Batch Sampling (BS) and RRK load-balancing algorithms under the $(4, 2)$ storage scheme. From Fig. 6(a), we can observe that for the BS load-balancing policy, the mean delay under the correlated downloading time is always better than that under the i.i.d. downloading time, with larger improvement in the lower traffic regime. In this sense, the correlation of the chunk downloading time actually helps improve the delay performance of the BS policy. Thus, the results in the paper may be interpreted as characterizing the worst-case performance of the BS policy. However, from Fig. 6(b), we can see that this correlation significantly degrades the system performance of the RRK algorithm especially when the traffic load is high. The simulations for $(6, 3)$ code in our technical report [22] also show similar observations. Thus, the efficiency of the RRK policy heavily depends on the independence assumption on the chunk downloading time as we discussed in Section II. For this reason, we only analytically study the BS policy in this paper.

V. Conclusions

In this paper, we studied the mean file access delay performance under coding in cloud storage systems with a very large number of files stored in a very large number of servers. We formulated an appropriate load-balancing problem, and studied its delay performance in the large-system limit, i.e., when the number of servers goes to infinity. In particular, we obtained the steady-state distribution of the number of file access requests waiting at each server, and utilized this to show that coding always improves the mean file access delay compared with the simple replication scheme at all traffic loads, without sacrificing any storage and reliability. We further show that the improvement factor by coding in the heavy-traffic regime is at least as large as in the light-traffic regime. Finally, extensive simulations are performed to validate our theoretical results.

Acknowledgment

This work is supported in part by NSF grants ECCS-1202065, CCF-1409106, CCF-1320416, CCF-1149860, and ARO MURI W911NF-12-1-0385.

References

[1] S. Lin and D. J. Costello, *Error Control Coding, 2nd Ed.* Prentice Hall, 2004.

[2] S. Jain, M. Demmer, R. Patra, and K. Fall, "Using redundancy to cope with failures in a delay tolerant network," in *Proc. ACM Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications (SIGCOMM)*, Philadelphia, PA, USA, August 2005.

[3] G. Ananthanarayanan, A. Ghodsi, S. Shenker, and I. Stoica, "Why let resources idle? aggressive cloning of jobs with dolly," in *Proc. USENIX Conference on Hot Topics in Cloud Ccomputing (HotCloud)*, Boston, MA, USA, June 2012.

[4] A. Vulimiri, P. B. Godfrey, R. Mittal, J. Sherry, S. Ratnasamy, and S. Shenker, "Low latency via redundancy," in *Proc. ACM Conference on Emerging Networking Experiments and Technologies (CoNEXT)*, Santa Barbara, CA, USA, December 2013.

[5] L. Huang, S. Pawar, H. Zhang, and K. Ramchandran, "Codes can reduce queueing delay in data centers," in *Proc. IEEE International Symposium on Information Theory (ISIT)*, Cambridge, MA, USA, July 2012.

[6] N. B. Shah, K. Lee, and K. Ramchandran, "The MDS queue: Analysing the latency performance of erasure codes," in *Proc. IEEE International Symposium on Information Theory (ISIT)*, Honolulu, HI, USA, July 2014.

[7] ——, "When do redundant requests reduce latency?" in *Proc. Allerton Conference on Communication, Control, and Computing (Allerton)*, Monticello, IL, USA, October 2013.

[8] G. Joshi, Y. Liu, and E. Soljanin, "On the delay-storage trade-off in content download from coded distributed storage systems," *IEEE Journal on Selected Areas in Communications*, vol. 32, no. 5, pp. 989–997, 2014.

[9] Y. Xiang, T. Lan, V. Aggarwal, and Y. F. R. Chen, "Joint latency and cost optimization for erasurecoded data center storage," *ACM SIGMETRICS Performance Evaluation Review*, vol. 42, no. 2, pp. 3–14, 2014.

[10] S. Chen, Y. Sun, U. C. Kozat, L. Huang, P. Sinha, G. Liang, X. Liu, and N. B. Shroff, "When queueing meets coding: Optimal-latency data retrieving scheme in storage clouds," in *Proc. IEEE International Conference on Computer Communications (INFOCOM)*, Toronto, Canada, April 2014.

[11] G. Liang and U. C. Kozat, "TOFEC: Achieving optimal throughput-delay trade-off of cloud storage using erasure codes," in *Proc. IEEE International Conference on Computer Communications (INFOCOM)*, Toronto, Canada, April 2014.

[12] Y. Sun, Z. Zheng, C. E. Koksal, K. Kim, and N. B. Shroff, "Probably delay efficient data retrieving in storage clouds," in *Proc. IEEE International Conference on Computer Communications (INFOCOM)*, Hong Kong, China, April 2015.

[13] K. Gardner, S. Zbarsky, S. Doroudi, M. Harchol-Balter, E. Hyytia, and A. Scheller-Wolf., "Reducing latency via redundant requests: Exact analysis," in *Proc. ACM International Conference on Measurement and Modeling of Computer Systems (SIGMETRICS)*, Portland, OR, USA, June 2015.

[14] N. D. Vvedenskaya, R. L. Dobrushin, and F. I. Karpelevich, "Queueing system with selection of the shortest of two queues: An asymptotic approach," *Problemy Peredachi Informatsii*, vol. 32, no. 1, pp. 20–34, 1996.

[15] M. Mitzenmacher, *The power of two choices in randomized load balancing*. Ph.D. Thesis, University of California at Berkeley, 1996.

[16] M. Bramson, Y. Lu, and B. Prabhakar, "Randomized load balancing with general service time distributions," in *Proc. ACM International Conference on Measurement and Modeling of Computer Systems (SIGMETRICS)*, New York, NY, USA, June 2010.

[17] Y. Lu, Q. Xie, G. Kliot, A. Geller, J. R. Larus, and A. Greenberg, "Join-idle-queue: A novel load balancing algorithm for dynamically scalable web services," *Performance Evaluation*, vol. 68, no. 11, pp. 1056–1071, 2011.

[18] L. Ying, R. Srikant, and X. Kang, "The power of slightly more than one sample in randomized load balancing," in *Proc. IEEE International Conference on Computer Communications (INFOCOM)*, Hong Kong, April 2015.

[19] A. L. Stolyar, "Pull-based load distribution in large-scale heterogeneous service systems," *Queueing Systems*, vol. 80, no. 11, pp. 341–361, 2015.

[20] K. Ousterhout, P. Wendell, M. Zaharia, and I. Stoica, "Sparrow: distributed, low latency scheduling," in *Proc. ACM Symposium on Operating Systems Principles (SOSP)*, Pennsylvania, PA, USA, November 2013.

[21] M. Lugo, *A Note for Stat 134 Fall 2011: The Expectation of the Maximum of Exponentials*. University of California at Berkeley, 2011.

[22] B. Li, A. Ramamoorthy, and R. Srikant, "Technical report for INFOCOM'2016: Mean-field-analysis of coding versus replication in cloud storage systems," *https://www.dropbox.com/s/08a1fx4wrhftxho/CloudStorage2016.pdf?dl=0*, 2016.