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

Yu Xiang[1], Tian Lan[1], Vaneet Aggarwal[2], and Yih-Farn R Chen[2]
[1] Department of ECE, George Washington University, DC 20052, USA
[2] AT&T Labs-Research, Bedminster, NJ 07921
{xy336699, tlan}@gwu.edu, {vaneet, chen}@research.att.com

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 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.

1. INTRODUCTION

1.1 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 [1]. The increased storage demands have led companies to launch cloud storage services like Amazon's S3 [2] 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.

Erasure coding has been widely studied for distributed storage systems [5, and references therein] and used by companies like Facebook [3] and Google [4] since it provides space-optimal data redundancy to protect against data loss.

IFIP WG 7.3 Performance 2014, October 7-9, Turin, Italy.

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 homogeneous files [33, 39, 41, 43], much less is known about the latency performance of multiple 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 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: (i) the scheduling policy that decides what request in the buffer to process when a node becomes available, (ii) the placement of file chunks over distributed storage nodes, and (iii) erasure coding parameters that decides how many chunks are created. A joint optimization over these three dimensions is very challenging because the latency performance 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 files and propose an efficient solution to the joint optimization of both latency and storage cost.

1.2 Related Work

Quantifying the exact service delay in an erasure-coded storage is an open problem. Prior works focusing on asymptotic queuing delay behaviours [35, 37] are not applicable because redundancy factor in practical data centers typi-

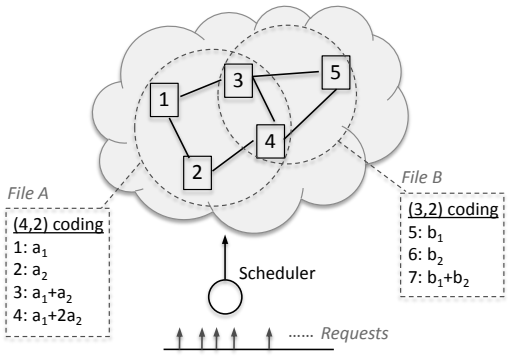

Figure 1: An erasure-coded storage of 2 files, which partitioned into 2 blocks and encoded using $(4, 2)$ and $(3, 2)$ MDS codes, respectively. Resulting file chunks are spread over 7 storage nodes. Any file request must be processed by 2 distinct nodes that have the desired chunks. Nodes $3, 4$ are shared and can process request for both files.

cally remain small due to storage cost concerns. Due to the lack of analytical delay 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., [14, 16, 19, 24, 22], which demonstrate latency improvement due to erasure coding in different system implementations, for example, in [11] the authors use the LT erasure codes to adjust the system to meet user requirements such as availability, integrity and confidentiality. Related design can also be found in data access scheduling [6, 8, 12, 13], access collision avoidance [9, 10], and encoding/decoding time optimization [25, 26]. Restricting to the special case of a single file, service delay bounds of erasure-coded storage have been recently studied in [33, 39, 41, 43].

Queueing-theoretic analysis. For homogeneous files and under an assumption of exponential service time distribution, the authors in [33, 39] 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 queueing-theoretic analysis for MDS codes with $k = 2$. Later, the approach is extended in [43] to general (n, k) erasure codes, yet for a single file. 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 [7] which has been extensively studied in the literature. Recently, the authors in [41] proposed a (n, k) fork-join queue 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. Using 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 homogeneous files and exponentially-distributed service time. However, the approach cannot be applied to a 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. Further, under a folk-join queue, each file request must be served by all n nodes or a set of pre-specified nodes. It falls short to address dynamic load-balancing of heterogeneous files.

1.3 Our Contributions

This paper aims to propose a systematic framework that *(i) 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 *(ii) 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, (i) 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 services time distributions. (ii) Since the latency for probabilistic scheduling for all probabilities over $\binom{n}{k}$ subsets is hard to evaluate, we show that the probabilistic scheduling is equivalent to accessing each of the n storage node 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. (iii) The policy that selects each storage node with certain probability generates memoryless requests at each of the node and thus the delay at each storage node can be characterized by the latency of M/G/1 queue. (iv) 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 [38]. Not only does our result supersede previous latency analysis [33, 39, 41, 43] 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 homogeneous-file case.

The main application of our latency analysis is a joint optimization of latency and storage cost for 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 parametrized by $\beta > 0$ can be efficiently computed using a standard projected gradient method and is shown to converge to the orig-

inal problem as $\beta \to \infty$.

To validate our theoretical analysis and joint latency-plus-cost optimization, we provide a prototype of the proposed algorithm in *Tahoe* [42], 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 heterogenerous files and different request arrival rates on the testbed. Our measurement of real service time distribution falsifies the exponential assumption in [33, 39, 43]. Our analysis outperforms the upper bound in [41] even in the 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.

2. 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 [12, 13, 15], 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 which 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*.

[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 [14, 16, 18].

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 [27, 26] and file-size distribution [28, 29]. 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.

Consider the 2-file storage example in Section 1, 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 initial 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 server 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 an 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 (i) buffers all requests in a queue of infinite capacity; (ii) assigns at most 1 chunk request from a batch to each appropriate node, and (iii) 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 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, 39] that blocks all but the first t batches does not apply to heterogeneous files because a Markov-chain representation of the resulting queue is required to have each state encapsulating

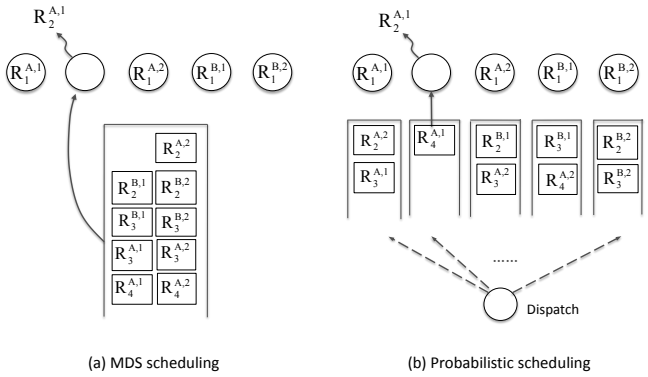

Figure 2: Functioning of (a) an optimal scheduling policy and (b) a probabilistic scheduling policy.

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 which 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 [41] also falls short because each file request must be forked to n_i servers, inevitably causing conflict at shared servers.

3. 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.

3.1 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) *An Probabilistic scheduling policy (i) dispatches each batch of chunk requests to appropriate nodes with predetermined probabilities; (ii) 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.*

Proof: Any probabilistic scheduling with feasible probabilities $\{\mathbb{P}(\mathcal{A}_i) : \forall i, \mathcal{A}_i\}$ can be viewed as an equivalent non-optimal scheduling, where chunk requests are buffered in a common queue and randomly assigned to appropriate servers. It is easy to see that it satisfies Properties (i) and (ii) of optimal scheduling. Since the optimal scheduling assigns chunk requests in a way to minimize average latency, probabilistic scheduling provides an upper bound.

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 $\sum_i (n_i\text{-choose-}k_i)$ 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 $\sum_i n_i$ 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

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

where $\mathbf{1}_{\{j \in \mathcal{A}_i\}}$ is an indicator function which 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*

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

The proof of Theorem 1 relying on Farkas-Minkowski Theorem [46] is detailed in [36]. 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 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 form 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 $\lambda_i \pi_{i,j}$. The resulting queueing system under probabilistic scheduling is stable if all local queues are stable.

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

COROLLARY 1. *The queuing system governed 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:*

$$\sum_{\mathcal{A}_i \subseteq \mathcal{S}_i} \mathbf{1}_{\{j \in \mathcal{A}_i\}} \cdot \mathbb{P}(\mathcal{A}_i) = \pi_{i,j}, \ \forall j \in \mathcal{S}_i \tag{3}$$

and also:

$$\Lambda_j = \sum_i \lambda_i \pi_{i,j} < \mu_j, \ \forall j. \tag{4}$$

Proof: The queuing system governed by a probabilistic scheduling policy is stable if all the local queues are stable. If $\{\pi_{i,j} \in [0,1], \forall i,j\}$ satisfies (3), due to Theorem 1, there exists a feasible scheduling policy with appropriate probabilities $\{\mathbb{P}(\mathcal{A}_i)$ such that $\pi_{i,j} = \sum_{\mathcal{A}_i : \mathcal{A}_i \subseteq \mathcal{S}_i} \mathbb{P}(\mathcal{A}_i) \cdot \mathbf{1}_{\{j \in \mathcal{A}_i\}}$ are the conditional probabilities of selecting node j for a file i request. Therefore, each local queue seems a superposition of r Poisson arrival processes, each with rate $\lambda_i \pi_{i,j}$, which is also a Poisson Process with rate $\Lambda_j \sum_i \lambda_i \pi_{i,j}$. The process is stable under traffic $\Lambda_j = \sum_i \lambda_i \pi_{i,j} < \mu_j, \ \forall j$.

3.2 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.,

$$\bar{T}_i = \mathbb{E}\left[\mathbb{E}_{\mathcal{A}_i}\left(\max_{j \in \mathcal{A}_i}\{\mathbf{Q}_j\}\right)\right], \tag{5}$$

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 [38], 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*

$$\begin{aligned}\bar{T}_i \quad \leq \min_{z \in \mathbb{R}} \Bigg\{ & z + \sum_{j \in \mathcal{S}_i} \frac{\pi_{i,j}}{2} \left(\mathbb{E}[\mathbf{Q}_j] - z\right) \\ & + \sum_{j \in \mathcal{S}_i} \frac{\pi_{i,j}}{2} \left[\sqrt{(\mathbb{E}[\mathbf{Q}_j] - z)^2 + \mathrm{Var}[\mathbf{Q}_j]}\right] \Bigg\}. \end{aligned} \tag{6}$$

The bound is tight in the sense that there exists a distribution of $\mathbf{Q}_j$ such that (6) 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 $\mathbb{E}[\mathbf{X}^2] - \mathbb{E}[\mathbf{X}]^2 = \sigma_j^2$, second moment $\mathbb{E}[\mathbf{X}^2] = \Gamma_j^2$, and third moment $\mathbb{E}[\mathbf{X}^3] = \hat{\Gamma}_j^3$. These statistics can be readily inferred from existing work on network delay [27, 26] and file-size distribution [28, 29].

LEMMA 3. *Using Pollaczek-Khinchin transform [39], expected delay and variance for total queueing and network delay are given by*

$$\mathbb{E}[\mathbf{Q}_j] = \frac{1}{\mu_j} + \frac{\Lambda_j \Gamma_j^2}{2(1-\rho_j)}, \tag{7}$$

$$\mathrm{Var}[\mathbf{Q}_j] = \sigma_j^2 + \frac{\Lambda_j \hat{\Gamma}_j^3}{3(1-\rho_j)} + \frac{\Lambda_j^2 \Gamma_j^4}{4(1-\rho_j)^2}, \tag{8}$$

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

Combining Lemma 2 and Lemma 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}.

4. 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.

4.1 Formulating the Joint Optimization

We showed that a probabilistic scheduling policy can be optimization over three sets of control variables: erasure coding 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.,

$$\begin{aligned} \min \quad & \sum_{i=1}^{r} \frac{\lambda_i}{\hat{\lambda}} \bar{T}_i + \theta \sum_{i=1}^{r} \sum_{j \in \mathcal{S}} V_j \\ \text{s.t.} \quad & (1),\ (2),\ (4),\ (6),\ (7),\ (8). \\ \text{var.} \quad & n_i,\ \pi_{i,j},\ \mathcal{S}_i \in \mathcal{M},\ \forall i,j. \end{aligned} \tag{9}$$

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 (9) 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 3. 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 (9) the results in Section 3 and obtain a Joint Latency-Cost Minimization (JLCM) with respect to probabilistic scheduling[3]:

Problem JLCM:

$$\min \quad z + \sum_{j=1}^{m} \frac{\Lambda_j}{2\hat{\lambda}} \left[X_j + \sqrt{X_j^2 + Y_j} \right] + \theta \sum_{i=1}^{r} \sum_{j \in \mathcal{S}_i} V_j \quad (10)$$

$$s.t. \quad X_j = \frac{1}{\mu_j} + \frac{\Lambda_j \Gamma_j^2}{2(1-\rho_j)} - z, \ \forall j \quad (11)$$

$$Y_j = \sigma_j^2 + \frac{\Lambda_j \hat{\Gamma}_j^3}{3(1-\rho_j)} + \frac{\Lambda_j^2 \Gamma_j^4}{4(1-\rho_j)^2}, \ \forall j \quad (12)$$

$$\rho_j = \Lambda_j / \mu_j < 1; \ \Lambda_j = \sum_{i=1}^{r} \pi_{i,j} \lambda_i \ \forall j \quad (13)$$

$$\sum_{j=1}^{m} \pi_{i,j} = k_i; \ \pi_{i,j} \in [0,1]; \ \pi_{i,j} = 0 \ \forall j \notin \mathcal{S}_i \quad (14)$$

$$|\mathcal{S}_i| = n_i \text{ and } \mathcal{S}_i \subseteq \mathcal{M}, \ \forall i \quad (15)$$

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

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 (15), while changing chunk placement $\mathcal{S}_i$ affects the feasibility of probabilities $\pi_{i,j}$ due to (14). 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

4.2 Constructing convex approximations

In the 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 4. *The optimal chunk placement of Problem JLCM must satisfy $\mathcal{S}_i = \{j : \pi_{i,j} > 0\}$ $\forall i$, which implies*

$$\sum_{j \in \mathcal{S}_i} V_j = \sum_{j=1}^{m} V_j \mathbf{1}_{(\pi_{i,j} > 0)}, \quad n_i = \sum_{j=1}^{m} V_j \mathbf{1}_{(\pi_{i,j} > 0)} \quad (16)$$

[3]The optimization is relaxed by applying the same axillary variable z to all $\bar{T}_i$, which still satisfies the inequality (6).

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 (14), with respect to the following objective function:

$$z + \sum_{j=1}^{m} \frac{\Lambda_j}{2\hat{\lambda}} \left[X_j + \sqrt{X_j^2 + Y_j} \right] + \theta \sum_{i=1}^{r} \sum_{j=1}^{m} V_j \mathbf{1}_{(\pi_{i,j} > 0)}. \quad (17)$$

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 (17) with in a small neighbourhood of the reference point. For all i, j, we have

$$V_j \mathbf{1}_{(\pi_{i,j} > 0)} \approx \left[V_j \mathbf{1}_{\left(\pi_{i,j}^{(t)} > 0\right)} + \frac{V_j (\pi_{i,j} - \pi_{i,j}^{(t)})}{(\pi_{1,j}^{(t)} + 1/\beta) \log \beta} \right], \quad (18)$$

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 (10) is convex in $\pi_{i,j}$ when all other variables are fixed.

LEMMA 5. *The following function, in which X_j and Y_j are functions of Λ_j defined by (11) and (12), is convex in Λ_j:*

$$F(\Lambda_j) = \frac{\Lambda_j}{2\hat{\lambda}} \left[X_j + \sqrt{X_j^2 + Y_j} \right]. \quad (19)$$

4.3 Algorithm JLCM and convergence analysis

Leveraging the linear local approximation in (18) 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 5 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 [40].

The proposed algorithm is shown in Figure 4.1. 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. More precisely, for $t = 1, 2, \ldots$ we

Algorithm JLCM :

Choose sufficiently large $\beta > 0$
Initialize $t = 0$ and feasible $(\pi_{i,j}^{(0)}\ \forall i,j)$
Compute current objective value $B^{(0)}$
while $B^{(0)} - B^{(1)} > \epsilon$
  Approximate cost function using (18) and $(\pi_{i,j}^{(t)}\ \forall i,j)$
  Call *projected_gradient*() to solve optimization (20)
    $(\pi_{i,j}^{(t+1)}\ \forall i,j) = \arg\min$ (20)
  $z = \arg\min$ (20)
  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 (16)
Output: $(n_i, \mathcal{S}_i, \pi_{i,j}^{(t)})\ \forall i,j$

Figure 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 $\nabla(18)$ with respect to $\pi_{i,j}^{(s)}$
  $\pi_{i,j}^{(s+1)} = \pi_{i,j}^{(s)} + \delta_s \cdot \nabla(18)$
  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: $(\pi_{i,j}^{(s)},\ \forall i,j)$

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

solve

$$\begin{aligned} \min \quad & \theta \sum_{i=1}^{r}\sum_{j=1}^{m}\left[V_j \mathbf{1}_{\left(\pi_{i,j}^{(t)}>0\right)} + \frac{V_j(\pi_{i,j} - \pi_{i,j}^{(t)})}{(\pi_{1,j}^{(t)} + 1/\beta)\log\beta}\right] \\ & + z + \sum_{j=1}^{m} \frac{\Lambda_j}{2\hat{\lambda}}\left[X_j + \sqrt{X_j^2 + Y_j}\right] \qquad (20) \\ s.t. \quad & \text{Constraints (11), (12), (13)} \\ & \sum_{j=1}^{m} \pi_{i,j} = k_i \text{ and } \pi_{i,j} \in [0,1] \\ var. \quad & \pi_{i,j}\ \forall i,j. \end{aligned}$$

Due to Lemma 5, 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 Figure 4.1. 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 JLCM using a standard Euclidean projection. It is shown that such a projected gradient descent method solves the optimal solution of Problem (20). 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:

$$\hat{C} = \sum_{i=1}^{r}\sum_{j=1}^{m} V_j \frac{\log(\beta\pi_{i,j} + 1)}{\log\beta}. \qquad (21)$$

The key idea in our proof is that the linear approximation of storage cost function in (18) can be seen as a sub-gradient 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.,

$$\lim_{\beta\to\infty} V_j \frac{\log(\beta\pi_{i,j} + 1)}{\log\beta} = V_j \mathbf{1}_{(\pi_{i,j}>0)}. \qquad (22)$$

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 $\beta \to \infty$. Further, it is shown that $\hat{h}$ is a concave function. Thus, minimizing $z + \sum_j F(\Lambda_j) + \theta\hat{h}$ can be viewed as optimizing the difference between 2 convex objectives, namely $z + \sum_j F(\Lambda_j)$ and $-\theta\hat{h}$, which can be also solved via a Difference-of-Convex Programming (DCP). In this context, our linear approximation of cost function in (18) can be viewed as an approximated supper-gradient in DCP. Please refer to [44] for a comprehensive study of regularization techniques in DCP to speed up the convergence of Algorithm JLCM.

5. IMPLEMENTATION AND EVALUATION

5.1 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* [42], which is an open-source, distributed filesystem based on the *zfec* erasure coding library. It provides three special instances of a generic *node*: (a) *Tahoe Introducer*: it keeps track of a collection of storage servers and clients and introduces them to each other. (b) *Tahoe Storage Server*: it exposes attached storage to external clients and stores erasure-coded shares. (c) *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 a (n, k) encoding scheme) and then distributed to (ideally) n distinct storage servers. The set of blocks on each storage

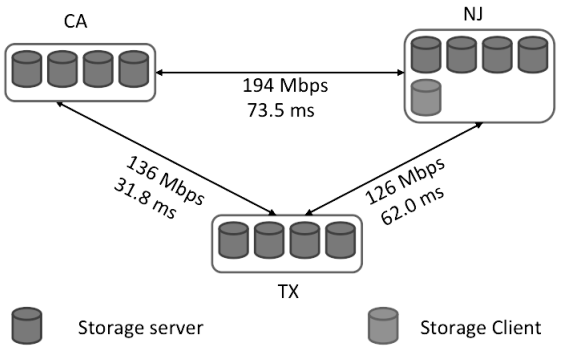

Figure 5: Our Tahoe testbed with average ping (RTT) and bandwidth measurements among three data centers in New Jersey, Texas, and California

server constitute a chunk. Thus, the file equivalently consists of k chunks which 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 responds 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 (RTT). 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.

We deployed 12 Tahoe storage servers 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. The deployment is shown in Figure 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 time measured by ping does not necessarily correlate to the bandwidth number. Further, the current implementation of Tahoe does not use up the maximum available bandwidth, even with our multi-port revision.

5.2 Experiments and Evaluation

[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.

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 actual service time distribution on our testbed. We upload a 50MB file using a $(7, 4)$ erasure code and measure the chunk service time. Figure 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 seconds with a standard deviation of 4.3 seconds, 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, 39] is falsified by empirical data. The observation is also evident because a distribution never has positive probability for very small service time. Further, 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 with the outer bound in [41]. Even though our upper bound holds for multiple heterogeneous files, and includes connection delay, we restrict our comparison to the case for homogeneous files without any connection delay for a fair comparison (since the upper bound in [41] only works for a single file). We plot the latency upper bound that we give in this paper and the upper bound in [Theorem 3, [41]] in Figure 7. In our probabilistic scheduling, access requests are dispatched uniformly to all storage nodes. We find that our bound significantly outperforms the upper bound in [41] for a wide range of $1/\lambda < 32$, which represents medium to high traffic regime. In particular, our bound works fine in high traffic regime with $1/\lambda < 18$, whereas the upper bound in [41] goes to infinity and thus fail to offer any useful information. Under low traffic, the two bounds get very close to each other with a less than 4% gap.

Validate Algorithm JLCM and joint optimization. We implemented Algorithm JLCM and used MOSEK [45], a commercial optimization solver, to realize the projected gradient routine. For 12 distributed storage nodes in our testbed, Figure 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 time-scale, 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 $\epsilon = 0.01$. 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 with three oblivious schemes, each of which minimize latency-plus-cost over only a subset of the 3 dimensions: load-balancing (LB),

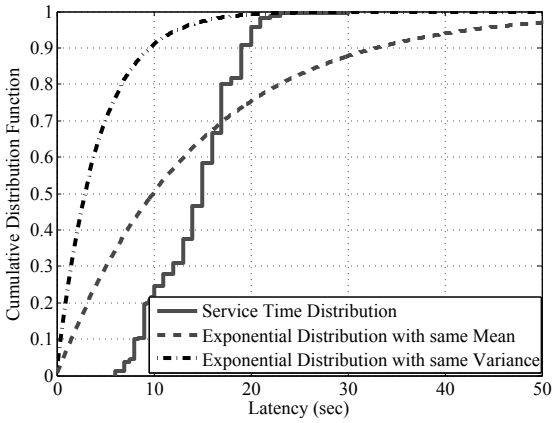

Figure 6: Comparison of actual service time distribution and an exponential distribution with the same mean. It verifies that actual service time does not follow an exponential distribution, falsifying the assumption in previous work [34, 40].

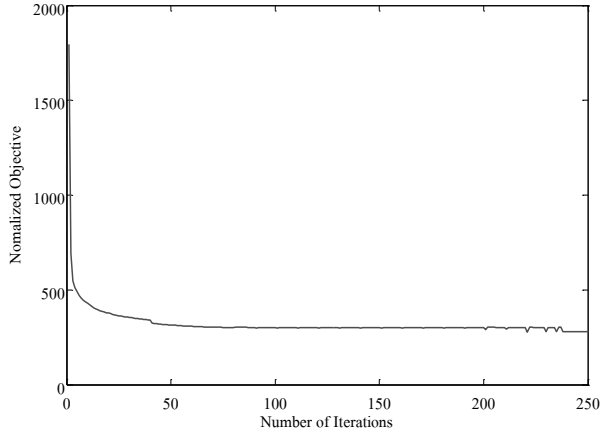

Figure 8: Convergence of Algorithm JLCM for different problem size with $r = 1000$ files for our 12-node testbed. The algorithm efficiently computes a solution in less than 250 iterations.

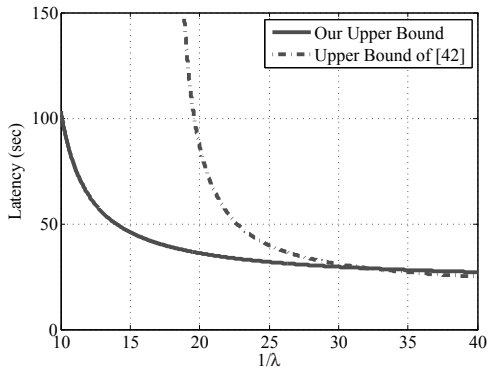

Figure 7: Comparison of our latency upper bound with previous work [42]. Our bound significantly improves previous result under medium to high traffic and comes very close to that of [42] under low traffic (with less than 4% gap).

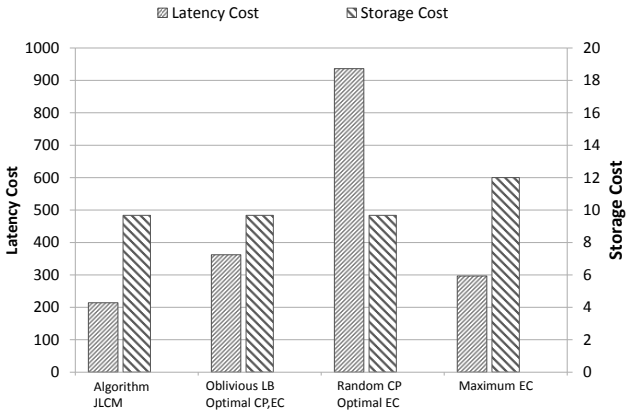

Figure 9: Comparison of Algorithm JLCM with some oblivious approaches. Algorithm JLCM minimizes latency-plus-cost over 3 dimensions: load-balancing (LB), chunk placement (CP), and erasure code (EC), while any optimizations over a subset of the dimensions is non-optimal.

chunk placement (CP), and erasure code (EC). We run Algorithm JLCM for $r = 1000$ files of size $150MB$ on our testbed, with $V_j = \$1$ for every $25MB$ storage and a tradeoff factor of $\theta = 2$ sec/dollar. The result is shown in Figure. 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*, which adopts the best outcome of 100 random runs. Large latency increment 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 $n = m$ and selects all nodes for chunk placement. 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 3 dimensions.

Evaluate the performance of our solution. First, we choose $r = 1000$ files of size $150MB$ and the same storage cost and tradeoff factor as in the previous experiment. Request arrival rates are set to $\lambda_i = 1.25/(10000sec)$, for $i = 1, 4, 7, \cdots 997$, $\lambda_i = 1.25/(10000sec)$, for $i = 2, 5, 8, \cdots 998$ and $\lambda_i = 1.25/(12000sec)$, for $i = 3, 6, 9, \cdots 999, 1000$ respectively, which leads to an aggregate file request arrival rate of $\lambda = 0.118$ /sec. 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 (11,6), (10,7), (10,6) and (9,4) for each quarter of the 1000 files respectively, as well as as-

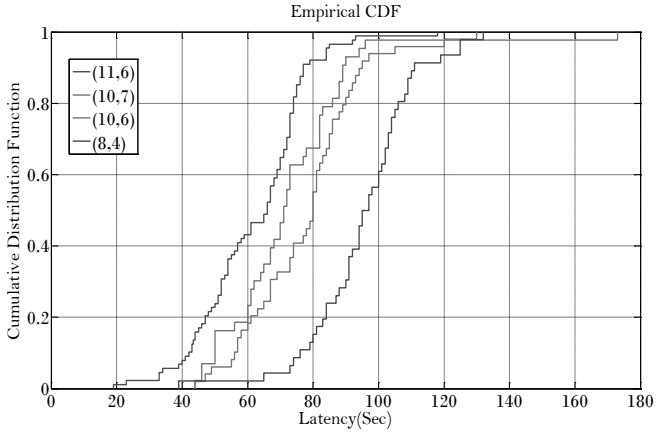

Figure 10: Actual service latency distribution of an optimal solution from Algorithm JLCM for 1000 files of size $150MB$ using erasure code (11,6), (10,7), (10,6) and (8,4) for each quarter with aggregate request arrival rates are set to $\lambda_i = 0.118$ /sec

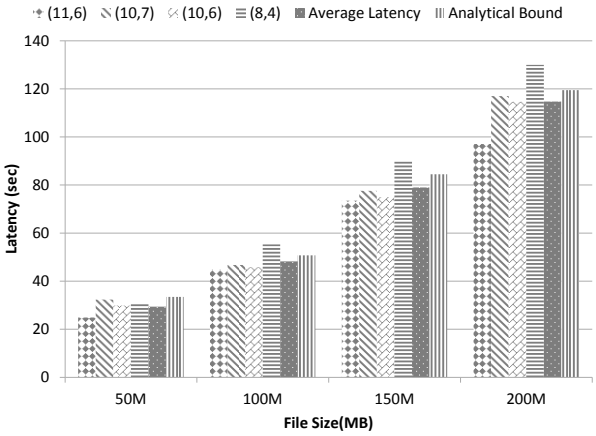

Figure 11: Evaluation of different chunk sizes. Latency increases super-linearly as file size grows due to queuing delay. Our analytical latency bound taking both network and queuing delay into account tightly follows actual service latency.

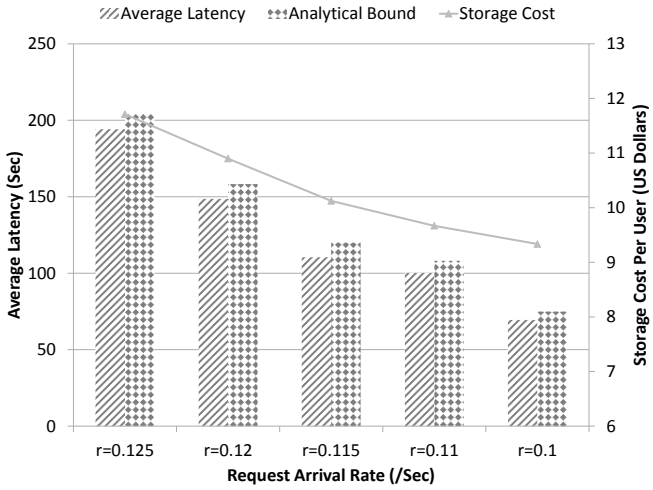

Figure 12: Evaluation of different request arrival rates. As arrival rates increase, latency increases and becomes more dominating in the latency-plus-cost objective than storage cost. The optimal solution from Algorithm JLCM allows higher storage cost, resulting in a nearly-linear growth of average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

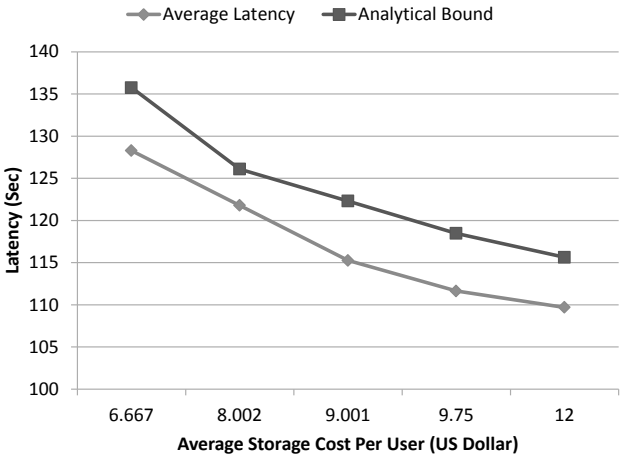

Figure 13: Visualization of latency and cost tradeoff for varying $\theta = 0.5$ second/dollar to $\theta = 200$ second/dollar. As θ increases, higher weight is placed on the storage cost component of the latency-plus-cost objective, leading to less file chunks and higher latency.

sociated 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 Figure 10. We note that 95% of download requests for files with erasure code (10,7) complete within 100 seconds, while the same percentage of requests for files using (11,6) erasure code complete within 32 seconds due to higher level of redundancy. In this experiment erasure code (10,6) 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.

To demonstrate the effectiveness of our joint optimization, we vary file size in the experiment from 50MB to 200MB and plot the average download latency of the 1000 individual files, out of which each quarter is using a distinct erasure code (11,6), (10,7), (10,6) and (9,4), and our analytical latency upper bound in Figure 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). Further, smaller files always have lower latency because it is less costly to achieve higher redundancy for these files. We also observe that our analytical latency bound tightly follows actual average service latency.

Next, we varied aggregate file request arrival rate from $\lambda_i = 0.125$ /sec to $\lambda_i = 0.1$ /sec (with individual arrival rates also varies accordingly), while keeping tradeoff factor at $\theta = 2$ sec/dollar and file size at $200MB$. Actual service delay and our analytical bound for each scenario is shown by a bar plot in Figure 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. Figure 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.

Visualize latency and cost tradeoff. Finally, we demonstrate the tradeoff between latency and storage cost in our joint optimization framework. Varying the tradeoff factor in Algorithm JLCM from $\theta = 0.5$ sec/dollar to $\theta = 200$ sec/dollar for fixed file size of $200MB$ and aggregate arrival rates $\lambda_i = 0.125$ /sec, 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 Figure 13. When $\theta = 0.5$, the optimal solution from Algorithm JLCM chooses three sets of erasure codes (12,6), (12,7), and (12,4) erasure codes, which is the maximum erasure code length in our framework and leads to highest storage cost (i.e., 12 dollars for each user), yet lowest latency (i.e., 110 sec). 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 seconds. Further, the theoretical tradeoff calculated by our analytical bound and Algorithm JLCM is very close to the actual measurement on our testbed. To the best our our knowledge, this is the first work proposing a joint optimization algorithm to exploit such tradeoff in an erasure-coded, distributed storage system.

6. 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 non-convex 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 latency and dynamic data management have been ignored in this paper and open up avenues for future work.

7. REFERENCES

[1] A.D. Luca and M. Bhide, "Storage virtualization for dummies, Hitachi Data Systems Edition," *John and Wiley Publishing*, 2009.

[2] Amazon S3, "Amazon Simple Storage Service," *available online at http://aws.amazon.com/s3/.*

[3] Sathiamoorthy, Maheswaran, et al. "Xoring elephants: Novel erasure codes for big data." Proceedings of the 39th international conference on Very Large Data Bases. VLDB Endowment, 2013.

[4] Fikes, Andrew. "Storage architecture and challenges." Talk at the Google Faculty Summit,*available online at http://bit.ly/nUylRW*, 2010.

[5] A. G. Dimakis, K. Ramchandran, Y. Wu, C. Suh, "A Survey on Network Codes for Distributed Storage," arXiv:1004.4438, Apr. 2010

[6] A. Fallahi and E. Hossain, "Distributed and energy-Aware MAC for differentiated services wireless packet networks: a general queuing analytical framework," *IEEE CS, CASS, ComSoc, IES, SPS*, 2007.

[7] F.Baccelli, A.Makowski, and A.Shwartz, "The fork-join queue and related systems with synchronization constraints: stochastic ordering and computable bounds, *Advances in Applied Probability*, pp. 629660, 1989.

[8] A.S. Alfa, "Matrix-geometric solution of discrete time MAP/PH/1 priority queue," *Naval research logistics*, vol. 45, 00. 23-50, 1998.

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

[10] E. Ziouva and T. Antoankopoulos, "CSMA/CA Performance under high traffic conditions: throughput and delay analysis," *Computer Comm*, vol. 25, pp. 313-321, 2002.

[11] C. Angllano, R. Gaeta and M. Grangetto, "Exploiting Rateless Codes in Cloud Storage Systems," *IEEE Transactions on Parallel and Distributed Systems*, Pre-print 2014.

[12] N.E. Taylor and Z.G. Ives, "Reliable storage and querying for collaborative data sharing systems," *IEEE ICED Conference*, 2010.

[13] R. Rosemark and W.C. Lee, "Decentralizing query processing in sensor networks," *Proceedings of the second MobiQuitous: networking and services*, 2005

[14] Dimakis, Alexandros D G , "Distributed data storage in sensor networks using decentralized erasure codes," *Signals, Systems and Computers, 2004. Conference Record of the Thirty-Eighth Asilomar.*, 2004.

[15] R. Rojas-Cessa, L. Cai and T. Kijkanjanarat, "Scheduling memory access on a distributed cloud storage network," *IEEE 21st annual WOCC*, 2012.

[16] M.K. Aguilera, R. Janakiraman, L. Xu, "Using Erasure Codes Efficiently for Storage in a Distributed System," *Proceedings of the 2005 International Conference on DSN*, pp. 336-345, 2005.

[17] S. Chen, K.R. Joshi and M.A. Hiltunem, "Link Gradients: Predicting the Impact of Network Latency on Multi-Tier Applications," *Proc. IEEE INFOCOM*, 2009.

[18] Q. Lv, P. Cao, E. Cohen, K. Li and S. Shenker, "Search and replication in unstructured peer-to-peer networks," *Proceedings of the 16th ICS*, 2002.

[19] H. Kameyam and Y. Sato, "Erasure Codes with Small Overhead Factor and Their Distributed Storage Applications," *CISS '07. 41st Annual Conference*, 2007.

[20] H.Y. Lin, and W.G. Tzeng, "A Secure Decentralized Erasure Code for Distributed Networked Storage," *Parallel and Distributed Systems, IEEE Transactions*, 2010.

[21] W. Luo, Y. Wang and Z. Shen, "On the impact of erasure coding parameters to the reliability of distributed brick storage systems," *Cyber-Enabled Distributed Computing and Knowledge Discovery, International Conference*, 2009.

[22] J. Li, "Adaptive Erasure Resilient Coding in Distributed Storage," *Multimedia and Expo, 2006 IEEE International Conference*, 2006.

[23] K. V. Rashmi, N. Shah, and V. Kumar, "Enabling node repair in any erasure code for distributed storage," *Proceedings of IEEE ISIT*, 2011.

[24] X. Wang, Z. Xiao, J. Han and C. Han, "Reliable Multicast Based on Erasure Resilient Codes over InfiniBand," *Communications and Networking in China, First International Conference*, 2006.

[25] S. Mochan and L. Xu, "Quantifying Benefit and Cost of Erasure Code based File Systems." *Technical report available at* $http://nisl.wayne.edu/Papers/Tech/cbefs.pdf$, 2010.

[26] H. Weatherspoon and J. D. Kubiatowicz, "Erasure Coding vs. Replication: A Quantitative Comparison." *In Proceedings of the First IPTPS,*2002

[27] A. Abdelkefi and J. Yuming, "A Structural Analysis of Network Delay," *Ninth Annual CNSR*, 2011.

[28] A.B. Downey, "The structural cause of file size distributions," *Proceedings of Ninth International Symposium on MASCOTS*, 2011.

[29] F. Paganini, A. Tang, A. Ferragut and L.L.H. Andrew, "Network Stability Under Alpha Fair Bandwidth Allocation With General File Size Distribution," *IEEE Transactions on Automatic Control*, 2012.

[30] P. Corbett, B. English, A. Goel, T. Grcanac, S. Kleiman, J. Leong and S. Sankar, "Row-diagonal parity for double disk failure correction," *In Proceedings of the 3rd USENIX FAST'*, pp. 1-14, 2004.

[31] B. Calder, J. Wang, A. Ogus, N. Nilakantan, A. Skjolsvold, S. McKelvie, Y. Xu, S. Srivastav, J. Wu, H. Simitci, et al., " Windows azure storage: A highly available cloud storage service with strong consistency," *In Proceedings of the Twenty-Third ACM SOSP*, pages 143–157, 2011.

[32] O. Khan, R. Burns, J. Plank, W. Pierce, and C. Huang, "Rethinking erasure codes for cloud file systems: Minimizing I/O for recovery and degraded reads," *In Proceedings of FAST*, 2012.

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

[34] G. Ananthanarayanan, S. Agarwal, S. Kandula, A Greenberg, and I. Stoica, "Scarlett: Coping with skewed content popularity in MapReduce," *Proceedings of ACM EuroSys*, 2011.

[35] M. Bramson, Y. Lu, and B. Prabhakar, "Randomized load balancing with general service time distributions," *Proceedings of ACM Sigmetrics*, 2010.

[36] Y. Xiang, T. Lan, V. Aggarwal, Y. Chen, "Joint Latency and Cost Optimization for Erasure-coded Data Center Storage," *Full paper available at* $http://arxiv.org/pdf/1404.4975.pdf$, 2014.

[37] 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," *29th IFIPPERFORMANCE*, 2010.

[38] D. Bertsimas and K. Natarajan, "Tight bounds on Expected Order Statistics," *Probability in the Engineering and Informational Sciences*, 2006.

[39] L. Huang, S. Pawar, H. Zhang and K. Ramchandran, "Codes Can Reduce Queueing Delay in Data Centers," *Journals CORR*, vol. 1202.1359, 2012.

[40] S. Boyd and L. Vandenberghe, "Convex Optimization," *Cambridge University Press*, 2005.

[41] G. Joshi, Y. Liu, and E. Soljanin, "On the Delay-Storage Trade-off in Content Download from Coded Distributed Storage Systems," *arXiv:1305.3945v1*, May 2013.

[42] B. Warner, Z. Wilcox-O'Hearn and R. Kinninmont, "Tahoe-LAFS docs," *available online at https://tahoe-lafs.org/trac/tahoe-lafs.*

[43] N. Shah, K. Lee, and K. Ramachandran, "The MDS queue: analyzing latency performance of codes and redundant requests," *arXiv:1211.5405*, Nov. 2012.

[44] L.T. Hoai An and P.D. Tao,"The DC (Difference of Convex Functions) Programming and DCA Revisited with DC Models of Real World Non-convex Optimization Problems," *Annals of Operations Research*, vol. 133, Issue 1-4, pp. 23-46, Jan 2005.

[45] MOSEK, "MOSEK: High performance software for large-scale LP, QP, SOCP, SDP and MIP," available online at *http://www.mosek.com/.*

[46] T. Angell, "The Farkas-Minkowski Theorem". Lecture nodes available online at *www.math.udel.edu/∼angell/Opt/farkas.pdf*, 2002.