Joint Latency and Cost Optimization for Erasure-Coded Data Center Storage

Yu Xiang, Tian Lan, *Member, IEEE*, Vaneet Aggarwal, *Senior Member, IEEE*, and Yih-Farn R. Chen

***Abstract*—Modern distributed storage systems offer large capacity to satisfy the exponentially increasing need of storage space. They often use erasure codes to protect against disk and node failures to increase reliability, while trying to meet the latency requirements of the applications and clients. This paper provides an insightful upper bound on the average service delay of such erasure-coded storage with arbitrary service time distribution and consisting of multiple heterogeneous files. Not only does the result supersede known delay bounds that only work for a single file or homogeneous files, it also enables a novel problem of joint latency and storage cost minimization over three dimensions: selecting the erasure code, placement of encoded chunks, and optimizing scheduling policy. The problem is efficiently solved via the computation of a sequence of convex approximations with provable convergence. We further prototype our solution in an open-source cloud storage deployment over three geographically distributed data centers. Experimental results validate our theoretical delay analysis and show significant latency reduction, providing valuable insights into the proposed latency–cost tradeoff in erasure-coded storage.**

***Index Terms*—Content placement, data center, difference-of-convex programming, distributed storage, erasure code, gradient descent, joint optimization, latency, Pollaczek–Kinchin transform, queueing theory, scheduling.**

I. Introduction

A. Motivation

CONSUMERS are engaged in more social networking and E-commerce activities these days and are increasingly storing their documents and media in the online storage. Businesses are relying on Big Data analytics for business intelligence and are migrating their traditional IT infrastructure to the cloud. These trends cause the online data storage demand to rise faster than Moore's Law [8]. The increased storage demands have led companies to launch cloud storage services like Amazon's S3 [9] and personal cloud storage services like Amazon's Cloud drive, Apple's iCloud, DropBox, Google Drive, Microsoft's SkyDrive, and AT&T Locker. Storing redundant information on distributed servers can increase reliability for storage systems since users can retrieve duplicated pieces in case of disk, node, or site failures.

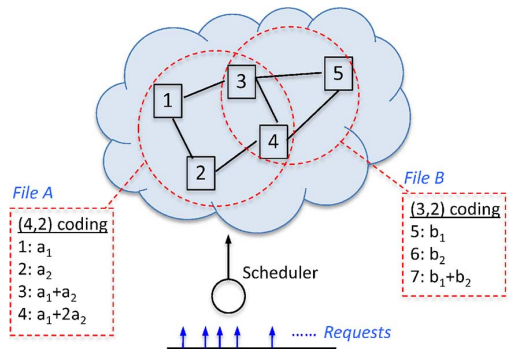


Fig. 1. Erasure-coded storage of 2 files, partitioned into 2 blocks and encoded using (4, 2) and (3, 2) MDS codes, respectively. Resulting file chunks are spread over 5 storage nodes. Any file request must be processed by 2 distinct nodes that have the desired chunks. Nodes 3 and 4 are shared and can process requests for both files.

Erasure coding has been widely studied for distributed storage systems ([12] and references therein) and used by companies like Facebook [10] and Google [11] since it provides space-optimal data redundancy to protect against data loss. There is, however, a critical factor that affects the service quality that the user experiences, which is the delay in accessing the stored file. In distributed storage, the bandwidth between different nodes is frequently limited and so is the bandwidth from a user to different storage nodes, which can cause a significant delay in data access and perceived as poor quality of service. In this paper, we consider the problem of jointly minimizing both service delay and storage cost for the end-users.

While a latency–cost tradeoff is demonstrated for the special case of a single file, or homogeneous files with exactly the same properties (file size, type, coding parameters, etc.) [33], [37], [41], [42], much less is known about the latency performance of multiple heterogeneous files that are coded with different parameters and share common storage servers. The main goal of this paper can be illustrated by an abstracted example shown in Fig. 1. We consider two files, each partitioned into $k = 2$ blocks of equal size and encoded using maximum distance separable (MDS) codes. Under an (n, k) MDS code, a file is encoded and stored in n storage nodes such that the chunks stored in any k of these n nodes suffice to recover the entire file. There is a centralized scheduler that buffers and schedules all incoming requests. For instance, a request to retrieve file A can be completed after it is successfully

Manuscript received August 05, 2014; revised March 09, 2015 and May 18, 2015; accepted July 16, 2015; approved by IEEE/ACM TRANSACTIONS ON NETWORKING Editor L. Ying. Date of publication September 16, 2015; date of current version August 16, 2016. This work was presented in part at the IFIP Performance, Oct. 2014.

Y. Xiang and T. Lan are with Department of Electrical and Computer Engineering, George Washington University, Washington, DC 20052 USA (email: xy336699@gwu.edu; tlan@gwu.edu).

V. Aggarwal was with AT&T Labs—Research, Bedminster, NJ 07921 USA. He is now with the School of Industrial Engineering, Purdue University, West Lafayette, IN 47907 USA (e-mail: vaneet@purdue.edu).

Y. R. Chen is with AT&T Labs—Research, Bedminster, NJ 07921 (e-mail: chen@research.att.com).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TNET.2015.2466453

processed by 2 distinct nodes chosen from $\{1, 2, 3, 4\}$ where desired chunks of A are available. Due to shared storage nodes and joint request scheduling, delay performances of the files are highly correlated and are collectively determined by control variables of both files over three dimensions: 1) the scheduling policy that decides what request in the buffer to process when a node becomes available; 2) the placement of file chunks over distributed storage nodes; and 3) erasure coding parameters that decide how many chunks are created. A joint optimization over these three dimensions is very challenging because the latency performances of different files are tightly entangled. While increasing erasure code length of file B allows it to be placed on more storage nodes, potentially leading to smaller latency (because of improved load-balancing) at the price of higher storage cost, it inevitably affects service latency of file A due to resulting contention and interference on more shared nodes. In this paper, we present a quantification of service latency for erasure-coded storage with multiple heterogeneous files and propose an efficient solution to the joint optimization of both latency and storage cost.

B. Related Work

The effect of coding on content retrieval latency in data-center storage systems is drawing more and more significant attention these days, as Google and Amazon have published that every 500 ms extra delay means a 1.2% user loss [1]. However, to our best knowledge quantifying the exact service delay in an erasure-coded storage system is an open problem, prior works focusing on asymptotic queuing delay behaviors [34], [35] are not applicable because redundancy factor in practical data centers typically remain small due to storage cost concerns. Due to the lack of analytical latency models for erasure-coded storage, most of the literature is focused on reliable distributed storage system design, and latency is only presented as a performance metric when evaluating the proposed erasure coding scheme, e.g., [20], [22], [24], [25], and [26], which demonstrate latency improvement due to erasure coding in different system implementations. Related design can also be found in data access scheduling [13], [15], [18], [19], access collision avoidance [16], [17], and encoding/decoding time optimization [27], [28], and there are also some works using the LT erasure codes to adjust the system to meet user requirements such as availability, integrity, and confidentiality [6]. Restricting to the special case of a single file or homogeneous files, service delay bounds of erasure-coded storage have been recently studied in [33], [37], [41], and [42].

Queuing-Theoretic Analysis: For a single file or multiple but homogeneous files, under an assumption of exponential service time distribution, the authors in [5] proved an asymptotic result for symmetric large-scale systems that can be applied to provide a computable approximation for expected latency, however, under a assumption that chunk placement is fixed and so is coding policy for all requests, which is not the case in reality. Also, the authors in [33] and [37] proposed a *block-one-scheduling* policy that only allows the request at the head of the buffer to move forward. An upper bound on the average latency of the storage system is provided through queuing-theoretic analysis for MDS codes with $k = 2$. Later, the approach is extended in [41] to general (n, k) erasure codes, yet for a single file or homogeneous files. A family of *MDS-Reservation*(t) scheduling policies that block all except the first t of file requests are proposed and lead to numerical upper bounds on the average latency. It is shown that as t increases, the bound becomes tighter while the number of states concerned in the queueing-theoretic analysis grows exponentially.

Fork-Join Queue Analysis: A queuing model closely related to erasure-coded storage is the fork-join queue [14], which has been extensively studied in the literature. Recently, in [2], the authors proposed a heuristic transmission scheme using this Fork-join queuing model where a file request is forked to all n storage nodes that host the file chunks, and it exits the system when any k chunks are processed to dynamically tuning coding parameters to improve latency performance. In [4], the authors proposed a self-adaptive strategy that can dynamically adjust chunk size and number of redundancy requests according to dynamic workload status in erasure-coded storage systems to minimize queuing delay in fork-join queues. Also, the authors in [42] used this (n, k) fork-join queue to model the latency performance of erasure-coded storage, a closed-form upper bound of service latency is derived for the case of a single file or homogeneous files and exponentially distributed service time. However, the approach cannot be applied to a multiple-heterogeneous file storage where each file has a separate folk-join queue and the queues of different files are highly dependent due to shared storage nodes and joint request scheduling. In another work [3], the authors applied this fork-join queue to optimize threads allocation to each request, which is similar to our weighted queue model, however, both proposed greedy/shared scheme would waste system resources because in fork-join queue there will always be some threads have unfinished downloads due to redundant assignment. In addition, in [7], the authors proposed a distributed storage system that analyzed through the (n, k) Fork-join queue framework with heterogeneous jobs, and provide lower and upper bounds on the average latency for jobs of different classes under various scheduling policies, such as First-Come-First-Serve, preemptive and non-preemptive priority scheduling policies, based on the analysis of mean and second moment of waiting time. However, under a folk-join queue, each file request must be served by all n nodes or a set of prespecified nodes. It falls short to address dynamic load-balancing of multiple heterogeneous files.

C. Our Contributions

This paper aims to propose a systematic framework that: *1) quantifies the outer bound on the service latency of arbitrary erasure codes and for any number of files* in distributed data center storage with general service time distributions; and *2) enables a novel solution to a joint minimization of latency and storage cost* by optimizing the system over three dimensions: erasure coding, chunk placement, and scheduling policy.

The outer bound on the service latency is found using four steps.

1) We present a novel *probabilistic scheduling* policy, which dispatches each file request to k distinct storage nodes who then manages their own local queues independently. A file request exits the system when all the k chunk requests are processed. We show that probabilistic scheduling provides an upper bound on average latency of erasure-coded storage for arbitrary erasure codes, any number of files, and general service time distributions.

2) Then, we show that the probabilistic scheduling is equivalent to accessing each of the n storage nodes with certain probability. If there is a strategy that accesses each storage node with certain probability, there exist a probabilistic scheduling strategy over all $\binom{n}{k}$ subsets.
3) The policy that selects each storage node with certain probability generates memoryless requests at each of the nodes and thus the delay at each storage node can be characterized by the latency of M/G/1 queue.
4) Knowing the exact delay from each storage node, we find a tight bound on the delay of the file by extending ordered statistic analysis in [36]. Not only does our result supersede previous latency analysis [33], [37], [41], [42] by incorporating multiple heterogeneous files and arbitrary service time distribution, it is also shown to be tighter for a wide range of workloads even in the single-file or homogeneous-files case.

Multiple extensions to the outer bound on the service latency are considered. The first is the case when multiple chunks can be placed on the same node. As a result, multiple chunk requests corresponding to the same file request can be submitted to the same queue, which processes the requests sequentially and results in dependent chunk service times. The second is the case when the file can be retrieved from more than k nodes. In this case, smaller amount of data can be obtained from more nodes. Obtaining data from more nodes has an effect of considering worst ordered statistics having an effect on increasing latency, while the smaller file size from each of the node helping more parallelism, and thus decreasing latency. The optimal value of the number of disks to access can then be optimized.

The main application of our latency analysis is a joint optimization of latency and storage cost for multiple-heterogeneous file storage over three dimensions: erasure coding, chunk placement, and scheduling policy. To the best of our knowledge, this is the first paper to explore all these three design degrees of freedoms and to optimize an aggregate latency-plus-cost objective for all end-users in an erasure-coded storage. Solving such a joint optimization is known to be hard due to the integer property of storage cost, as well as the coupling of control variables. While the length of erasure code determines not only storage cost but also the number of file chunks to be created and placed, the placement of file chunks over storage nodes further dictates the possible options of scheduling future file requests. To deal with these challenges, we propose an algorithm that constructs and computes a sequence of local convex approximations of the latency-plus-cost minimization that is a mixed integer optimization. The sequence of approximations can be efficiently computed using a standard projected gradient method and is shown to converge to the original problem in the end.

To validate our theoretical analysis and joint latency-plus-cost optimization, we provide a prototype of the proposed algorithm in *Tahoe* [40], which is an open-source, distributed filesystem based on the *zfec* erasure coding library for fault tolerance. A Tahoe storage system consisting of 12 storage nodes are deployed as virtual machines in an OpenStack-based data center environment distributed in New Jersey (NJ), Texas (TX), and California (CA). Each site has four storage servers. One additional storage client was deployed in the NJ data center to issue storage requests. First, we validate our latency analysis via experiments with multiple-heterogeneous files and different request arrival rates on the testbed. Our measurement of real service time distribution falsifies the exponential assumption in [33], [37], and [41]. Our analysis outperforms the upper bound in [42] even in the single-file/homogeneous-file case. Second, we implement our algorithm for joint latency-plus-cost minimization and demonstrate significant improvement of both latency and cost over oblivious design approaches. Our entire design is validated in various scenarios on our testbed, including different files sizes and arrival rates. The percentage improvement increases as the file size increases because our algorithm reduces queuing delay, which is more effective when file sizes are larger. Finally, we quantify the tradeoff between latency and storage cost. It is shown that the improved latency shows a diminished return as storage cost/redundancy increase, suggesting the importance of identifying a particular tradeoff point.

II. System Model

We consider a data center consisting of m heterogeneous servers, denoted by $\mathcal{M} = \{1, 2, \ldots, m\}$, called storage nodes. To distributively store a set of r files, indexed by $i = 1, \ldots, r$, we partition each file i into k_i fixed-size chunks[1] and then encode it using an (n_i, k_i) MDS erasure code to generate n_i distinct chunks of the same size for file i. The encoded chunks are assigned to and stored on n_i distinct storage nodes, which leads to a *chunk placement subproblem*, i.e., to find a set $\mathcal{S}_i$ of storage nodes, satisfying $\mathcal{S}_i \subseteq \mathcal{M}$ and $n_i = |\mathcal{S}_i|$, to store file i. Therefore, each chunk is placed on a different node to provide high reliability in the event of node or network failures. While data locality and network delay have been one of the key issues studied in data center scheduling algorithms [18], [19], [21], the prior work does not apply to erasure-coded systems.

The use of (n_i, k_i) MDS erasure code allows the file to be reconstructed from any subset of k_i-out-of-n_i chunks, whereas it also introduces a redundancy factor of n_i/k_i. To model storage cost, we assume that each storage node $j \in \mathcal{M}$ charges a constant cost V_j per chunk. Since k_i is determined by file size and the choice of chunk size, we need to choose an appropriate n_i that not only introduces sufficient redundancy for improving chunk availability, but also achieves a cost-effective solution. We refer to the problem of choosing n_i to form a proper (n_i, k_i) erasure code as an *erasure coding subproblem*.

For known erasure coding and chunk placement, we shall now describe a queueing model of the distributed storage system. We assume that the arrival of client requests for each file i form an independent Poisson process with a known rate λ_i. We consider chunk service time $\mathbf{X}_j$ of node j with *arbitrary distributions*, whose statistics can be obtained inferred from existing work on network delay [28], [29] and file-size distribution [30], [31]. Under MDS codes, each file i can be retrieved from any k_i distinct nodes that store the file chunks. We model this by treating each file request as a *batch* of k_i chunk requests, so that a file request is served when all k_i chunk requests in the batch are processed by distinct storage nodes. All requests are buffered in a common queue of infinite capacity.

[1]While we make the assumption of fixed chunk size here to simplify the problem formulation, all results in this paper can be easily extended to variable chunk sizes. Nevertheless, fixed chunk sizes are indeed used by many existing storage systems [20], [22], [23].

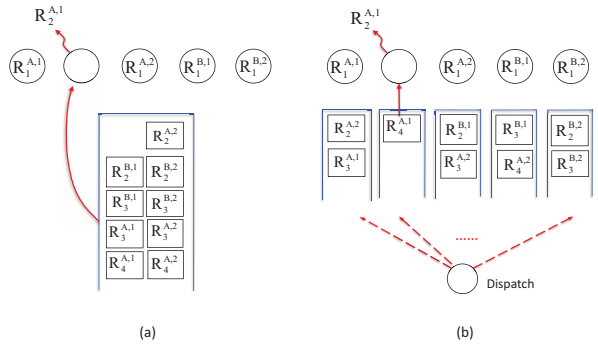


Fig. 2. Functioning of (a) an optimal scheduling policy and (b) a probabilistic scheduling policy. (a) MDS scheduling, (b) Probabilistic scheduling.

Consider the 2-file storage example in Section I, where files A and B are encoded using $(4,2)$ and $(3,2)$ MDS codes, respectively, file A will have chunks as A_1, A_2, A_3, and A_4, and file B will have chunks B_1, B_2, and B_3. As depicted in Fig. 2(a), each file request comes in as a batch of $k_i = 2$ chunk requests, e.g., $(R_1^{A,1}, R_1^{A,2})$, $(R_2^{A,1}, R_2^{A,2})$, and $(R_1^{B,1}, R_1^{B,2})$, where $R_i^{A,j}$, denotes the ith request of file A, $j = 1, 2$ denotes the first or second chunk request of this file request. Denote the five nodes (from left to right) as servers 1, 2, 3, 4, and 5, and we initialize 4 file requests for file A and 3 file requests for file B, i.e., requests for the different files have different arrival rates. The two chunks of one file request can be any two different chunks from A_1, A_2, A_3, and A_4 for file A and B_1, B_2, and B_3 for file B. Due to chunk placement in the example, any 2 chunk requests in file A's batch must be processed by 2 distinct nodes from $\{1, 2, 3, 4\}$, while 2 chunk requests in file B's batch must be served by 2 distinct nodes from $\{3, 4, 5\}$. Suppose that the system is now in a state depicted by Fig. 2(a), wherein the chunk requests $R_1^{A,1}, R_2^{A,1}, R_1^{A,2}, R_1^{B,1}$, and $R_2^{B,2}$ are served by the 5 storage nodes, and there are 9 more chunk requests buffered in the queue. Suppose that node 2 completes serving chunk request $R_2^{A,1}$ and is now free to serve another request waiting in the queue. Since node 2 has already served a chunk request of batch $(R_2^{A,1}, R_2^{A,2})$ and node 2 does not host any chunk for file B, it is not allowed to serve either $R_2^{A,2}$ or $R_2^{B,j}, R_3^{B,j}$, where $j = 1, 2$ in the queue. One of the valid requests, $R_3^{A,j}$ and $R_4^{A,j}$, will be selected by a scheduling algorithm and assigned to node 2. We denote the scheduling policy that minimizes average expected latency in such a queuing model as *optimal scheduling*.

Definition 1 (Optimal Scheduling): An optimal scheduling policy: 1) buffers all requests in a queue of infinite capacity; 2) assigns at most 1 chunk request from a batch to each appropriate node; and 3) schedules requests to minimize average latency if multiple choices are available.

An exact analysis of optimal scheduling is extremely difficult. Even for given erasure codes and chunk placement, it is unclear what scheduling policy leads to minimum average latency of multiple heterogeneous files. For example, when a shared storage node becomes free, one could schedule either the earliest valid request in the queue or the request with scarcest availability, leading to different implications on average latency. A scheduling policy similar to [33] and [37] that blocks all but the first t batches does not apply to multiple heterogeneous files because a Markov-chain representation of the resulting queue is required to have each state encapsulating not only the status of each batch in the queue, but also the exact assignment of chunk requests to storage nodes, since nodes are shared by multiple files and are no longer homogeneous. This leads to a Markov chain that has a huge state space and is hard to quantify analytically even for small t. On the other hand, the approach relying on (n, k) fork-join queue in [42] also falls short because each file request must be forked to n_i servers, inevitably causing conflict at shared servers.

III. Upper Bound: Probabilistic Scheduling

This section presents a class of scheduling policies (and resulting latency analysis), which we call the probabilistic scheduling, whose average latency upper-bounds that of optimal scheduling.

A. Probabilistic Scheduling

Under (n_i, k_i) MDS codes, each file i can be retrieved by processing a batch of k_i chunk requests at distinct nodes that store the file chunks. Recall that each encoded file i is spread over n_i nodes, denoted by a set $\mathcal{S}_i$. Upon the arrival of a file i request, in probabilistic scheduling we randomly dispatch the batch of k_i chunk requests to k_i out of n_i storage nodes in $\mathcal{S}_i$, denoted by a subset $\mathcal{A}_i \subseteq \mathcal{S}_i$ (satisfying $|\mathcal{A}_i| = k_i$) with predetermined probabilities. Then, each storage node manages its local queue independently and continues processing requests in order. A file request is completed if all its chunk requests exit the system. An example of probabilistic scheduling is depicted in Fig. 2(b), wherein 5 chunk requests are currently served by the 5 storage nodes, and there are 9 more chunk requests that are randomly dispatched to and are buffered in 5 local queues according to chunk placement, e.g., requests B_2, B_3 are only distributed to nodes $\{3, 4, 5\}$. Suppose that node 2 completes serving chunk request A_2. The next request in the node's local queue will move forward.

Definition 2 (Probabilistic Scheduling): A probabilistic scheduling policy: 1) dispatches each batch of chunk requests to appropriate nodes with predetermined probabilities; 2) each node buffers requests in a local queue and processes in order.

It is easy to verify that such probabilistic scheduling ensures that at most 1 chunk request from a batch to each appropriate node. It provides an upper bound on average service latency for the optimal scheduling since rebalancing and scheduling of local queues are not permitted. Let $\mathbb{P}(\mathcal{A}_i)$ for all $\mathcal{A}_i \subseteq \mathcal{S}_i$ be the probability of selecting a set of nodes $\mathcal{A}_i$ to process the $|\mathcal{A}_i| = k_i$ distinct chunk requests.[2]

Lemma 1: For given erasure codes and chunk placement, average service latency of probabilistic scheduling with feasible probabilities $\{\mathbb{P}(\mathcal{A}_i) : \forall i, \mathcal{A}_i\}$ upper-bounds the latency of optimal scheduling.

Clearly, the tightest upper bound can be obtained by minimizing average latency of probabilistic scheduling over all feasible probabilities $\mathbb{P}(\mathcal{A}_i)$ $\forall \mathcal{A}_i \subseteq \mathcal{S}_i$ and $\forall i$, which involves $\sum_i \binom{n_i}{k_i}$ decision variables. We refer to this optimization as a *scheduling subproblem*. While it appears prohibitive computationally, we will demonstrate next that the optimization can be transformed into an equivalent form, which only requires $\sum_i n_i$

[2]It is easy to see that $\mathbb{P}(\mathcal{A}_i) = 0$ for all $\mathcal{A}_i \nsubseteq \mathcal{S}_i$ and $|\mathcal{A}_i| = k_i$ because such node selections do not recover k_i distinct chunks and thus are inadequate for successful decode.

variables. The key idea is to show that it is sufficient to consider the conditional probability (denoted by $\pi_{i,j}$) of selecting a node j, given that a batch of k_i chunk requests of file i are dispatched. It is easy to see that for given $\mathbb{P}(\mathcal{A}_i)$, we can derive $\pi_{i,j}$ by

$$\pi_{i,j} = \sum_{\mathcal{A}_i:\mathcal{A}_i\subseteq\mathcal{S}_i} \mathbb{P}(\mathcal{A}_i)\cdot \mathbf{1}_{\{j\in\mathcal{A}_i\}} \qquad \forall i \tag{1}$$

where $\mathbf{1}_{\{j\in\mathcal{A}_i\}}$ is an indicator function that equals to 1 if node j is selected by $\mathcal{A}_i$, and 0 otherwise.

Theorem 1: A probabilistic scheduling policy with feasible probabilities $\{\mathbb{P}(\mathcal{A}_i):\ \forall i, \mathcal{A}_i\}$ exists if and only if there exists conditional probabilities $\{\pi_{i,j}\in[0,1], \forall i,j\}$ satisfying

$$\sum_{j=1}^{m}\pi_{i,j} = k_i \quad \forall i \quad \text{and} \quad \pi_{i,j} = 0 \text{ if } j\notin\mathcal{S}_i. \tag{2}$$

The proof of Theorem 1 relying on Farkas–Minkowski Theorem [44] is detailed in Appendix-A. Intuitively, $\sum_{j=1}^{m}\pi_{i,j} = k_i$ holds because each batch of requests is dispatched to exact k_i distinct nodes. Moreover, a node that does not host file i chunks should not be selected, meaning that $\pi_{i,j}=0$ if $j\notin\mathcal{S}_i$. Using this result, it is sufficient to study probabilistic scheduling via conditional probabilities $\pi_{i,j}$, which greatly simplifies our analysis. In particular, it is easy to verify that under our model, the arrival of chunk requests at node j forms a Poisson Process with rate $\Lambda_j = \sum_i \lambda_i\pi_{i,j}$, which is the superposition of r Poisson processes each with rate $\lambda_i\pi_{i,j}$, μ_j is the service rate of node j. The resulting queuing system under probabilistic scheduling is stable if all local queues are stable.

Corollary 1: The queuing system can be stabilized by a probabilistic scheduling policy under request arrival rates $\lambda_1,\lambda_2,\ldots,\lambda_r$ if there exists $\{\pi_{i,j}\in[0,1],\forall i,j\}$ satisfying (2) and

$$\Lambda_j = \sum_i \lambda_i\pi_{i,j} < \mu_j \qquad \forall j. \tag{3}$$

B. Latency Analysis and Upper Bound

An exact analysis of the queuing latency of probabilistic scheduling is still hard because local queues at different storage nodes are dependent of each other as each batch of chunk requests are dispatched jointly. Let $\mathbf{Q}_j$ be the (random) waiting time a chunk request spends in the queue of node j. The expected latency of a file i request is determined by the maximum latency that k_i chunk requests experience on distinct servers, $\mathcal{A}_i\subseteq\mathcal{S}_i$, which are randomly scheduled with predetermined probabilities, i.e.,

$$\bar{T}_i = \mathbb{E}\left[\mathbb{E}_{\mathcal{A}_i}\left(\max_{j\in\mathcal{A}_i}\{\mathbf{Q}_j\}\right)\right] \tag{4}$$

where the first expectation is taken over system queuing dynamics and the second expectation is taken over random dispatch decisions $\mathcal{A}_i$.

If the server scheduling decision $\mathcal{A}_i$ were deterministic, a tight upper bound on the expected value of the highest-order statistic can be computed from marginal mean and variance of these random variables [36], namely $\mathbb{E}[\mathbf{Q}_j]$ and $\mathrm{Var}[\mathbf{Q}_j]$. Relying on Theorem 1, we first extend this bound to the case of randomly selected servers with respect to conditional probabilities $\{\pi_{i,j}\in[0,1],\forall i,j\}$ to quantify the latency of probabilistic scheduling.

Lemma 2: The expected latency $\bar{T}_i$ of file i under probabilistic scheduling is upper-bounded by

$$\bar{T}_i \le \min_{z\in\mathbb{R}}\left\{ z + \sum_{j\in\mathcal{S}_i}\frac{\pi_{i,j}}{2}\left(\mathbb{E}[\mathbf{Q}_j]-z\right) + \sum_{j\in\mathcal{S}_i}\frac{\pi_{i,j}}{2}\left[\sqrt{(\mathbb{E}[\mathbf{Q}_j]-z)^2+\mathrm{Var}[\mathbf{Q}_j]}\right]\right\}. \tag{5}$$

The bound is tight in the sense that there exists a distribution of $\mathbf{Q}_j$ such that (5) is satisfied with exact equality.

Next, we realize that the arrival of chunk requests at node j form a Poisson Process with superpositioned rate $\Lambda_j = \sum_i \lambda_i\pi_{i,j}$. The marginal mean and variance of waiting time $\mathbf{Q}_j$ can be derived by analyzing them as separate M/G/1 queues. We denote $\mathbf{X}_j$ as the service time per chunk at node j, which has an arbitrary distribution satisfying finite mean $\mathbb{E}[\mathbf{X}_j] = 1/\mu_j$, variance $\mathbb{E}[\mathbf{X}_j^2]-\mathbb{E}[\mathbf{X}_j]^2 = \sigma_j^2$, second moment $\mathbb{E}[\mathbf{X}_j^2]=\Gamma_j^2$, and third moment $\mathbb{E}[\mathbf{X}_j^3]=\hat{\Gamma}_j^3$. These statistics can be readily inferred from existing work on network delay [29], [28] and file-size distribution [30], [31].

Lemma 3: Using Pollaczek–Khinchin transform [37], expected delay and variance for total queueing and network delay are given by

$$\mathbb{E}[\mathbf{Q}_j] = \frac{1}{\mu_j} + \frac{\Lambda_j\Gamma_j^2}{2(1-\rho_j)} \tag{6}$$

$$\mathrm{Var}[\mathbf{Q}_j] = \sigma_j^2 + \frac{\Lambda_j\hat{\Gamma}_j^3}{3(1-\rho_j)} + \frac{\Lambda_j^2\Gamma_j^4}{4(1-\rho_j)^2} \tag{7}$$

where $\rho_j = \Lambda_j/\mu_j$ is the request intensity at node j.

Combining Lemmas 2 and 3, a tight upper bound on expected latency of file i under probabilistic scheduling can be obtained by solving a single-variable minimization problem over real $z\in\mathbb{R}$ for given erasure codes n_i, chunk placement $\mathcal{S}_i$, and scheduling probabilities π_{ij}.

Remark 1: Consider the homogeneous case studied in previous work [3], [33], [37], [42] where all nodes have the same service time distribution and where files have the same chunk placement (i.e., $|\mathcal{S}_i| = n_i = m\ \forall i$). It is easy to show that due to symmetry, the optimal scheduling probabilities $\pi_{i,j}$ minimizing total system latency is $\pi_{i,j} = k_i/m$ for all i,j. Therefore, each node j receives an equal request arrival rate Λ_j, resulting in equal mean and variance of waiting time $\mathbf{Q}_j$. Using the convexity of our bound with respect to z, the latency upper bound in (5) can be derived in closed form

$$\bar{T}_i \le \mathbb{E}[\mathbf{Q}_j] + \sqrt{k_i-1}\cdot\mathrm{Var}[\mathbf{Q}_j] \tag{8}$$

where $\mathbb{E}[\mathbf{Q}_j]$ and $\mathrm{Var}[\mathbf{Q}_j]$ are mean and variance of waiting time $\mathbf{Q}_j$ given by (6) and (7).

C. Extensions of the Latency Upper Bound

In the above upper bound, we assumed that each file i uses (n_i,k_i) MDS code, places exactly one chunk on each selected node, and is retrieved from k_i out of n_i nodes on which the file is placed. In practice, more complicated storage schemes can be designed to offer a higher degree of elasticity by: 1) placing multiple chunks on selected nodes; or 2) accessing the file from more than k_i nodes in parallel. In this section, we further extend our latency upper bound to address these cases.

Placing Multiple Chunks on Each Node: This case arises when a group of storage nodes share a single bottleneck (e.g., outgoing bandwidth at a regional datacenter) and must be modeled by a single queue, or in small clusters where the number of storage nodes is less than that of created file chunks (i.e., $n_i > m$). As a result, multiple chunk requests corresponding to the same file request can be submitted to the same queue, which processes the requests sequentially and results in dependent chunk waiting times.

To extend our latency bound, we assume that each node can host up to c chunks. Thus, our probabilistic scheduling policy dispatches (random) $\mathbf{x}$ chunk requests of file i to node j with predetermined probability $\mathbb{P}(\mathbf{x} = x) = \hat{\pi}^x_{i,j}$ for $x = 0, \ldots, c$. Since $\sum_x x\hat{\pi}^x_{i,j}$ represents the average number of chunks retrieved from node j, we must have $\sum_j \sum_x x\hat{\pi}^x_{i,j} = k_i$ to guarantee access to enough chunks for successful file retrieval. Furthermore, to maintain the Poisson arrival in our queuing model, we group these $\mathbf{x}$ chunk requests into a "super-chunk" request. The service time of this "super-chunk" request is given by $\mathbf{X}'_j = \sum_{\ell=1}^{x} \mathbf{X}^\ell_j$, where $\mathbf{X}^1_j, \mathbf{X}^2_j, \ldots$ are *i.i.d.* chunk service times of node j as before and $\mathbf{x}$ follows distribution $\hat{\pi}^x_{i,j}$. Therefore, we can find the mean, second, and third moment of the new service time $\mathbf{X}'_j$, denoted by $1/\mu'_j, \Gamma^2_j$ and $\hat{\Gamma'}^3_j$, respectively. Under this model, the request queue at each storage server j can still be modeled as separate M/G/1 queues, the latency of each file i can be characterized by the following lemma.

Lemma 4: The expected latency $\bar{T}_i$ of file i is upper-bounded by

$$\bar{T}_i \leq \min_{z\in\mathbb{R}} \left\{ z + \sum_{j\in\mathcal{S}_i} \sum_{x=1}^{c} \frac{\hat{\pi}^x_{i,j}}{2} \left(\mathbb{E}[\hat{\mathbf{Q}}^x_{ij}] - z \right) + \sum_{j\in\mathcal{S}_i} \sum_{x=1}^{c} \frac{\hat{\pi}^x_{i,j}}{2} \left[\sqrt{(\mathbb{E}[\hat{\mathbf{Q}}^x_{ij}] - z)^2 + \mathrm{Var}[\hat{\mathbf{Q}}^x_{ij}]} \right] \right\} \quad (9)$$

where $\hat{\mathbf{Q}}^x_{ij}$ is the waiting time for all x chunk request of file i submitted together to the queue of node j, with moments given by

$$\mathbb{E}\left[\hat{\mathbf{Q}}^x_{ij}\right] = \frac{x}{\mu_j} + \frac{\Lambda'_j \Gamma'^2_j}{2(1-\rho'_j)} \quad (10)$$

$$\mathrm{Var}\left[\hat{\mathbf{Q}}^x_{ij}\right] = x\sigma^2_j + \frac{\Lambda'_j \hat{\Gamma}'^3_j}{3(1-\rho'_j)} + \frac{\Lambda'^2_j \Gamma'^4_j}{4(1-\rho'_j)^2} \quad (11)$$

where $\Lambda'_j = \sum_i \sum_{x=1}^{c} x\lambda_i \hat{\pi}^x_{i,j}$ is the equivalent request arrival and $\rho'_j = \Lambda'_j/\mu'_j$ is the equivalent intensity at node j.

The proof is very similar to that of Lemma 3, recognizing that each batch of chunk requests can be considered as a single "super-chunk" request in this case, with service time $\mathbf{X}'_j = \sum_{\ell=1}^{x} \mathbf{X}^\ell_j$. Thus, the arrival of these "super-chunk" requests at each server queue form a Poisson process with a new service time distribution. Finally, for all $\mathbf{x}$ chunk requests in the same batch, only the latency of last request (i.e., waiting time in the queue plus processing time of all $\mathbf{x}$ chunk requests in the batch) has to be considered in the order statistic analysis because it strictly dominates the queuing latency of other $\mathbf{x} - 1$ requests. Using the *i.i.d.* property of service times and updating equivalent request arrival rate $\Lambda'_j = \sum_i \sum_x x\lambda_i \hat{\pi}^x_{i,j}$, the proof of Lemma 4 is straightforward.

Remark: Retrieving File From More Than k_i Nodes: Let F_i be the size of file i. We now consider the scenario where files have different chunk sizes and where each file i can be obtained from $d_i \geq k_i$ nodes, requiring only F_i/d_i amount of data from each node. The scheme allows a higher degree of parallelism in file access. Since less content is requested from each node, it may lead to lower service latency at the cost of accessing more nodes and more complicated coding strategy. We note that in this case the latency bound and its queuing analysis is the same as when (n_i, d_i) MDS code is applied.

IV. Application: Joint Latency and Cost Optimization

In this section, we address the following questions: What is the optimal tradeoff point between latency and storage cost for a erasure-coded system? While any optimization regarding exact latency is an open problem, the analytical upper bound using probabilistic scheduling enables us to formulate a novel optimization of joint latency and cost objectives. Its solution not only provides a theoretical bound on the performance of optimal scheduling, but also leads to implementable scheduling policies that can exploit such tradeoff in practical systems.

A. Formulating the Joint Optimization

We showed that a probabilistic scheduling policy can be optimization over three sets of control variables: erasure coding parameter n_i, chunk placement $\mathcal{S}_i$, and scheduling probabilities π_{ij}. However, a latency optimization without considering storage cost is impractical and leads to a trivial solution where every file ends up spreading over all nodes. To formulate a joint latency and cost optimization, we assume that storing a single chunk on node j requires cost V_j, reflecting the fact that nodes may have heterogeneous quality of service and thus storage prices. Therefore, total storage cost is determined by both the level of redundancy (i.e., erasure code length n_i) and chunk placement $\mathcal{S}_i$. Under this model, the cost of storing file i is given by $C_i = \sum_{j\in\mathcal{S}_i} V_j$. In this paper, we only consider the storage cost of chunks while network cost would be an interesting future direction.

Let $\hat{\lambda} = \sum_i \lambda_i$ be the total arrival rate, so $\lambda_i/\hat{\lambda}$ is the fraction of file i requests, and average latency of all files is given by $\sum_i (\lambda_i/\hat{\lambda})\bar{T}_i$. Our objective is to minimize an aggregate *latency-cost* objective, i.e.,

$$\begin{aligned} &\min \sum_{i=1}^{r} \frac{\lambda_i}{\hat{\lambda}} \bar{T}_i + \theta \sum_{i=1}^{r} \sum_{j\in\mathcal{S}} V_j \\ &\text{s.t. } (1),\ (2),\ (3),\ (5),\ (6),\ (7). \\ &\text{var. } n_i,\ \pi_{i,j},\ \mathcal{S}_i \in \mathcal{M},\ \forall i, j. \end{aligned} \quad (12)$$

Here, $\theta \in [0, \infty)$ is a tradeoff factor that determines the relative importance of latency and cost in the minimization problem. Varying from $\theta = 0$ to $\theta \to \infty$, the optimization solution to (12) ranges from those minimizing latency to ones that achieve lowest cost.

The joint latency-cost optimization is carried out over three sets of variables: erasure code n_i, scheduling probabilities $\pi_{i,j}$, and chunk placement $\mathcal{S}_i$, subject to the constraints derived in Section III. Varying θ, the optimization problem allows service providers to exploit a latency–cost tradeoff and to determine the optimal operating point for different application demands. We

plug into (12) the results in Section III and obtain a Joint Latency-Cost Minimization (JLCM) with respect to probabilistic scheduling[3]:

Problem JLCM:

$$\min\ z+\sum_{j=1}^{m}\frac{\Lambda_j}{2\hat{\lambda}}\left[X_j+\sqrt{X_j^2+Y_j}\right]+\theta\sum_{i=1}^{r}\sum_{j\in\mathcal{S}_i}V_j \tag{13}$$

$$\text{s.t. } X_j=\frac{1}{\mu_j}+\frac{\Lambda_j\Gamma_j^2}{2(1-\rho_j)}-z \qquad \forall j \tag{14}$$

$$Y_j=\sigma_j^2+\frac{\Lambda_j\hat{\Gamma}_j^3}{3(1-\rho_j)}+\frac{\Lambda_j^2\Gamma_j^4}{4(1-\rho_j)^2} \qquad \forall j \tag{15}$$

$$\rho_j=\Lambda_j/\mu_j<1;\quad \Lambda_j=\sum_{i=1}^{r}\pi_{i,j}\lambda_i \qquad \forall j \tag{16}$$

$$\sum_{j=1}^{m}\pi_{i,j}=k_i;\quad \pi_{i,j}\in[0,1];\quad \pi_{i,j}=0 \qquad \forall j\notin\mathcal{S}_i \tag{17}$$

$$|\mathcal{S}_i|=n_i \quad\text{and}\quad \mathcal{S}_i\subseteq\mathcal{M} \qquad \forall i \tag{18}$$

$$\text{var. } z,\ n_i,\ \mathcal{S}_i,\ \pi_{i,j},\ \forall i,j.$$

Problem JLCM is challenging due to two reasons. First, all optimization variables are highly coupled, making it hard to apply any greedy algorithm that iterative optimizes over different sets of variables. The number of nodes selected for chunk placement (i.e., $\mathcal{S}_i$) is determined by erasure code length n_i in (18), while changing chunk placement $\mathcal{S}_i$ affects the feasibility of probabilities $\pi_{i,j}$ due to (17). Second, Problem JLCM is a mixed-integer optimization over $\mathcal{S}_i$ and n_i, and storage cost $C_i=\sum_{j\in\mathcal{S}_i}V_j$ depends on the integer variables. Such a mixed-integer optimization is known to be difficult in general.

B. Constructing Convex Approximations

Next, we develop an algorithmic solution to Problem JLCM by iteratively constructing and solving a sequence of convex approximations. This section shows the derivation of such approximations for any given reference point, while the algorithm and its convergence will be presented later.

Our first step is to replace chunk placement $\mathcal{S}_i$ and erasure coding n_i by indicator functions of $\pi_{i,j}$. It is easy to see that any nodes receiving a zero probability $\pi_{i,j}=0$ should be removed from $\mathcal{S}_i$, since any chunks placed on them do not help reducing latency.

Lemma 5: The optimal chunk placement of Problem JLCM must satisfy $\mathcal{S}_i=\{j:\pi_{i,j}>0\}\ \forall i$, which implies

$$\sum_{j\in\mathcal{S}_i}V_j=\sum_{j=1}^{m}V_j\mathbf{1}_{(\pi_{i,j}>0)} \quad \mathbf{n_i}=\sum_{\mathbf{j=1}}^{\mathbf{m}}\mathbf{1}_{(\pi_{i,j}>0)}. \tag{19}$$

Thus, Problem JLCM becomes to an optimization over only $(\pi_{i,j}\ \forall i,j)$, constrained by $\sum_{j=1}^{m}\pi_{i,j}=k_i$ and $\pi_{i,j}\in[0,1]$ in (17), with respect to the following objective function:

$$z+\sum_{j=1}^{m}\frac{\Lambda_j}{2\hat{\lambda}}\left[X_j+\sqrt{X_j^2+Y_j}\right]+\theta\sum_{i=1}^{r}\sum_{j=1}^{m}V_j\mathbf{1}_{(\pi_{i,j}>0)}. \tag{20}$$

[3]The optimization is relaxed by applying the same axillary variable z to all $\bar{T}_i$, which still satisfies the inequality (5).

However, the indicator functions above that are neither continuous nor convex. To deal with them, we select a fixed reference point $(\pi_{i,j}^{(t)}\ \forall i,j)$ and leverage a linear approximation of (20) with in a small neighbourhood of the reference point. For all i,j, we have

$$V_j\mathbf{1}_{(\pi_{i,j}>0)}\approx\left[V_j\mathbf{1}_{\left(\pi_{i,j}^{(t)}>0\right)}+\frac{V_j(\pi_{i,j}-\pi_{i,j}^{(t)})}{(\pi_{\subset,j}^{(t)}+1/\beta)\log\beta}\right] \tag{21}$$

where $\beta>0$ is a sufficiently large constant relating to the approximation ratio. It is easy to see that the approximation approaches the real cost function within a small neighbourhood of $(\pi_{i,j}^{(t)}\ \forall i,j)$ as β increases. More precisely, when $\pi_{i,j}^{(t)}=0$ the approximation reduces to $\pi_{i,j}(V_j\beta/\log\beta)$, whose gradient approaches infinity as $\beta\to\infty$, whereas the approximation converges to constant V_j for any $\pi_{i,j}^{(t)}=0$ as $\beta\to\infty$.

It is easy to verify that the approximation is linear and differentiable. Therefore, we could iteratively construct and solve a sequence of approximated version of Problem JLCM. Next, we show that the rest of optimization objective in (13) is convex in $\pi_{i,j}$ when all other variables are fixed.

Lemma 6: The following function, in which X_j and Y_j are functions of Λ_j defined by (14) and (15), is convex in Λ_j:

$$F(\Lambda_j)=\frac{\Lambda_j}{2\hat{\lambda}}\left[X_j+\sqrt{X_j^2+Y_j}\right]. \tag{22}$$

C. Algorithm JLCM and Convergence Analysis

Leveraging the linear local approximation in (21) our idea to solve Problem JLCM is to start with an initial $(\pi_{i,j}^{(0)}\ \forall i,j)$, solve its optimal solution, and iteratively improve the approximation by replacing the reference point with an optimal solution computed from the previous step. Lemma 6 shows that such approximations of Problem JLCM are convex and can be solved by off-the-shelf optimization tools, e.g., Gradient Descent Method and Interior Point Method [38].

The proposed algorithm is shown in Fig. 3. For each iteration t, we solve an approximated version of Problem JLCM over $(\pi_{i,j}^{(0)}\ \forall i,j)$ with respect to a given reference point and a fixed parameter z, where $B^{(t)}$ is the objective value. More precisely, for $t=1,2,\ldots$ we solve

$$\begin{aligned}\min\ \mathbf{B^{(t)}}=\theta\sum_{i=1}^{r}\sum_{j=1}^{m}\left[V_j\mathbf{1}_{(\pi_{i,j}^{(t)}>0)}+\frac{V_j(\pi_{i,j}-\pi_{i,j}^{(t)})}{(\pi_{\subset,j}^{(t)}+1/\beta)\log\beta}\right]\\ +z+\sum_{j=1}^{m}\frac{\Lambda_j}{2\hat{\lambda}}\left[X_j+\sqrt{X_j^2+Y_j}\right]\end{aligned}$$

$$\text{s.t. Constraints (14), (15), (16)}$$

$$\sum_{j=1}^{m}\pi_{i,j}=k_i \quad\text{and}\quad \pi_{i,j}\in[0,1]$$

$$\text{var. } \pi_{i,j} \quad \forall i,j. \tag{23}$$

Due to Lemma 6, the above minimization problem with respect to a given reference point has a convex objective function and linear constraints. It is solved by a projected gradient descent routine in Fig. 4. Notice that the updated probabilities $(\pi_{i,j}^{(t)}\ \forall i,j)$ in each step are projected onto the feasibility set $\{\sum_j\pi_{i,j}=k_i,\ \pi_{i,j}\in[0,1],\ \forall i,j\}$ as required by Problem

Algorithm JLCM :

Choose sufficiently large $\beta > 0$
Initialize $t = 0$ and feasible $(z(0), \pi_{i,j}^{(0)}\ \forall i, j)$
Compute current objective value $B^{(0)}$
while $B^{(t)} - B^{(t-1)} > \epsilon$
  Approximate cost function using (21) and $(z(t), \pi_{i,j}^{(t)}\ \forall i, j)$
  Call $projected\_gradient()$ to solve optimization (23)
    $(\pi_{i,j}^{(t+1)}\ \forall i, j) = \arg\min$ (23)
  $z(t+1) = \arg\min$ (23)
  Compute new objective value $B^{(t+1)}$
  Update $t = t + 1$
end while
Find chunk placement $\mathcal{S}_i$ and erasure code n_i by (19)
Output: $(n_i, \mathcal{S}_i, \pi_{i,j}^{(t)})\ \forall i, j$

Fig. 3. Algorithm JLCM: Our proposed algorithm for solving Problem JLCM.

Routine $projected\_gradient()$:

Choose proper stepsize $\delta_1, \delta_2, \delta_3, \ldots$
Initialize $s = 0$ and $\pi_{i,j}^{(s)} = \pi_{i,j}^{(t)}$
while $\sum_{i,j} |\pi_{i,j}^{(s+1)} - \pi_{i,j}^{(s)}| > \epsilon$
  Calculate gradient $\nabla(23)$ with respect to $\pi_{i,j}^{(s)}$
  $\pi_{i,j}^{(s+1)} = \pi_{i,j}^{(s)} + \delta_s \cdot \nabla(23)$
  Project $\pi_{i,j}^{(s+1)}$ onto feasibility set:
    $\{\pi_{i,j}^{(s+1)}:\ \sum_j \pi_{i,j}^{s+1} = k_i,\ \pi_{i,j}^{s+1} \in [0,1],\ \forall i, j\}$
  Update $s = s + 1$
end while
Output: $(\pi_{i,j}^{(s)},\ \forall i, j)$

Fig. 4. Projected Gradient Descent Routine, used in each iteration of Algorithm JLCM.

JLCM using a standard Euclidean projection. It is shown that such a projected gradient descent method solves the optimal solution of Problem (23). Next, for fixed probabilities $(\pi_{i,j}^{(t)}\ \forall i, j)$, we improve our analytical latency bound by minimizing it over $z \in \mathbb{R}$. The convergence of our proposed algorithm is proven in the following theorem.

Theorem 2: Algorithm JLCM generates a descent sequence of feasible points, $\pi_{i,j}^{(t)}$ for $t = 0, 1, \ldots$, which converges to a local optimal solution of Problem JLCM as β grows sufficiently large.

Remark: To prove Theorem 2, we show that Algorithm JLCM generates a series of decreasing objective values $z + \sum_j F(\Lambda_j) + \theta\hat{C}$ of Problem JLCM with a modified cost function

$$\hat{C} = \sum_{i=1}^{r}\sum_{j=1}^{m} V_j \frac{\log(\beta\pi_{i,j} + 1)}{\log\beta}. \tag{24}$$

The key idea in our proof is that the linear approximation of storage cost function in (21) can be seen as a subgradient of $V_j \log(\beta\pi_{i,j} + 1)/\log\beta$, which converges to the real storage cost function as $\beta \to \infty$, i.e.,

$$\lim_{\beta\to\infty} V_j \frac{\log(\beta\pi_{i,j} + 1)}{\log\beta} = V_j \mathbf{1}_{(\pi_{i,j}>0)}. \tag{25}$$

Therefore, a converging sequence for the modified objective $z + \sum_j F(\Lambda_j) + \theta\hat{C}$ also minimizes Problem JLCM, and the optimization gap becomes zero as $\beta \to \infty$. $\hat{h}$ is the linear approximation of storage cost, defined in (44) in the Appendix. Furthermore, it is shown that $\hat{h}$ is a concave function. Thus, minimizing $z + \sum_j F(\Lambda_j) + \hat{h}$ can be viewed as optimizing the difference between 2 convex objectives, namely $z + \sum_j F(\Lambda_j)$ and $-\hat{h}$, which can be also solved via a Difference-of-Convex Programming (DCP). In this context, our linear approximation of cost function in (21) can be viewed as an approximated supergradient in DCP. Please refer to [39] for a comprehensive study of regularization techniques in DCP to speed up the convergence of Algorithm JLCM.

V. Implementation and Evaluation

A. Tahoe Testbed

To validate our proposed algorithms for joint latency and cost optimization (i.e., Algorithm JLCM) and evaluate their performance, we implemented the algorithms in *Tahoe* [40], which is an open-source, distributed filesystem based on the *zfec* erasure coding library. It provides three special instances of a generic *node*: 1) *Tahoe Introducer*: It keeps track of a collection of storage servers and clients and introduces them to each other. 2) *Tahoe Storage Server*: It exposes attached storage to external clients and stores erasure-coded shares. 3) *Tahoe Client*: It processes upload/download requests and connects to storage servers through a Web-based REST API and the Tahoe-LAFS (Least-Authority File System) storage protocol over SSL.

Our algorithm requires customized erasure code, chunk placement, and server selection algorithms. While Tahoe uses a default (10, 3) erasure code, it supports arbitrary erasure code specification statically through a configuration file. In Tahoe, each file is encrypted, and is then broken into a set of segments, where each segment consists of k blocks. Each segment is then erasure-coded to produce n blocks (using an (n, k) encoding scheme) and then distributed to (ideally) n distinct storage servers. The set of blocks on each storage server constitute a chunk. Thus, the file equivalently consists of k chunks that are encoded into n chunks and each chunk consists of multiple blocks.[4] For chunk placement, the Tahoe client randomly selects a set of available storage servers with enough storage space to store n chunks. For server selection during file retrievals, the client first asks all known servers for the storage chunks they might have. Once it knows where to find the needed k chunks (from the k servers that respond the fastest), it downloads at least the first segment from those servers. This means that it tends to download chunks from the "fastest" servers purely based on round-trip times (RTTs). In our proposed JLCM algorithm, we consider RTT plus expected queuing delay and transfer delay as a measure of latency.

In our experiment, we modified the upload and download modules in the Tahoe storage server and client to allow for customized and explicit server selection, which is specified in the configuration file that is read by the client when it starts. In addition, Tahoe performance suffers from its single-threaded design on the client side for which we had to use multiple clients with separate ports to improve parallelism and bandwidth usage during our experiments.

[4]If there are not enough servers, Tahoe will store multiple chunks on one sever. Also, the term "chunk" we used in this paper is equivalent to the term "share" in Tahoe terminology. The number of blocks in each chunk is equivalent to the number of segments in each file.

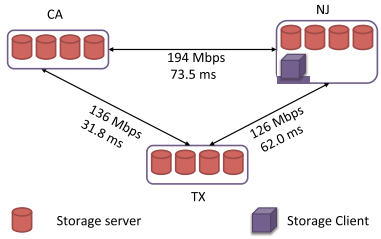


Fig. 5. Our Tahoe testbed with average ping (RTT) and bandwidth measurements among three data centers in New Jersey, Texas, and California.

We deployed 12 Tahoe storage servers as medium-sized virtual machine (VM) instances in an OpenStack-based data center environment distributed in NJ, TX, and CA. Each instance has 2 VCPUs, 2 GB of memory, and a 500 GB volume attached. VMs residing on the same site can be treated as separate storage servers in one data-center. Each site has four storage servers. One additional storage client was deployed in the NJ data center to issue storage requests. The deployment is shown in Fig. 5 with average ping (round-trip time) and bandwidth measurements listed among the three data centers. We note that while the distance between CA and NJ is greater than that of TX and NJ, the maximum bandwidth is higher in the former case. The RTT measured by ping does not necessarily correlate with the bandwidth number. Therefore, this testbed is very representative of real geographically distributed data centers since our theory has network connection delay considered into the model. Furthermore, the current implementation of Tahoe does not use up the maximum available bandwidth, even with our multiport revision.

B. Experiments and Evaluation

Validate Our Latency Analysis: While our service delay bound applies to arbitrary distribution and works for systems hosting any number of files, we first run an experiment to understand the actual service time distribution on our testbed. We uploaded a 50-MB file using a (7, 4) erasure code and measured the chunk service time. Service time depends on the code only because it depends on the size of the chunk. We chose the (7, 4) code as an example code, which together with 50-MB file size gives a chunk size of 12.5 MB. Fig. 6 depicts the cumulative distribution function (CDF) of the chunk service time. Using the measured results, we get the mean service time of 13.9 s with a standard deviation of 4.3 s, second moment of 211.8 s^2, and the third moment of 3476.8 s^3. We compare the distribution to the exponential distribution (with the same mean and the same variance, respectively) and note that the two do not match. It verifies that actual service time does not follow an exponential distribution, and therefore the assumption of exponential service time in [33] and [37] is falsified by empirical data. The observation is also evident because a typical real distribution is unlikely to have positive probabilities for very small service times. Choosing a different code for our tests can result in different chunk sizes, and thus the distribution will be different, but it will still not follow an exponential distribution. Furthermore, the mean and the standard deviation are very different from each other and cannot be matched by any exponential distribution.

Using the service time distribution obtained above, we compare the upper bound on latency that we propose in this paper to

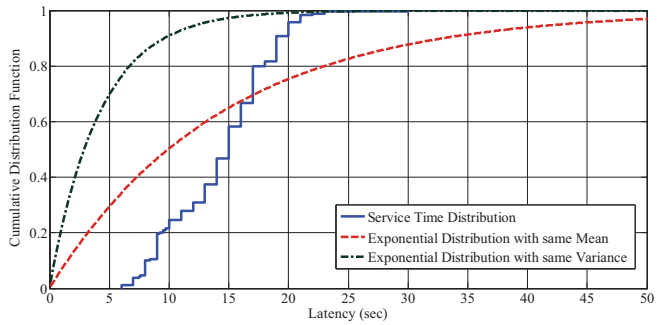


Fig. 6. Comparison of actual service time distribution and an exponential distribution with the same mean. It verifies that actual service time does not follow an exponential distribution, falsifying the assumption in previous work [33], [37].

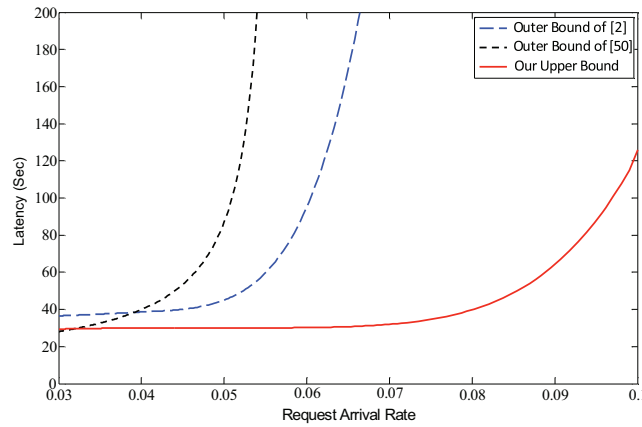


Fig. 7. Comparison of our upper bound on latency to previous work [2] and [42]. Our bound significantly improves previous result under medium to high traffic and comes very close to that of [42] under low traffic (with less than 4% gap).

the outer bound in [2] and [42]. Even though our upper bound holds for multiple heterogeneous files, and includes connection delay, we restrict our comparison to the case for a single file/homogeneous file (multiple homogeneous files with exactly the same properties can be reduced to the case of single file) without any connection delay for a fair comparison (since the upper bound in [42] only works for the case of a single file/homogeneous files). For queuing models, [42] is using a modified (n, k) fork-join queue, where each request is forked to n servers that "store the coded content," and is marked as served when any k chunk requests are served, and the rest of the $n - k$ chunk requests would be abandoned immediately. Reference [2] is using fixed (N, L) erasure code (requests are submitted to L out of N storage servers) with general service time distribution. For a fair comparison in Fig. 7, we use a (7, 4) erasure code for all three of the models. We plot the latency upper bound that we give in this paper and the upper bound in [42, Theorem 3] and [2] in Fig. 7, and mean of service time in the three bounds is set to be equal in this case. In our probabilistic scheduling, access requests are dispatched uniformly to all storage nodes. We find that our bound significantly outperforms the upper bound in [2] and [42] for a wide range of $1/\lambda < 32$ (when comparing to [42]) and $1/\lambda < 24$ (when comparing to [2]), which represents medium to high traffic regime. Under low traffic, the three bounds get very close to each other with a less than 4% gap.

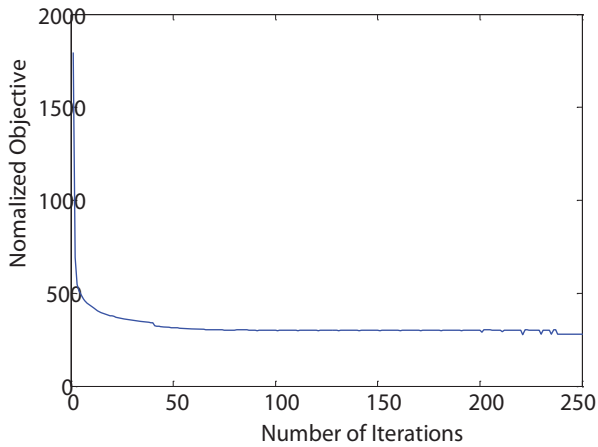


Fig. 8. Convergence of Algorithm JLCM for different problem size with $r = 1000$ files for our 12-node testbed. The algorithm efficiently computes a solution in less than 250 iterations.

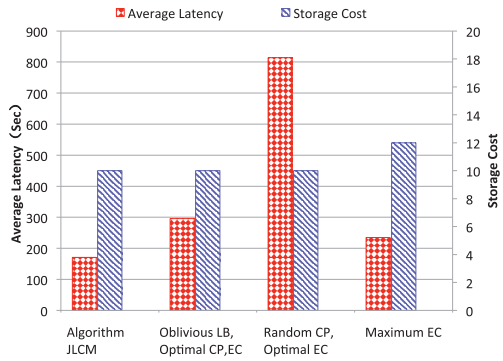


Fig. 9. Comparison of Implementation results of Algorithm JLCM to some oblivious approaches. Algorithm JLCM minimizes latency-plus-cost over three dimensions: LB, CP, and EC, while any optimization over a subset of the dimensions is nonoptimal.

Validate Algorithm JLCM and Joint Optimization: We implemented Algorithm JLCM and used MOSEK [43], a commercial optimization solver, to realize the projected gradient routine. For 12 distributed storage nodes in our testbed, Fig. 8 demonstrates the convergence of Algorithm JLCM, which optimizes latency-plus-cost over three dimensions: erasure code length n_i, chunk placement $\mathcal{S}_i$, and load balancing $\pi_{i,j}$. Convergence of Algorithm JLCM is guaranteed by Theorem 2. To speed up its calculation, in this experiment we merge different updates, including the linear approximation, the latency bound minimization, and the projected gradient update, into one single loop. By performing these updates on the same timescale, our Algorithm JLCM efficiently solves the joint optimization of problem size $r = 1000$ files. It is observed that the normalized objective (i.e., latency-plus-cost normalized by the minimum) converges within 250 iterations for a tolerance $\epsilon = 0.01$, where each iteration has an average run time of 0.81 s, when the algorithm is running on a 8-core machine with i7-3770 CPU, i.e., the algorithm converges within 202 s on average. To achieve dynamic file management, our optimization algorithm can be executed repeatedly upon file arrivals and departures.

To demonstrate the joint latency-plus-cost optimization of Algorithm JLCM, we compare its solution to three oblivious schemes, each of which minimize latency-plus-cost over only a subset of the three dimensions: load-balancing (LB), chunk placement (CP), and erasure code (EC). We implemented the four algorithms for $r = 1000$ files of size 150 MB on our testbed, with $V_j = \$1$ for every 25 MB storage and tradeoff factor of $\theta = 200$ s/dollar for Algorithm JLCM. The result is shown in Fig. 9. First, even with the optimal erasure code and chunk placement (which means the same storage cost as the optimal solution from Algorithm JLCM), higher latency is observed in *Oblivious LB*, which schedules chunk requests according to a load-balancing heuristic that selects storage nodes with probabilities proportional to their service rates. Second, we keep optimal erasure codes and employ a random chunk placement algorithm, referred to as *Random CP*. Large latency increment in the implementation outcome resulted by Random CP highlights the importance of joint chunk placement and load balancing in reducing service latency. Finally, *Maximum EC* uses maximum possible erasure code. For both *Random CP* and *Maximum EC*, we use a round-robin request scheduling policy to select k_i storage nodes for each request. *Maximum EC* approach uses an erasure code with the maximum length (i.e., $n_i = m$) to encode each file into m chunks, so that one encoded chunk is placed on each storage node. While this part is similar to fork-join in [42], to schedule a request, *Maximum EC* employs a round robin strategy to select k_i storage nodes for each request. Although its latency is comparable to the optimal solution from Algorithm JLCM, higher storage cost is observed. We verify that minimum latency-plus-cost can only be achieved by jointly optimizing over all three dimensions.

Evaluate the Performance of Our Solution: First, we choose $r = 1000$ files of size 150 MB and the same storage cost and tradeoff factor as in the previous experiment. The files are divided into four classes (each class has 250 files) with erasure code parameter $k = 6, 7, 6, 4$, respectively (class-1 files using $k = 6$, class-2 files using $k = 7$, class 3 using $k = 6$, and class 4 has $k = 4$). Aggregate request arrival rate for each file class are set to $\lambda_1 = \lambda_4 = 0.0354$/s, $\lambda_2 = \lambda_3 = 0.0236$/s, which leads to an aggregate file request arrival rate of $\lambda = 0.118$/s. We are choosing the k_i values of erasure codes for a proper chunk size for our experiments so that the file sizes are widely used for today's data center storage users, and setting different request arrival rates for the two classes using the same k value to see the performance of JLCM on the storage-latency tradeoff. We obtain the service time statistics (including mean, variance, second and third moment) at all storage nodes and run Algorithm JLCM to generate an optimal latency-plus-cost solution, which results in four different sets of optimal erasure code (12, 6), (10, 7), (10, 6), and (8, 4) for each quarter of the 1000 files, respectively, as well as associated chunk placement and load-balancing probabilities. Implementing this solution on our testbed, we retrieve the 1000 files at the designated request arrival rate and plot the CDF of download latency for each file in Fig. 10. We note that 95% of download requests for files with erasure code (10, 7) complete within 100 s, while the same percentage of requests for files using (12, 6) erasure code complete within 32 s due to higher level of redundancy. In this experiment, erasure code (12, 6) outperforms (8, 4) in latency though they have the same level of redundancy because the latter has larger chunk size when file size are set to be the same.

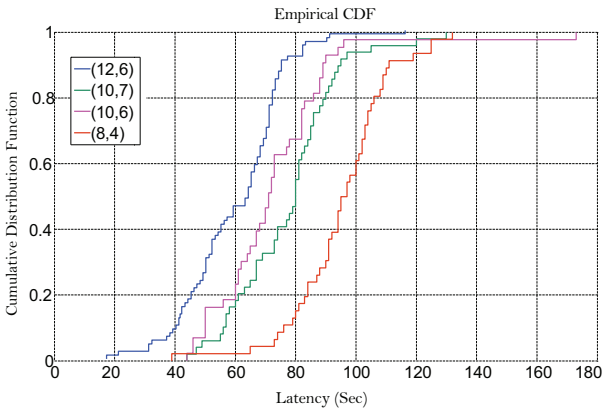


Fig. 10. Actual service latency distribution of an optimal solution from Algorithm JLCM for 1000 files of size 150 MB using erasure code (12, 6), (10, 7), (10, 6), and (8, 4) for each quarter with aggregate request arrival rates set to $\lambda_i = 0.118$/s.

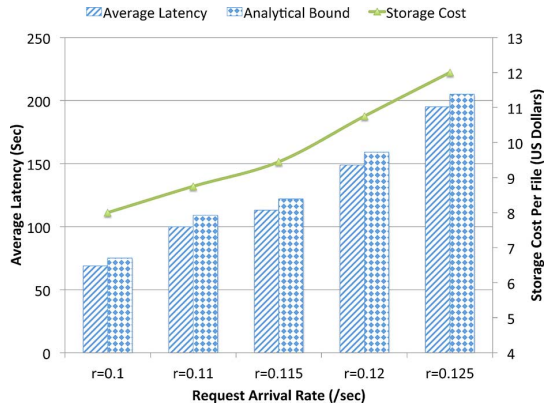


Fig. 12. Evaluation of different request arrival rates. As arrival rates increase, latency increases and becomes more dominating in the latency-plus-cost objective than storage cost. The optimal solution from Algorithm JLCM allows higher storage cost, resulting in a nearly linear growth of average latency.

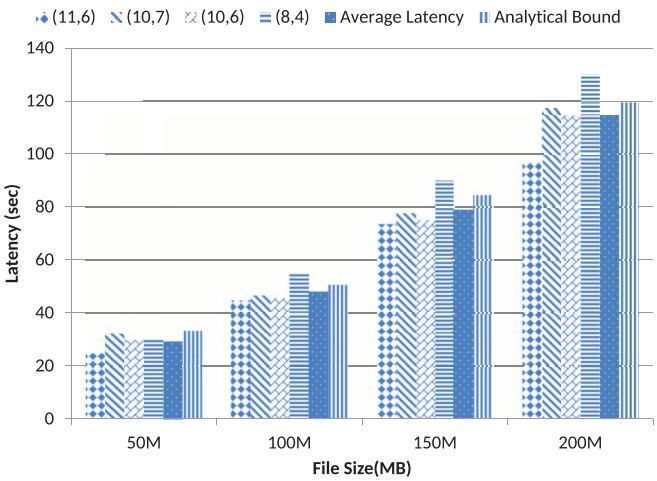


Fig. 11. Evaluation of different chunk sizes. Latency increases super-linearly as file size grows due to queuing delay. Our analytical latency bound taking both network and queuing delay into account tightly follows actual service latency, with error percentage less than 9%.

To demonstrate the effectiveness of our joint optimization, we vary file size in the experiment from 50 to 200 MB and plot the average download latency of the 1000 individual files, out of which each quarter is using a distinct erasure code (12, 6), (10, 7), (10, 6), and (8, 4), and our analytical latency upper bound in Fig. 11. We see that latency increases super-linearly as file size grows since it generates higher load on the storage system, causing larger queuing latency (which is super-linear according to our analysis). Furthermore, smaller files always have lower latency because it is less costly to achieve higher redundancy for these files. We also observe that our analytic latency bound tightly follows actual average service latency, and the average error percentage between the two is no more than 9%.

Next, we varied aggregate file request arrival rate from $\lambda_i = 0.125$/s to $\lambda_i = 0.1$/s (with individual arrival rates also varies accordingly), while keeping tradeoff factor at $\theta = 2$ s/dollar and file size at 200 MB. Actual service delay and our analytical bound for each scenario is shown by a bar plot in Fig. 12 and associated storage cost by a curve plot. Our analytical bound provides a close estimate of service latency. As arrival rates increase, latency increases and becomes more dominating in the latency-plus-cost objective than storage cost. Thus, the marginal benefit of adding more chunks (i.e., redundancy) eventually outweighs higher storage cost introduced at the same time. Fig. 12 shows that to achieve a minimization of the latency-plus-cost objective, the optimal solution from Algorithm JLCM allows higher storage cost for larger arrival rates, resulting in a nearly linear growth of average latency as the request arrival rates increase. For instance, Algorithm JLCM chooses (12, 6), (12, 7), (11, 6), and (11, 4) erasure codes at the largest arrival rates, while (10, 6), (10, 7), (8, 6), and (8, 4) codes are selected at the smallest arrival rates in this experiment. We believe that this ability to autonomously manage latency and storage cost for latency-plus-cost minimization under different workload is crucial for practical distributed storage systems relying on erasure coding. Also our latency bound accurately predicts average service latency, with error percentage less than 10% in this experiment.

Visualize Latency and Cost Tradeoff: Finally, we demonstrate the tradeoff between latency and storage cost in our joint optimization framework. In this experiment, we consider three classes of files, each with a fixed k_i, for $k_i = 6, 7, 4$ respectively. Tradeoff factor θ is very important for system planning. Increasing value of θ means a more important role of storage cost from storage clients, while decreasing value of θ means a more important role of latency. With different values of θ, algorithm JLCM will provide different values of n_i for each class of file with a fixed k_i, i.e., with a small value of θ, algorithm JLCM will provide the user with low latency and relatively high storage cost (larger n_i), and with a large value of θ, implies a result of higher latency and lower storage cost (smaller n_i). Varying the tradeoff factor in Algorithm JLCM from $\theta = 0.5$ s/dollar to $\theta = 200$ s/dollar for fixed file size of 200 MB and aggregate arrival rates $\lambda_i = 0.125$/s, we obtain a sequence of solutions, minimizing different latency-plus-cost objectives. As θ increases, higher weight is placed on the storage cost component of the latency-plus-cost objective, leading to less file chunks in the storage system and higher latency. This tradeoff is visualized in Fig. 13. When $\theta = 0.5$, the optimal solution from Algorithm JLCM chooses three sets of erasure codes (12, 6), (12, 7), and (12, 4), which is the maximum erasure code length in our framework and leads to highest storage cost (i.e., 12 dollars for each

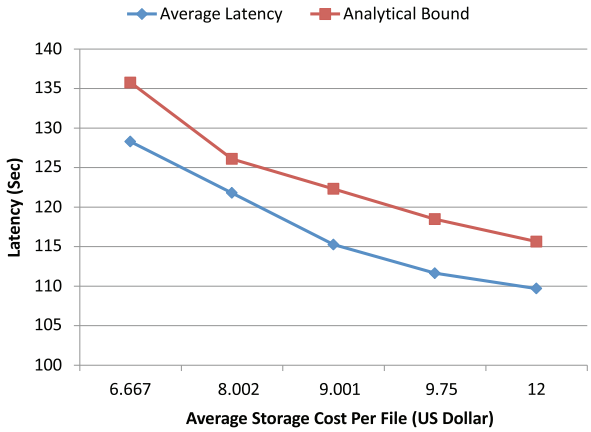


Fig. 13. Visualization of latency and cost tradeoff for varying $\theta = 0.5$ s/dollar to $\theta = 200$ s/dollar. As θ increases, higher weight is placed on the storage cost component of the latency-plus-cost objective, leading to less file chunks and higher latency.

user), yet lowest latency (i.e., 110 s). On the other hand, $\theta = 200$ results in the choice of (6, 6), (8, 7), and (6, 4) erasure code, which is almost the minimum possible cost for storing the three file, with the highest latency of 128 s. Furthermore, the theoretical tradeoff calculated by our analytical bound and Algorithm JLCM is very close to the actual measurement on our testbed. To the best of our knowledge, this is the first work proposing a joint optimization algorithm to exploit such tradeoff in an erasure-coded, distributed storage system.

VI. Conclusion

Relying on a novel probabilistic scheduling policy, this paper develops an analytical upper bound on average service delay of erasure-coded storage with arbitrary number of files and any service time distribution. A joint latency and cost minimization is formulated by collectively optimizing over erasure code, chunk placement, and scheduling policy. The minimization is solved using an efficient algorithm with proven convergence. Even though only local optimality can be guaranteed due to the nonconvex nature of the mixed-integer optimization problem, the proposed algorithm significantly reduces a latency-plus-cost objective. Both our theoretical analysis and algorithm design are validated via a prototype in Tahoe, an open-source distributed file system. Several practical design issues in erasure-coded distributed storage, such as incorporating network cost in the joint optimization and dynamic data management, have been ignored in this paper and open up avenues for future work.

Appendix

A. Proof of Theorem 1

We first prove that the conditions $\sum_{j=1}^{m} \pi_{i,j} = k_i \ \forall i$ and $\pi_{i,j} \in [0,1]$ are necessary. $\pi_{i,j} \in [0,1]$ for all i, j is obvious due to its definition. Then, it is easy to show that

$$\begin{aligned}\sum_{j=1}^{m} \pi_{i,j} &= \sum_{j=1}^{m} \sum_{\mathcal{A}_i \subseteq \mathcal{S}_i} \mathbf{1}_{\{j \in \mathcal{A}_i\}} \mathbb{P}(\mathcal{A}_i) \\ &= \sum_{\mathcal{A}_i \subseteq \mathcal{S}_i} \sum_{j \in \mathcal{A}_i} \mathbb{P}(\mathcal{A}_i) \\ &= \sum_{\mathcal{A}_i \subseteq \mathcal{S}_i} k_i \mathbb{P}(\mathcal{A}_i) = k_i \end{aligned} \tag{26}$$

where the first step is due to (1), $\mathbf{1}_{\{j \in \mathcal{A}_i\}}$ is an indicator function, which is 1 if $j \in \mathcal{A}_i$, and 0 otherwise. The second step changes the order of summation, and the last step uses the fact that each set $\mathcal{A}_i$ contain exactly k_i nodes and that $\sum_{\mathcal{A}_i \subseteq \mathcal{S}_i} \mathbb{P}(\mathcal{A}_i) = 1$.

Next, we prove that for any set of $\pi_{i,1}, \ldots, \pi_{i,m}$ (i.e., node selection probabilities of file i) satisfying $\sum_{j=1}^{m} \pi_{i,j} = k_i$ and $\pi_{i,j} \in [0,1]$, there exists a probabilistic scheduling scheme with feasible load balancing probabilities $\mathbb{P}(\mathcal{A}_i)\ \forall \mathcal{A}_i \subseteq \mathcal{S}_i$ to achieve the same node selection probabilities. We start by constructing $\mathcal{S}_i = \{j : \pi_{i,j} > 0\}$, which is a set containing at least k_i nodes, because there must be at least k_i positive probabilities $\pi_{i,j}$ to satisfy $\sum_{j=1}^{m} \pi_{i,j} = k_i$. Then, we choose erasure code length $n_i = |\mathcal{S}_i|$ and place chunks on nodes in $\mathcal{S}_i$. From (1), we only need to show that when $\sum_{j \in \mathcal{S}_i} \pi_{i,j} = k_i$ and $\pi_{i,j} \in [0,1]$, the following system of n_i linear equations have a feasible solution $\mathbb{P}(\mathcal{A}_i)\ \forall \mathcal{A}_i \subseteq \mathcal{S}_i$

$$\sum_{\mathcal{A}_i \subseteq \mathcal{S}_i} \mathbf{1}_{\{j \in \mathcal{A}_i\}} \cdot \mathbb{P}(\mathcal{A}_i) = \pi_{i,j} \qquad \forall j \in \mathcal{S}_i. \tag{27}$$

We will make use of the following lemma.

Lemma 7: Farkas–Minkowski Theorem [44]. Let $\mathbf{A}$ be an $m \times n$ matrix with real entries, and $\mathbf{x} \in \mathbb{R}^n$ and $\mathbf{b} \in \mathbb{R}^m$ be two vectors. A necessary and sufficient condition that $\mathbf{A} \cdot \mathbf{x} = \mathbf{b},\ \mathbf{x} \geq 0$ has a solution is that, for all $\mathbf{y} \in \mathbb{R}^m$ with the property that $\mathbf{A}^T \cdot \mathbf{y} \geq 0$, we have $\langle \mathbf{y}, \mathbf{b} \rangle \geq 0$.

We prove the desired result using mathematical induction. It is easy to show that the statement holds for $n_i = k_i$. In this case, we have a unique solution $\mathcal{A}_i = \mathcal{S}_i$ and $\mathbb{P}(\mathcal{A}_i) = \pi_{i,j} = 1$ for the system of linear equations (27) because all chunks must be selected to recover file i. Now assume that the system of linear equations (27) has a feasible solution for some $n_i \geq k_i$. Consider the case with arbitrary $|\mathcal{S}_i + \{h\}| = n_i + 1$ and $\pi_{i,h} + \sum_{j \in \mathcal{S}_i} \pi_{i,j} = k_i$. We have a system of linear equations

$$\sum_{\mathcal{A}_i \subseteq \mathcal{S}_i + \{h\}} \mathbf{1}_{\{j \in \mathcal{A}_i\}} \cdot \mathbb{P}(\mathcal{A}_i) = \pi_{i,j} \qquad \forall j \in \mathcal{S}_i + \{h\}. \tag{28}$$

Using the Farkas–Minkowski Theorem [44], a sufficient and necessary condition that (28) has a nonnegative solution is that, for any $y_1, \ldots, y_m$ and $\sum_j y_j \pi_{i,j} < 0$, we have

$$\sum_{j \in \mathcal{S}_i + \{h\}} y_j \mathbf{1}_{\{j \in \mathcal{A}_i\}} < 0 \quad \text{for some } \mathcal{A}_i \subseteq \mathcal{S}_i + \{h\}. \tag{29}$$

Toward this end, we construct $\hat{\pi}_{i,j} = \pi_{i,j} + [u - \pi_{i,j}]^+$ for all $j \in \mathcal{S}_i$. Here, $[x]^+ = \max(x, 0)$ is a truncating function and u is a proper *water-filling level* satisfying

$$\sum_{j \in \mathcal{S}_i} [u - \pi_{i,j}]^+ = \pi_{i,h}. \tag{30}$$

It is easy to show that $\sum_{j \in \mathcal{S}_i} \hat{\pi}_{i,j} = \pi_{i,h} + \sum_{j \in \mathcal{S}_i} \pi_{i,j} = k_i$ and $\hat{\pi}_{i,j} \in [0,1]$ because $\hat{\pi}_{i,j} = \max(u, \pi_{i,j}) \in [0,1]$. Here, we used the fact that $u < 1$ since $k_i = \sum_{j \in \mathcal{S}_i} \hat{\pi}_{i,j} \geq \sum_{j \in \mathcal{S}_i} u \geq k_i u$. Therefore, the system of linear equations in (27) with $\hat{\pi}_{i,j}$ on the right-hand side must have a nonnegative solution due to our induction assumption for $n_i = |\mathcal{S}_i|$. Furthermore, without loss of generality, we assume that $y_h \geq y_j$ for all $j \in \mathcal{S}_i$ (otherwise a different h can be chosen). It implies that

$$\sum_{j \in \mathcal{S}_i} y_j \hat{\pi}_{i,j} = \sum_{j \in \mathcal{S}_i} y_j (\pi_{i,j} + [u - \pi_{i,j}]^+)$$

$$\overset{(a)}{\leq} \sum_{j\in\mathcal{S}_i} y_j\pi_{i,j} + \sum_{j\in\mathcal{S}_i} y_h[u-\pi_{i,j}]^+$$
$$\overset{(b)}{=} \sum_{j\in\mathcal{S}_i} y_j\pi_{i,j} + y_h \sum_{j\in\mathcal{S}_i} [u-\pi_{i,j}]^+$$
$$\overset{(c)}{=} \sum_{j\in\mathcal{S}_i} y_j\pi_{i,j} + y_h\pi_{i,h} \overset{(d)}{\leq} 0 \tag{31}$$

where (a) uses $y_h \geq y_j$, (b) uses that y_h is independent of j, (c) follows from (30), and the last step uses $\sum_j y_j\pi_{i,j} < 0$.

Applying the Farkas–Minkowski Theorem to the system of linear equations in (27) with $\hat{\pi}_{i,j}$ on the right-hand side, the existence of a nonnegative solution (due to our induction assumption for n_i) implies that $\sum_{j\in\mathcal{S}_i} y_j\mathbf{1}_{\{j\in\mathcal{A}_i\}} < 0$ for some $\hat{\mathcal{A}}_i \subseteq \mathcal{S}_i$. It means that

$$\sum_{j\in\mathcal{S}_i+\{h\}} y_j\mathbf{1}_{\{j\in\hat{\mathcal{A}}_i\}} = y_h\mathbf{1}_{\{h\in\hat{\mathcal{A}}_i\}} + \sum_{j\in\mathcal{S}_i} y_j\mathbf{1}_{\{j\in\hat{\mathcal{A}}_i\}} < 0. \tag{32}$$

The last step uses $\mathbf{1}_{\{h\in\hat{\mathcal{A}}_i\}} = 0$ since $h \notin \mathcal{S}_i$ and $\hat{\mathcal{A}}_i \subseteq \mathcal{S}_i$. This is exactly the desired inequality in (29). Thus, (28) has a nonnegative solution due to the Farkas–Minkowski Theorem. The induction statement holds for n_i+1. Finally, the solution indeed gives a probability distribution since $\sum_{\mathcal{A}_i\subseteq\mathcal{S}_i+\{h\}} \mathbb{P}(\mathcal{A}_i) = \sum_j \pi_{i,j}/k_i = 1$ due to (26). This completes the proof. □

B. Proof of Lemma 2

Proof: Let $\mathbf{Q}_{\max}$ be the maximum of waiting time $\{\mathbf{Q}_j, j \in \mathcal{A}_i\}$. We first show that $\mathbf{Q}_{\max}$ is upper-bounded by the following inequality for arbitrary $z \in \mathbb{R}$:

$$\mathbf{Q}_{\max} \leq z + [\mathbf{Q}_{\max} - z]^+ \leq z + \sum_{j\in\mathcal{A}_i} [\mathbf{Q}_j - z]^+ \tag{33}$$

where $[a]^+ = \max\{a, 0\}$ is a truncate function. Now, taking the expectation on both sides of (33), we have

$$\mathbb{E}[\mathbf{Q}_{\max}] \leq z + \mathbb{E}\left[\sum_{j\in\mathcal{A}_i} [\mathbf{Q}_j - z]^+\right]$$
$$= z + \mathbb{E}\left[\sum_{j\in\mathcal{A}_i} \frac{1}{2}(\mathbf{Q}_j - z + |\mathbf{Q}_j - z|)\right]$$
$$= z + \mathbb{E}_{\mathcal{A}_i}\left[\sum_{j\in\mathcal{A}_i} \frac{1}{2}(\mathbb{E}[\mathbf{Q}_j] - z + \mathbb{E}|\mathbf{Q}_j - z|)\right]$$
$$= z + \sum_{j\in\mathcal{A}_i} \frac{\pi_{i,j}}{2}(\mathbb{E}[\mathbf{Q}_j] - z + \mathbb{E}|\mathbf{Q}_j - z|) \tag{34}$$

where $\mathbb{E}_{\mathcal{A}_i}$ denotes the expectation over randomly selected k_i storage nodes in $\mathcal{A}_i \subseteq \mathcal{S}$ according to probabilities $\pi_{i,1}, \ldots, \pi_{i,m}$. From Cauchy–Schwarz inequality, we have

$$\mathbb{E}|\mathbf{Q}_j - z| \leq \sqrt{(\mathbb{E}[\mathbf{Z}_j] - z)^2 + \mathrm{Var}[\mathbf{Q}_j]}. \tag{35}$$

Combining (34) and (35), we obtain the desired result by taking a minimization over $z \in \mathbb{R}$.

Finally, it is easy to verify that the bound is tight for the same binary distribution constructed in [36], i.e., $\mathbf{Q}_j = z \pm \sqrt{(\mathbb{E}[\mathbf{Q}_j] - z)^2 + \mathrm{Var}[\mathbf{Q}_j]}$ with probabilities

$$P_+ = \frac{1}{2} + \frac{1}{2}\cdot\frac{\mathbb{E}[\mathbf{Q}_j] - z}{\sqrt{(\mathbb{E}[\mathbf{Q}_j] - z)^2 + \mathrm{Var}[\mathbf{Q}_j]}} \tag{36}$$

and $P_- = 1 - P_+$, which satisfy the mean and variance conditions. Therefore, the upper bound in (5) is tight for this binary distribution. □

C. Derivation of Problem JLCM

Proof: Plugging the results from Lemmas 2 and 3 into (12) and applying the same z to all $\bar{T}_i$ [which relax the problem and maintains inequality (5)], we obtain the desired objective function Problem JLCM. In the derivation, we used the fact that $\Lambda_j = \sum_i \lambda_i\pi_{i,j}\ \forall j$ from (3). Notice that the first summation is changed from $j \in \mathbb{S}_i$ in Lemma 2 to $j = \{1, \ldots, m\}$ because we should always assign $\pi_{i,j} = 0$ to storage node j that does not host any chunks of file i, i.e., for all $j \notin \mathcal{S}_i$. □

D. Proof of Lemma 6

Proof: We only need to show that $G = \left[X_j + \sqrt{X_j^2 + Y_j}\right]$ is convex in Λ_j as $F = \Lambda_j G/2$. To prove that $G = \left[X_j + \sqrt{X_j^2 + Y_j}\right]$ is convex in Λ_j, we have

$$\frac{\partial^2 G}{\partial\Lambda_j^2} = \frac{\partial^2 X_j}{\partial\Lambda_j^2} + \frac{X_j\frac{\partial^2 X_j}{\partial\Lambda_j^2} + Y_j\frac{\partial^2 Y_j}{\partial\Lambda_j^2}}{(X_j^2+Y_j^2)^{1/2}} + \frac{(X_j\frac{\partial Y_j}{\partial\Lambda_j} + Y_j\frac{\partial X_j}{\partial\Lambda_j})^2}{(X_j^2+Y_j^2)^{3/2}} \tag{37}$$

from where we can see that in order for $\frac{\partial^2 G}{\partial\Lambda_j^2}$ to be positive we only need $\frac{\partial^2 X_j}{\partial\Lambda_j^2}$ and $\frac{\partial^2 Y_j}{\partial\Lambda_j^2}$ to be positive. Then, we have

$$\frac{\partial^2 X_j}{\partial\Lambda_j^2} = \frac{\mu_j^2\Gamma_j^2}{(\mu_j - \Lambda_j)^3} > 0$$
$$\frac{\partial^2 Y_j}{\partial\Lambda_j^2} = \frac{2\mu_j^2\hat{\Gamma}_j^3}{3(\mu_j - \Lambda_j)^3} + \frac{\mu_j^4\Gamma_j^4(2\mu_j + 4\Lambda_j)}{(\mu_j - \Lambda_j)^4} > 0.$$

Now we can verify that $G = \left[X_j + \sqrt{X_j^2 + Y_j}\right]$ is convex in Λ_j. It further implies that $F = \Lambda_j G/2$ is also convex. This completes the proof. □

E. Proof of Theorem 2

Proof: To simplify notations, we first introduce two auxiliary functions

$$g = \sum_{j=1}^{m} \frac{\Lambda_j}{2}\left[X_j + \sqrt{X_j^2 + Y_j}\right] \tag{38}$$
$$h = \theta\sum_{i=1}^{r}\sum_{j=1}^{m}\left[V_j\mathbf{1}_{\left(\pi_{i,j}^{(t)}>0\right)} + \frac{V_j(\pi_{i,j} - \pi_{i,j}^{(t)})}{(\pi_{1,j}^{(t)} + 1/\beta)\log\beta}\right]. \tag{39}$$

Therefore Problem (23) is equivalent to $\min_\pi(g + h)$ over $\pi = (\pi_{i,j}^{(t)}\ \forall i, j)$. For any $\beta > 0$, due to the the concavity of logarithmic functions we have $\log(\beta y + 1) - \log(\beta x + 1) \leq \beta(y - x)/(\beta x + 1)$ for any nonnegative x, y. Choosing $x = \pi_{i,j}^{(t)}$ and $y = \pi_{i,j}^{(t+1)}$ and multiplying a constant $V_j/\log\beta$ on both sides of the inequality, we have

$$\frac{V_j\left(\pi_{i,j}^{(t+1)} - \pi_{i,j}^{(t)}\right)}{\left(\pi_{1,j}^{(t)} + \frac{1}{\beta}\right)\log\beta} \geq V_j\frac{\log\left(\beta\pi_{i,j}^{(t+1)} + 1\right)}{\log\beta} - V_j\frac{\log\left(\beta\pi_{i,j}^{(t)} + 1\right)}{\log\beta}. \tag{40}$$

Therefore, we construct a new auxiliary function

$$\hat{h} = \theta \sum_{i=1}^{r} \sum_{j=1}^{m} V_j \frac{\log(\beta \pi_{i,j} + 1)}{\log \beta}. \quad (41)$$

Since $\pi_{i,j}^{(t+1)}$ minimizes Problem (23), we have

$$g\left(\pi^{(t+1)}\right) + h\left(\pi^{(t+1)}\right) \leq g\left(\pi^{(t)}\right) + h\left(\pi^{(t)}\right). \quad (42)$$

Next, we consider a new objective function $[g + \hat{h}]$ and show that it generates a descent sequence, i.e.,

$$\begin{aligned}
&[g + \hat{h}](\pi^{(t+1)}) - [g + \hat{h}](\pi^{(t)}) \\
&\quad \leq h(\pi^{(t)}) - h(\pi^{(t+1)}) + \hat{h}(\pi^{(t+1)}) - \hat{h}(\pi^{(t)}) \\
&\quad = \sum_{i=1}^{r} \sum_{j=1}^{m} \frac{V_j(\pi_{i,j} - \pi_{i,j}^{(t)})}{(\pi_{1,j}^{(t)} + 1/\beta)\log\beta} + \hat{h}(\pi^{(t+1)}) - \hat{h}(\pi^{(t)}) \\
&\quad \leq 0 \quad (43)
\end{aligned}$$

where the first step uses (42) and the last step follows from (40). Therefore, Algorithm JLCM generates a descent sequence, $\pi_{i,j}^{(t)}$ for $t = 0, 1, \ldots$, for objective function $[g + \hat{h}]$. Notice that for any $\pi_{i,j} \in [0, 1]$, we have

$$\lim_{\beta \to \infty} \hat{h}(\pi) = \sum_{i=1}^{r} \sum_{j=1}^{m} V_j \mathbf{1}_{(\pi_{i,j} > 0)} \quad (44)$$

which is exactly the cost function in Problem JLCM. The converging point of the descent sequence is also a local optimal point of Problem JLCM as $\beta \to \infty$. □

REFERENCES

[1] E. Schurman and J. Brutlag, "The user and business impact of server delays, additional bytes and http chunking in web search," presented at the OReilly Velocity Web Perform. Oper. Conf., Jun. 2009.

[2] G. Liang and U. Kozat, "FAST CLOUD: Pushing the envelope on delay performance of cloud storage with coding," *IEEE/ACM Trans. Netw.*, vol. 22, no. 6, pp. 2012–2025, Nov. 2013.

[3] S. Chen *et al.*, "When queueing meets coding: Optimal-latency data retrieving scheme in storage clouds," in *Proc. IEEE INFOCOM*, Apr. 2014, pp. 1042–1050.

[4] G. Liang and U. C. Kozat, "TOFEC: Achieving optimal throughput-delay trade-off of cloud storage using erasure codes," in *Proc. IEEE INFOCOM*, Apr. 2014, pp. 826–834.

[5] V. Shah and G. Veciana, "Performance evaluation and asymptotics for content delivery networks," in *Proc. IEEE INFOCOM*, Apr. 2014, pp. 2607–2615.

[6] C. Angllano, R. Gaeta, and M. Grangetto, "Exploiting rateless codes in cloud storage systems," *IEEE Trans. Parallel Distrib. Syst.*, vol. 26, no. 5, pp. 1313–1322, May 2015.

[7] A. Kumar, R. Tandon, and T. C. Clancy, "On the latency of erasure-coded cloud storage systems," arXiv:1405.2833, May 2014.

[8] A. D. Luca and M. Bhide, *Storage Virtualization for Dummies, Hitachi Data Systems Edition*. Hoboken, NJ, USA: Wiley, 2009.

[9] Amazon S3, "Amazon Simple Storage Service," [Online]. Available: http://aws.amazon.com/s3/

[10] M. Sathiamoorthy *et al.*, "XORing elephants: Novel erasure codes for big data," in *Proc. 39th VLDB Endowment*, 2013, pp. 325–336.

[11] A. Fikes, "Storage architecture and challenges," Talk at the Google Faculty Summit, 2010 [Online]. Available: http://bit.ly/nUylRW

[12] A. G. Dimakis, K. Ramchandran, Y. Wu, and C. Suh, "A survey on network codes for distributed storage," arXiv:1004.4438, Apr. 2010.

[13] A. Fallahi and E. Hossain, "Distributed and energy-aware MAC for differentiated services wireless packet networks: A general queuing analytical framework," in *Proc. IEEE CS, CASS, ComSoc, IES, SPS*, 2007, pp. 381–394.

[14] F. Baccelli, A. Makowski, and A. Shwartz, "The fork-join queue and related systems with synchronization constraints: Stochastic ordering and computable bounds," *Adv. Appl. Probab.*, vol. 21, no. 3, pp. 629–660, 1989.

[15] A. S. Alfa, "Matrix-geometric solution of discrete time MAP/PH/1 priority queue," *Naval Res. Logistics*, vol. 45, pp. 23–50, 1998.

[16] J. H. Kim and J. K. Lee, "Performance of carrier sense multiple access with collision avoidance in wireless LANs," in *Proc. IEEE IPDS*, 1998, pp. 161–183.

[17] E. Ziouva and T. Antoankopoulos, "CSMA/CA performance under high traffic conditions: Throughput and delay analysis," *Comput. Commun.*, vol. 25, pp. 313–321, 2002.

[18] N. E. Taylor and Z. G. Ives, "Reliable storage and querying for collaborative data sharing systems," in *Proc. IEEE ICDE*, 2010, pp. 40–51.

[19] R. Rosemark and W. C. Lee, "Decentralizing query processing in sensor networks," in *Proc. 2nd Mobiquitous, Netw. Services*, 2005, pp. 270–280.

[20] A. D. G. Dimakis, V. Prabhakaran, and K. Ramchandran, "Distributed data storage in sensor networks using decentralized erasure codes," in *Conf. Rec. 38th Asilomar Conf. Signals, Syst. Comput.*, 2004, pp. 1387–1391.

[21] R. Rojas-Cessa, L. Cai, and T. Kijkanjanarat, "Scheduling memory access on a distributed cloud storage network," in *Proc. IEEE 21st Annu. WOCC*, 2012, pp. 71–76.

[22] M. K. Aguilera, R. Janakiraman, and L. Xu, "Using erasure codes efficiently for storage in a distributed system," in *Proc. Int. Conf. DSN*, 2005, pp. 336–345.

[23] Q. Lv, P. Cao, E. Cohen, K. Li, and S. Shenker, "Search and replication in unstructured peer-to-peer networks," in *Proc. 16th ICS*, 2002, pp. 84–95.

[24] H. Kameyam and Y. Sato, "Erasure codes with small overhead factor and their distributed storage applications," in *Proc. 41st Annu. CISS*, 2007, pp. 80–85.

[25] J. Li, "Adaptive erasure resilient coding in distributed storage," in *Proc. IEEE Int. Conf. Multimedia Expo*, 2006, pp. 561–564.

[26] X. Wang, Z. Xiao, J. Han, and C. Han, "Reliable multicast based on erasure resilient codes over Infiniband," in *Proc. Commun. Netw. China, 1st Int. Conf.*, 2006, pp. 1–6.

[27] S. Mochan and L. Xu, "Quantifying benefit and cost of erasure code based file systems," Technical Report, Dec. 2007 [Online]. Available: http://nisl.wayne.edu/Papers/Tech/cbefs.pdf

[28] H. Weatherspoon and J. D. Kubiatowicz, "Erasure coding vs. replication: A quantitative comparison," in *Proc. 1st IPTPS*, 2002, pp. 328–338.

[29] A. Abdelkefi and J. Yuming, "A structural analysis of network delay," in *Proc. 9th Annu. CNSR*, 2011, pp. 41–48.

[30] A. B. Downey, "The structural cause of file size distributions," in *Proc. 9th MASCOTS*, 2011, p. 361.

[31] F. Paganini, A. Tang, A. Ferragut, and L. L. H. Andrew, "Network stability under alpha fair bandwidth allocation with general file size distribution," *IEEE Trans. Autom. Control*, vol. 57, no. 3, pp. 579–591, Mar. 2012.

[32] B. Calder *et al.*, "Windows azure storage: A highly available cloud storage service with strong consistency," in *Proc. 23rd ACM SOSP*, 2011, pp. 143–157.

[33] L. Huang, S. Pawar, H. Zhang, and K. Ramchandran, "Codes can reduce queueing delay in data centers," in *Proc. IEEE ISIT*, 2012, pp. 2766–2770.

[34] M. Bramson, Y. Lu, and B. Prabhakar, "Randomized load balancing with general service time distributions," in *Proc. ACM SIGMETRICS*, 2010, pp. 275–286.

[35] Y. Lu, Q. Xie, G. Kliot, A. Geller, J. Larus, and A. Greenberg, "Joinidle-queue: A novel load balancing algorithm for dynamically scalable web services," *Perform. Eval. Archive 2011*, vol. 68, no. 11, pp. 1056–1071, 2011.

[36] D. Bertsimas and K. Natarajan, "Tight bounds on expected order statistics," *Probab. Eng. Inf. Sci.*, vol. 20, no. 4, pp. 667–686, 2006.

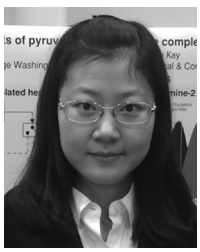
[37] L. Huang, S. Pawar, H. Zhang, and K. Ramchandran, "Codes can reduce queueing delay in data centers," arXiv: 1202.1359, 2012.

[38] S. Boyd and L. Vandenberghe, *Convex Optimization*. Cambridge, U.K.: Cambridge Univ. Press, 2005.

[39] L. T. H. An and P. D. Tao, "The DC (difference of convex functions) programming and DCA revisited with DC models of real world nonconvex optimization problems," *Ann. Oper. Res.*, vol. 133, no. 1–4, pp. 23–46, Jan. 2005.

[40] B. Warner, Z. Wilcox-O'Hearn, and R. Kinninmont, "Tahoe-LAFS docs," 2015 [Online]. Available: https://tahoe-lafs.org/trac/tahoe-lafs

[41] N. Shah, K. Lee, and K. Ramachandran, "The MDS queue: Analyzing latency performance of codes and redundant requests," arXiv: 1211. 5405, Nov. 2012.
[42] G. Joshi, Y. Liu, and E. Soljanin, "On the delay-storage trade-off in content download from coded distributed storage systems," *IEEE J. Sel. Areas Commun.*, vol. 32, no. 5, pp. 989–997, May 2014.
[43] MOSEK, "MOSEK: High performance software for large-scale LP, QP, SOCP, SDP and MIP," [Online]. Available: http://www.mosek.com/
[44] T. Angell, "The Farkas-Minkowski theorem," Lecture notes, 2002 [Online]. Available: http://www.math.udel.edu/~angell/Opt/farkas.pdf



Yu Xiang received the B.A.Sc. degree from Harbin Institute of Technology, Harbin, China, in 2010, and is currently pursuing the doctoral degree in electrical and computer engineering at George Washington University, Washington, DC, USA.

Her current research interests are in cloud resource optimization and distributed storage systems.

Tian Lan (S'03–M'10) received the B.A.Sc. degree from Tsinghua University, Beijing, China, in 2003, the M.A.Sc. degree from the University of Toronto, Toronto, ON, Canada, in 2005, and the Ph.D. degree from Princeton University, Princeton, NJ, USA, in 2010, all in electrical engineering.

He is currently an Assistant Professor of electrical and computer engineering at George Washington University, Washington, DC, USA. His research interests are in cloud resource optimization, distributed systems, and cyber security.

Dr. Lan received the 2008 IEEE Signal Processing Society Best Paper Award, the 2009 IEEE GLOBECOM Best Paper Award, and the 2012 INFOCOM Best Paper Award.

Vaneet Aggarwal (S'08–M'11–SM'15) received the B.Tech. degree from the Indian Institute of Technology, Kanpur, India, in 2005, and the M.A. and Ph.D. degrees from Princeton University, Princeton, NJ, USA, in 2007 and 2010, respectively, all in electrical engineering.

He is currently an Assistant Professor with Purdue University, West Lafayette, IN, USA. Prior to this, he was a Senior Member of Technical Staff Research with AT&T Labs—Research, Bedminster, NJ, USA, and an Adjunct Assistant Professor with Columbia University, New York, NY, USA. His research interests are in applications of information and coding theory to wireless systems and distributed storage systems.

Dr. Aggarwal was the recipient of Princeton University's Porter Ogden Jacobus Honorific Fellowship in 2009.

Yih-Farn (Robin) Chen received the B.S. degree in electrical engineering from National Taiwan University, Taiwan, in 1980, the M.S. degree in computer science from the University of Wisconsin–Madison, Madison, WI, USA, in 1983, and the Ph.D. degree in computer science from the University of California, Berkeley, CA, USA, in 1987.

He is a Director of Inventive Science, leading the Cloud Platform Software Research Department with AT&T Labs—Research, Bedminster, NJ, USA. His current research interests include cloud computing, software-defined storage, mobile computing, distributed systems, World Wide Web, and IPTV.

Dr. Chen is an ACM Distinguished Scientist and a Vice Chair of the International World Wide Web Conferences Steering Committee (IW3C2). He also serves on the Editorial Board of *IEEE Internet Computing*.