Taming Tail Latency for Erasure-coded, Distributed Storage Systems

Vaneet Aggarwal, Abubakr O. Al-Abbasi, Jingxian Fan, and Tian Lan

Abstract—Distributed storage systems are known to be susceptible to long tails in response time. In modern online storage systems such as Bing, Facebook, and Amazon, the long tails of the service latency are of particular concern. with 99.9th percentile response times being orders of magnitude worse than the mean. As erasure codes emerge as a popular technique to achieve high data reliability in distributed storage while attaining space efficiency, taming tail latency still remains an open problem due to the lack of mathematical models for analyzing such systems. To this end, we propose a framework for quantifying and optimizing tail latency in erasure-coded storage systems. In particular, we derive upper bounds on tail latency in closed-form for arbitrary service time distribution and heterogeneous files. Based on the model, we formulate an optimization problem to jointly minimize weighted latency tail probability of all files over the placement of files on the servers, and the choice of servers to access the requested files. The non-convex problem is solved using an efficient, alternating optimization algorithm. Numerical results show significant reduction of tail latency for erasure-coded storage systems with realistic workload.

Index Terms—Tail latency, Erasure coding, Distributed Storage Systems, Bi-partite matching, Alternating optimization, Laplace Stieltjes transform.

I. Introduction

Due to emerging applications such as big data analytics and cloud computing, distributed storage systems today often store multiple petabytes of data [2–4]. As a result, these systems are transitioning from full data replication to the use of erasure code for encoding and spreading data chunks across multiple machines and racks, in order to achieve more efficient use of storage space while maintaining high reliability despite system failures. It is shown that using erasure codes can reduce the cost of storage by more than 50% [3] due to smaller storage space and datacenter footprint.

A key tradeoff for using erasure codes is performance. Distributed storage systems that employ erasure codes are known to be susceptible to long latency tails. Under full data replication, if a file is replicated n times, it can be recovered from any of the n replica copies. However, for an erasure-coded storage system using an (n, k) code, a file is encoded into n equal-size data chunks, allowing reconstruction from any subset of $k < n$ chunks. Thus, reconstructing the file requires fetching k distinct chunks from different servers, which leads to significant increase of tail latency, since service latency in such systems is determined by the *hottest* storage

V. Aggarwal. A. O. Al-Abbasi, and J. Fan are with Purdue University, West Lafayette IN 47907, email: {vaneet,aalabbas,fan137}@purdue.edu. T. Lan is with the George Washington University, Washington, DC 20052, email: tlan@gwu.edu. This work was presented in part at the IEEE International Conference on Computer Communications (Infocom) 2017 [1].

nodes with highest congestion and slowest speed, which effectively become performance bottlenecks. It has been shown that in modern Web applications such as Bing, Facebook, and Amazon's retail platform, the long tail of latency is of particular concern, with 99.9th percentile response times that are orders of magnitude worse than the mean [5, 6]. Despite mechanisms such as load-balancing and resource management, evaluations on large scale storage systems indicate that there is a high degree of randomness in delay performance [7]. The overall response time in erasure coded data-storage systems is dominated by the long tail distribution of the required parallel operations [8].

To the best of our knowledge, quantifying the impact of erasure coding on tail latency is an open problem for distributed storage systems. Although recent research progress has been made on providing bounds of mean service latency [9–13], much less is known on tail latency (i.e., xth-percentile latency for arbitrary $x \in [0, 1]$) in erasure-coded storage systems. Mean Service latency for replication-based systems for identical servers with independent exponential service-times has been characterized for homogeneous files in [14]. However, the problem for erasure-coded based systems is still an open problem. To provide an upper bound on mean service latency of homogeneous files, *Fork-join queue analysis* in [10, 15–19] provides upper bounds for mean service latency by forking each file request to all storage nodes. In a separate line of work, *Queuing-theoretic analysis* in [9, 11] proposes a *block-t-scheduling* policy that only allows the first t requests at the head of the buffer to move forward. However, both approaches fall short of quantifying tail latency due to a *state explosion* problem, because states of the corresponding queuing model must encapsulate not only a snapshot of the current system including chunk placement and queued requests but also past history of how chunk requests have been processed by individual nodes. Later, mean latency bounds for arbitrary service time distribution and heterogeneous files are provided in [12, 13] using order statistic analysis and a probabilistic request scheduling policy. The authors in [20] used probabilistic scheduling with uniform probabilities and exponential service times to show improved latency performance of erasure coding as compared to replication in the limit of large number of servers for replication-based systems. While reducing mean latency is found to have a positive impact on pushing down the latency envelop (e.g., reducing the 90th, and 99th percentiles) [7], quantifying and optimizing tail latency for erasure-coded storage is still an open problem.

In this paper, we propose an analytical framework to quantify tail latency in distributed storage systems that employ

erasure codes to store files. This problem is challenging because (i) tail latency is significantly skewed by performance of the slowest storage nodes; (ii) a joint chunk scheduling problem needs to be solved on the fly to decide n-choose-k chunks/servers serving each file request; and (iii) the problem is further complicated by the dependency and interference of chunk access times of different files on shared storage servers. Toward this end, we make use of probabilistic scheduling proposed in [12, 13, 21–24]. Upon the arrival of each file request, we randomly dispatch a batch of k chunk requests to k-out-of-n storage nodes selected with some 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. This probabilistic scheduling policy allows us to analyze the (marginal) queuing delay distribution of each storage node and then combine the results (through Laplace Stieltjes Transform and order statistic bounds) to obtain an upper bound on tail latency in closed-form for general service time distributions. The tightest bound is obtained via an optimization over all probabilistic schedulers and all Markov bounds on tail probability.

The proposed framework provides a mathematical crystallization of tail latency, illuminating key system design tradeoffs in erasure-coded storage. Prior evaluation of practical systems show that the latency spread is significant even when data object sizes are in the order of megabytes [7]. To tame tail latency in erasure coded storage, we propose an optimization problem to jointly minimize the sum probability that service latency of each file exceeds a given threshold. This optimization is carried out over three dimensions: the joint placement of all files, all probabilistic schedulers, and the auxiliary variables in the tail latency bounds. We note that the probabilistic scheduler helps decrease the differentiated tail latency of the files as compared to accessing the lowest-queue servers which is important for overall tail latency of files. Since data chunk transfer time in practical systems follows a shifted exponential distribution [13, 18, 25], we show that under this assumption, the tail latency optimization can be formulated in closed-form as a non-convex minimization. To solve the problem, we prove that it is convex in two of the optimization variables and propose an alternating optimization algorithm, while the optimization with respect to file placement can be solved optimally using bipartite matching.Extensive simulations shows significant reduction of tail latency for erasure-coded storage systems using the proposed optimization over five different baseline strategies.

The main contributions of this paper are summarized as follows:

- We propose an analytical framework to quantify tail latency for arbitrary erasure-coded storage systems and service time distributions.
- When chunk transfer time follows shifted-exponential distribution, we formulate a weighted latency tail probability optimization that simultaneous minimizes tail latency of all files by optimizing the system over three dimensions: chunk placement, auxiliary variables, and the scheduling policy.
- We develop an alternating optimization algorithm which is

shown to converge to a local optima for the tail latency optimization. Two of the subproblems are convex, while bipartite matching is used to solve the third subproblem. Significant tail latency reduction up to a few orders of magnitude is validated through numerical results.

The rest of the paper is organized as follows. Section II gives the system model for the problem. Section III finds an upper bound on tail latency through probabilistic scheduling and Laplace Stieltjes transform of the waiting time from each server. Section IV formulates and solves the tail latency optimization. Section V presents our numerical results and Section VI concludes the paper.

II. System Model

We consider a data center consisting of m heterogeneous servers, denoted by $M = 1, 2, ..., m$, also called storage nodes. To distributively store a set of r files, indexed by $i = 1, 2, ...r$, we partition each file i into k_i fixed-size chunks 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, represented by a set S_i of storage nodes, satisfying $S_i \subseteq M$ and $n_i = |S_i|$. 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. Thus, upon the arrival of each file request, k_i distinct chunks are selected by a scheduler and retrieved to reconstruct the desired file. Figure 1 illustrates a distributed storage system with 7 nodes. Three files are stored in the system using $(6, 4)$, $(5, 3)$, and $(3, 2)$ erasure codes, respectively. File requests arriving at the system are jointly scheduled to access k_i-out-of-n_i distinct chunks. Prior work analyzing erasure-coded storage systems mainly focus on mean latency, including two approaches using queuing-theoretic analysis in [9, 11] and fork-join queue analysis in [10, 15–19].

However, both approaches fall short of quantifying tail latency, because states of the corresponding queuing model must encapsulate not only a snapshot of the current system including chunk placement and queued requests, but also past history of how chunk requests have been processed by individual nodes. This leads to a *state explosion* problem as practical storage systems usually handle a large number of files and nodes [13]. To the best of our knowledge, quantifying tail latency for erasure-coded storage system is still an open problem because of challenges in joint request scheduling (i.e., selecting n-choose-k chunks for each request on the fly with the goal of minimizing tail latency) as well as the dependency of straggling fragments on hot storage nodes. Consider the erasure-coded storage system storing 3 files, as shown in Figure 1. It is easy to see that a simple scheduling policy that accesses available chunks with equal probability lead to high tail latency, which is determined by hot storage nodes (i.e., nodes 1 and 5 in this case) with slowest performance. Yet, a policy that load-balances the number of requests processed by each server does not necessarily optimize tail latency of all files, which employ different erasure codes resulting in different impact on service latency. Assuming that chunk
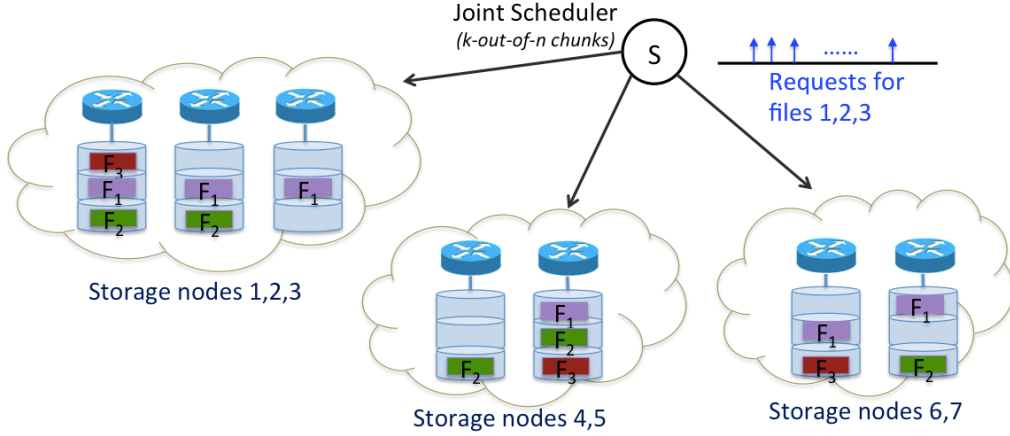
Fig. 1. An illustration of a distributed storage system equipped with 7 nodes and storing 3 files using different erasure codes.

transfer time from all storage nodes have the same distribution, file 1 using $(6,4)$ code could still have much higher tail latency than file 3 that uses $(3,2)$ code, since its service time of each file request is determined by the slowest of the 4 selected chunks (rather than 2 selected chunks).

In this paper, we use the Probabilistic Scheduling from [12, 13], which is a probabilistic scheduling policy: 1) dispatches each batch of chunk requests (corresponding to the same file request) to appropriate a set of nodes (denoted by set A_i of servers for file i) with predetermined probabilities ($P(A_i)$ for set A_i and file i); 2) each node buffers requests in a local queue and processes in order. The authors of [12, 13] have shown that a probabilistic scheduling policy with feasible probabilities $\{P(A_i) : \forall i, A_i\}$ exists if and only if there exists conditional probabilities $\pi_{i,j} \in [0,1], \forall i, j$ satisfying

$$\sum_{j=1}^{m} \pi_{i,j} = k_i \quad \forall i \quad \text{and} \quad \pi_{i,j} = 0 \text{ if } j \notin S_i.$$

Consider the example shown in Figure 1. Under probabilistic scheduling, upon the arrival of a file 1 request, we randomly select $k_1 = 4$ nodes (from $\{1, 2, 3, 5, 6, 7\}$) with available file chunks with respect to known probabilities $\{\pi_{1,j}, \forall j\}$ and dispatch a chunk request to each selected storage node. Then, each storage node manages its local queue independently and continues processing requests in order. The file request is completed if all its chunk requests are processed by individual nodes. While this probabilistic scheduling is used to provide an upper bound on mean service time in [12, 13], we extend the policy and provide an analytical model for tail latency, enabling a novel tail latency optimization.

We will 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 X_j of node j with arbitrary distributions, whose statistics can be obtained inferred from existing work on network delay [26, 27] 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. Even though the choice of codes for different files can be different, we assume that the chunk size is the same for all files. All requests are buffered in a common queue of infinite capacity.

TABLE I
MAIN NOTATIONS USED IN THIS PAPER

Symbol	Meaning
r	Number of files in system by $i = 1, 2, ..., r$
m	Number of storage nodes
(n_i, k_i)	Erasure code parameters for file i
λ_i	Arrival rate of file i
π_{ij}	Probability of retrieving chunk of file i from node j
L_i	Latency of retrieving file i
x	Parameter indexing latency tail probability
Q_j	Sojourn Time of node j
X_j	Chunk Service Time of node j
$M_j(t)$	Moment Generating Function for the service time of node j
μ_j	Mean service time of node j
Λ_j	Arrival rate on node j
ρ_j	Request intensity at node j
S_i	Set of storage nodes having chunks from file i
A_i	Set of nodes used to provide chunks from file i
(α_j, β_j)	Parameters of Shifted Exponential distributed service time at node j
ω_i	weight of file i

III. BOUNDS ON TAIL LATENCY

We first quantify tail latency for erasure-coded storage systems with arbitrary service time distribution (i.e., arbitrary known distribution of X_j). Let $\mathbf{Q}_j$ be the (random) time the chunk request spends in node j (sojourn time). Under probabilistic scheduling, the service time (denoted by L_i) of a file-i request is determined by the maximum chunk service time at a randomly selected set A_i of storage nodes.

Under probabilistic scheduling, the arrival of chunk requests at node j form a Poisson Process with rate $\Lambda_j = \sum_i \lambda_i \pi_{ij}$.

Let $M_j(t) = \mathbb{E}[e^{tX_j}]$ be the moment generating function of service time of processing a single chunk at server j. Then, the Laplace Stieltjes Transform of $\mathbf{Q}_j$ is given, using Pollaczek-Khinchine formula, as

$$\mathbb{E}[e^{-sQ_j}] = \frac{(1-\rho_j)sM_j(-s)}{s - \Lambda_j(1 - M_j(-s))}, \qquad (1)$$

where $\rho_j = \Lambda_j \mathbb{E}[X_j]$ is the request intensity at node j, and $M_j(t) = \mathbb{E}[e^{tX_j}]$ is the moment generating function of X_j [30]. Further, let the latency of the file i be denoted as L_i using probabilistic scheduling. The *latency tail probability* of file i is defined as the probability that L_i is greater than or equal to x, for a given x. For given weight w_i for file i, this paper wishes to minimize $\sum_i w_i \Pr(L_i \geq x)$. Since finding $\Pr(L_i \geq x)$ in closed form is hard for general service time distribution, we further use an upper bound on this and use that instead of $\Pr(L_i \geq x)$ in the objective.

The following theorem gives an upper bound on the latency tail probability of a file.

Theorem 1. *The latency tail probability for file i, $\Pr(L_i \geq x)$ using probabilistic scheduling is bounded by*

$$\Pr(L_i \geq x) \leq \sum_j \frac{\pi_{ij}}{e^{t_j x}} \frac{(1-\rho_j)t_j M_j(t_j)}{t_j - \Lambda_j(M_j(t_j) - 1)}, \qquad (2)$$

for any $t_j > 0$, $\rho_j = \Lambda_j \mathbb{E}[X_j]$, satisfying $M_j(t_j) < \infty$ and $\Lambda_j(M_j(t_j) - 1) < t_j$.

Proof. We consider an upper bound on latency tail probability using probabilistic scheduling as follows.

$$\Pr(L_i \geq x) \overset{(a)}{=} \Pr_{A_i, Q_j}\left(\max_{j \in A_i} Q_j \geq x\right) \qquad (3)$$

$$= \Pr_{A_i, Q_j}\left(Q_j \geq x \text{ for some } j \in A_i\right) \qquad (4)$$

$$= \mathbb{E}_{A_i, Q_j}\left[\max_{j \in A_i} 1_{(Q_j \geq x)}\right] \qquad (5)$$

$$\leq \mathbb{E}_{A_i, Q_j} \sum_{j \in A_i} [1_{(Q_j \geq x)}] \qquad (6)$$

$$= \mathbb{E}_{A_i} \sum_{j \in A_i} [\Pr(Q_j \geq x)] \qquad (7)$$

$$= \sum_j \pi_{ij}[\Pr(Q_j \geq x)], \qquad (8)$$

where (a) follows since for probabilistic scheduling, the time to retrieve the file is the maximum of the time of retrieving all the chunks from A_i.

Using Markov Lemma, we have $\Pr(Q_j \geq x) \leq \frac{\mathbb{E}[e^{t_j Q_j}]}{e^{t_j x}}$. In order to obtain $\mathbb{E}[e^{t_j Q_j}]$, we use Pollaczek-Khinchine formula for Laplace Stieltjes Transform of Q_j in (1) and use $s = -t_j$. However, the expression is finite only when $\Lambda_j(M_j(t_j) - 1) < t_j$. This proves the result as in the statement of the Theorem. $\square$

In some cases, the moment generating function may not exist, which means that the condition $\Lambda_j(M_j(t_j) - 1) < t_j$ may not be satisfied for any $t_j > 0$. In such cases, we will use the Laplace Stieltjes Transform directly to give another upper bound in the next theorem.

Theorem 2. *The latency tail probability for file i, $\Pr(L_i \geq x)$ is bounded by*

$$\Pr(L_i \geq x) \leq \sum_j \frac{\pi_{ij}(1 - \mathbb{E}[e^{-s_j Q_j}])}{1 - e^{-s_j x}}, \qquad (9)$$

for any $s_j > 0$, where $\rho_j = \Lambda_j \mathbb{E}[X_j]$, $\mathbb{E}[e^{-sQ_j}] = \frac{(1-\rho_j)sL_j(s)}{s - \Lambda_j(1 - L_j(s))}$, and $L_j(s) = E[e^{-sX_j}]$.

Proof. This result is a variant of Theorem 1, where Markov Lemma is used using Laplace Stieljes Transform of the Queue Waiting Time rather than the moment generating function. $\square$

We next consider the case when the service time distribution is a shifted exponential distribution. This choice is motivated by the Tahoe experiments [13] and Amazon S3 experiments [18]. Let the service time distribution from server j has probability density function $f_{X_j}(x)$, given as

$$f_{X_j}(x) = \begin{cases} \alpha_j e^{-\alpha_j(x - \beta_j)}, & \text{for } x \geq \beta_j \\ 0, & \text{for } x < \beta_j \end{cases}. \qquad (10)$$

Exponential distribution is a special case with $\beta_j = 0$. The Moment Generating Function is given as

$$M_j(t) = \frac{\alpha_j}{\alpha_j - t} e^{\beta_j t} \quad \text{for } t < \alpha_j. \qquad (11)$$

Using these expressions, we have the following result.

Corollary 1. *When the service time distributions of servers are given by shifted exponential distribution, the latency tail probability for file i, $\Pr(L_i \geq x)$, is bounded by*

$$\Pr(L_i \geq x) \leq \sum_j \frac{\pi_{ij}}{e^{t_j x}} \frac{(1-\rho_j)t_j M_j(t_j)}{t_j - \Lambda_j(M_j(t_j) - 1)}, \qquad (12)$$

for any $t_j > 0$, $\rho_j = \frac{\Lambda_j}{\alpha_j} + \Lambda_j \beta_j$, $\rho_j < 1$, and $t_j(t_j - \alpha_j + \Lambda_j) + \Lambda_j \alpha_j(e^{\beta_j t_j} - 1) < 0$.

Proof. We note that the condition $\Lambda_j(M_j(t_j) - 1) < t_j$ reduces to $t_j(t_j - \alpha_j + \Lambda_j) + \Lambda_j \alpha_j(e^{\beta_j t_j} - 1) < 0$. Since $t_j \geq \alpha_j$ will not satisfy $t_j(t_j - \alpha_j + \Lambda_j) + \Lambda_j \alpha_j(e^{\beta_j t_j} - 1) < 0$, the conditions in the statement of the Corollary implies $t_j < \alpha_j$ where the above moment generating function expression is used. $\square$

Since exponential distribution is a special case of the shifted exponential distribution, we have the following corollary.

Corollary 2. *When the service time distributions of servers are given by exponential distribution, the latency tail probability for file i, $\Pr(L_i \geq x)$, is bounded by*

$$\Pr(L_i \geq x) \leq \sum_j \frac{\pi_{ij}}{e^{t_j x}} \frac{(1-\rho_j)t_j M_j(t_j)}{t_j - \Lambda_j(M_j(t_j) - 1)}, \qquad (13)$$

for any $t_j > 0$, $\rho_j = \frac{\Lambda_j}{\alpha_j}$, $\rho_j < 1$, $t_j < \alpha_j(1 - \rho_j)$

IV. Optimizing Weighted Latency Tail Probability

Now we formulate a joint latency tail probability optimization for multiple, heterogeneous files. Since the latency tail probability is given by $\Pr(L_i \geq x)$ for $x > \max_j \beta_j$, we

consider an optimization that minimizes *weighted latency tail probability* of all files, defined by

$$\sum_i \omega_i \Pr(L_i \geq x), \tag{14}$$

where $\omega_i = \frac{\lambda_i}{\sum_i \lambda_i}$ is a positive weight assigned to file i so that the files with larger arrival rates are weighted higher, and latency tail probability of file-i service time is $\Pr(L_i \geq x)$. We consider the proposed bound on the latency tail probability to have the objective function as

$$\sum_i \lambda_i \left(\sum_j \frac{\pi_{ij}}{e^{t_j x}} \frac{(1-\rho_j)t_j M_j(t_j)}{t_j - \Lambda_j(M_j(t_j)-1)} \right). \tag{15}$$

Let $\boldsymbol{\pi} = \{\pi_{i,j} \forall i, j\}$, $\mathbf{t} = \{t_j \forall j\}$, and $\boldsymbol{\mathcal{S}} = \{\mathcal{S}_i \forall i\}$. We consider the following Weighted Latency Tail Probability (WLTP) optimization problem over the scheduling probabilities $\boldsymbol{\pi}$, the placement of files $\boldsymbol{\mathcal{S}}$, and auxiliary parameters $\mathbf{t}$, i.e.,

$$\min \quad \sum_j \Lambda_j e^{-t_j x} \frac{(1-\rho_j)t_j M_j(t_j)}{t_j - \Lambda_j(M_j(t_j)-1)} \tag{16}$$

$$s.t. \quad \Lambda_j = \sum_i \lambda_i \pi_{ij} \tag{17}$$

$$M_j(t) = \frac{\alpha_j}{\alpha_j - t} e^{\beta_j t} \tag{18}$$

$$\rho_j = \frac{\Lambda_j}{\alpha_j} + \Lambda_j \beta_j \tag{19}$$

$$\sum_j \pi_{i,j} = k_i \tag{20}$$

$$\pi_{i,j} = 0, j \notin \mathcal{S}_i \tag{21}$$

$$\pi_{i,j} \in [0,1] \tag{22}$$

$$|\mathcal{S}_i| = n_i, \ |\mathcal{A}_i| = k_i \ \forall i \tag{23}$$

$$t_j \geq 0 \tag{24}$$

$$t_j(t_j - \alpha_j + \Lambda_j) + \Lambda_j \alpha_j(e^{\beta_j t_j} - 1) < 0 \tag{25}$$

$$var. \quad \boldsymbol{\pi}, \mathbf{t}, \boldsymbol{\mathcal{S}} \tag{26}$$

Here, Constraint (17) gives the aggregate arrival rate Λ_j for each node under give scheduling probabilities $\pi_{i,j}$ and arrival rates λ_i, Constraint (18) defines moment generating function with respect to parameter t_j, Constraint (19) defines the traffic intensity of the servers, Constraints (20-22) guarantee that the scheduling probabilities are feasible, and finally, the moment generating function exists due to the technical constraint in (25). If (25) is satisfied, $\rho_j < 1$ holds too thus ensuring the stability of the storage system (i.e., queue length does not blow up to infinity under given arrival rates and scheduling probabilities). We note that $t_j > 0$ can be equivalently converted to $t_j \geq 0$ (and thus done in (24)) since $t_j = 0$ do not satisfy $t_j(t_j - \alpha_j + \Lambda_j) + \Lambda_j \alpha_j(e^{\beta_j t_j} - 1) < 0$ and has already been accounted for. We note that the the optimization over $\boldsymbol{\pi}$ helps decrease the weighted tail latency probability and gives significant flexibility over choosing the lowest-queue servers for accessing the files. The placement of the files $\boldsymbol{\mathcal{S}}$ helps separate the highly accessed files on different servers thus reducing the objective. Finally, the optimization over the auxiliary variables $\mathbf{t}$ gives a tighter bound on the weighted latency tail probability.

Remark 1. *The proposed WLTP optimization is non-convex, since Constraint (25) is non-convex in ($\boldsymbol{\pi}$, $\mathbf{t}$). Further, the content placement $\boldsymbol{\mathcal{S}}$ has integer constraints.*

To develop an algorithmic solution, we prove that the problem is convex individually with respect to the optimization variables $\mathbf{t}$ and $\boldsymbol{\pi}$, when the other variables are fixed. This result allows us to propose an alternating optimization algorithm for the problem. The next result shows the the problem is convex in $\mathbf{t} = (t_1, t_2, \cdots, t_m)$.

Theorem 3. *The objective function, $\sum_j \frac{\Lambda_j}{e^{t_j x}} \frac{(1-\rho_j)t_j M_j(t_j)}{t_j - \Lambda_j(M_j(t_j)-1)}$ is convex in $\mathbf{t} = (t_1, t_2, \cdots, t_m)$ in the region where the constraints in (17)-(25) are satisfied.*

Proof. We note that inside the summation, the term only depends on a single value of t_j. Thus, it is enough to show that $\frac{t_j e^{-t_j x} M_j(t_j)}{t_j - \Lambda_j(M_j(t_j)-1)}$ is convex with respect to t_j. Since there is only a single index j here, we ignore this subscipt for the rest of this proof.

We denote

$$F(t) = \frac{te^{-tx}M(t)}{t - \Lambda(M(t)-1)} \tag{27}$$

$$= \frac{\alpha t e^{(\beta-x)t}}{-t^2 + (\alpha - \Lambda)t + \Lambda\alpha - \Lambda\alpha e^{\beta t}} \tag{28}$$

$$= \frac{\alpha t e^{(\beta-x)t}}{-t^2 + (\alpha - \Lambda)t - \Lambda\alpha(e^{\beta t} - 1)} \tag{29}$$

$$= \frac{\alpha t e^{(\beta-x)t}}{-t^2 + (\alpha - \Lambda)t - \Lambda\alpha \sum_{u=1}^\infty \frac{(\beta t)^u}{u!}} \tag{30}$$

$$= \frac{\alpha e^{(\beta-x)t}}{-t + (\alpha - \Lambda) - \Lambda\alpha \sum_{u=1}^\infty \frac{(\beta)^u t^{u-1}}{u!}} \tag{31}$$

Thus, $F(t)$ can be written as product of $f(t) = \alpha e^{(\beta-x)t}$ and $g(t) = \frac{1}{h(t)}$, where $h(t) = -t + (\alpha - \Lambda) - \Lambda\alpha \sum_{u=1}^\infty \frac{(\beta)^u t^{u-1}}{u!}$. Since the constraints in (17)-(25) are satisfied, $h(t) > 0$. Further, all positive deriavatives of $h(t)$ are non-positive. Let $w(t) = -h'(t)$. Then, $w(t) \geq 0$, and $w'(t) \geq 0$.

Further, we have

$$g(t) = \frac{1}{h(t)}$$

$$g'(t) = \frac{w(t)}{h^2(t)}$$

$$g''(t) = \frac{h(t)w'(t) + 2w^2(t)}{h^3(t)}. \tag{32}$$

Using these, $F''(t)$ is given as

$$
\begin{aligned}
& F''(t) \\
=\ & f''(t)g(t) + f(t)g''(t) + 2f'(t)g'(t) \\
=\ & \alpha e^{(\beta-x)t}\left(((\beta-x)^2 g(t) + g''(t) + 2(\beta-x)g'(t))\right) \\
=\ & \frac{\alpha e^{(\beta-x)t}}{h^3(t)}\left((\beta-x)^2 h^2(t) + h(t)w'(t) + 2w^2(t)\right. \\
& \left. + 2(\beta-x)w(t)h(t)\right) \\
=\ & \frac{\alpha e^{(\beta-x)t}}{h^3(t)}\left(2\left(\frac{(\beta-x)h(t)}{2} + w(t)\right)^2 + h(t)w'(t)\right. \\
& \left. + \frac{(\beta-x)^2 h^2(t)}{4}\right) \\
\geq\ & 0, \hspace{5cm} (33)
\end{aligned}
$$

where the last step follows since $h(t) \geq 0$, and $w'(t) \geq 0$. Thus, the objective function is convex in $\mathbf{t} = (t_1, t_2, \cdots, t_m)$. $\square$

The next result shows that the proposed problem is convex in $\boldsymbol{\pi} = (\pi_{ij}\forall i = 1, \cdots, r, j = 1, \cdots, m)$.

Theorem 4. *The objective function, $\sum_j \frac{\Lambda_j}{e^{t_j x}}\frac{(1-\rho_j)t_j M_j(t_j)}{t_j - \Lambda_j(M_j(t_j)-1)}$ is convex in $\boldsymbol{\pi} = (\pi_{ij}\forall(i,j))$.*

Proof. Since the sum of convex functions is convex, it is enough to show that $\Lambda_j\frac{(1-\rho_j)}{t_j - \Lambda_j(M_j(t_j)-1)}$ is convex. Since Λ_j is a linear function of $\boldsymbol{\pi}$, it is enough to prove that $\Lambda_j\frac{(1-\rho_j)}{t_j - \Lambda_j(M_j(t_j)-1)}$ is convex in Λ_j. Let $H_j = \frac{1-\rho_j}{1 - \Lambda_j(M_j(t_j)-1)/t_j}$. We need to show that $\Lambda_j H_j$ is convex in Λ_j.

We will first show that H_j is increasing and convex in Λ_j. We note that H_j can be written as

$$H_j = \frac{1 - \Lambda_j C_1}{1 - \Lambda_j C_2}, \hspace{2cm} (34)$$

where $C_1 = \frac{1}{\alpha_j} + \beta_j$ and $C_2 = \frac{M_j(t_j)-1}{t_j}$. Further $C_2 \geq C_1$ since $M_j(t_j) - 1 = \mathbb{E}[e^{t_j X_j}] - 1 \geq \mathbb{E}[1 + t_j X_j] - 1 = t_j\mathbb{E}[X_j] = t_j\left(\frac{1}{\alpha_j} + \beta_j\right)$. Differentiating H_j w.r.t. Λ_j, we have

$$\frac{\delta}{\delta\Lambda_j}H_j = \frac{C_2 - C_1}{(1 - \Lambda_j C_2)^2} \geq 0 \hspace{1cm} (35)$$

$$\frac{\delta^2}{\delta\Lambda_j^2}H_j = 2C_2\frac{C_2 - C_1}{(1 - \Lambda_j C_2)^3} \geq 0. \hspace{0.5cm} (36)$$

Thus, H_j is an increasing convex function of Λ_j. Since Λ_j is also an increasing convex function of Λ_j and the product of two increasing convex functions is convex, the result follows. $\square$

V. ALGORITHM FOR WLTP OPTIMIZATION

We note that the WLTP optimization problem is convex with respect to individual $\mathbf{t}$ and $\boldsymbol{\pi}$. We note that the strict $<$ constraint can be modified as $\leq -\epsilon$ for an ϵ small

enough. The constraints are also convex in each of the variables individually. We will now develop an alternating minimization algorithm to solve the problem. In order to describe the Algorithm, we first define the three sub-problems:

t-Optimization: Input $\boldsymbol{\pi}, \mathcal{S}$

$$
\begin{aligned}
\min\quad & \sum_i \omega_i\left(\sum_j \frac{\pi_{ij}}{e^{t_j x}}\frac{(1-\rho_j)t_j M_j(t_j)}{t_j - \Lambda_j(M_j(t_j)-1)}\right) \\
s.t.\quad & (17),(18),(19),(24),(25) \\
var.\quad & \mathbf{t}
\end{aligned}
$$

$\boldsymbol{\pi}$-Optimization: Input $\mathbf{t}, \mathcal{S}$

$$
\begin{aligned}
\min\quad & \sum_i \omega_i\left(\sum_j \frac{\pi_{ij}}{e^{t_j x}}\frac{(1-\rho_j)t_j M_j(t_j)}{t_j - \Lambda_j(M_j(t_j)-1)}\right) \\
s.t.\quad & (17),(18),(19),(20),(21),(22),(25) \\
var.\quad & \boldsymbol{\pi}
\end{aligned}
$$

$\mathcal{S}$-Optimization: Input $\mathbf{t}, \boldsymbol{\pi}$

$$
\begin{aligned}
\min\quad & \sum_i \omega_i\left(\sum_j \frac{\pi_{ij}}{e^{t_j x}}\frac{(1-\rho_j)t_j M_j(t_j)}{t_j - \Lambda_j(M_j(t_j)-1)}\right) \\
s.t.\quad & (17),(18),(19),(20),(21),(22),(23) \\
var.\quad & \mathcal{S}
\end{aligned}
$$

The first two sub-problems (**t-Optimization** and **$\boldsymbol{\pi}$-Optimization**) are convex, and thus can be solved by Projected Gradient Descent Algorithm.

For the placement sub-problem ($\mathcal{S}$-**Optimization**), we consider optimizing over $\mathcal{S}$ for each file request separately with fixed $\boldsymbol{\pi}$ and $\mathbf{t}$. We first rewrite the latency tail probability for file i, $\mathbb{P}(L_i \geq x)$ as follows

$$
\begin{aligned}
\mathbb{P}(L_i \geq x) & \leq \sum_j \frac{\pi_{ij}}{e^{t_j x}}\frac{(1-\rho_j)t_j M_j(t_j)}{t_j - \Lambda_j(M_j(t_j)-1)} \hspace{0.3cm} (37) \\
& = \sum_j \frac{\pi_{ij}}{e^{t_j x}}F\left(\Lambda_j\right), \hspace{1cm} (38)
\end{aligned}
$$

where $F\left(\Lambda_j\right) = \frac{(1-\Lambda_j(1/\alpha_j+\beta_j))t_j M_j(t_j)}{t_j - \Lambda_j(M_j(t_j)-1)}$. To show that the placement sub-problem can be cast into a bipartite matching, we consider the problem of placing file i chunks on m available servers. Note that placing the chunks is equivalent to permuting the existing access probabilities $\{\pi_{ij}, \forall i\}$ on all m servers, because $\pi_{ij} > 0$ only if a chunk of file i is placed on server j. Let β be a permutation of m elements. Under new placement defined by β, the new probability of accessing file i chunk on server j becomes $\pi_{i,\beta(j)}$. Thus, our objective in this sub-problem is to find such a placement (or permutation) $\beta(j) \forall j$ that minimizes the average tail probability, which can be solved via a matching problem between the set of existing scheduling

---

Algorithm 1 Proposed algorithm for solving latency tail probability Problem

---
Initialize $k = 0$, $\epsilon > 0$,
Initialize feasible $\{t_i(0), \pi_{i,j}(0), \}$
while obj (k) − obj $(k-1) \geq \epsilon$
   //*Solve scheduling, auxiliary variables and placement with given* $\{t_i(k), \pi_{i,j}(k), \mathcal{S}_i(k)\}$
   Step 1: $\boldsymbol{t}(k+1) = \underset{\boldsymbol{t}}{\text{argmin}}(16)$ s.t. (17),(18),(19),(24),(25)
   Step 2: $\boldsymbol{\pi}(k+1) = \underset{\boldsymbol{\pi}}{\text{argmin}}(16)$ s.t. (17),(18),(19),(20), (21),(22),(25)
   //*Solve placement with given* $\{\boldsymbol{t}(k+1), \boldsymbol{\pi}(k+1)\}$
   Step 3:
     κ = random permuation for files $(1, \ldots, r)$
     for $y = 1, \cdots, r$
       $i = \kappa(y)$
       Calculate $\Lambda_j^{(-i)}$ using $\pi_{ij}(k+1)$
       Calculate $D_{u,v}$ from (39).
       $(\beta(u)\ \forall j) = Hungarian\ Algorithm(D_{u,v})$
       Update $\pi_{i,\beta(u)}(k+1) = \pi_{i,u}(k+1)\ \forall i,j$.
       $\mathcal{S}_i(k+1) = \{\beta(u) \forall u \in \mathcal{S}_i(k)\}$
     end for
end while
output: $\{\boldsymbol{\pi}, \mathcal{S}, \boldsymbol{t}\}$

---

probabilities $\{\pi_{ij}, \forall i\}$ and the set of m available servers, with respect to their load excluding the contribution of file i itself. Let $\Lambda_j^{-i} = \Lambda_j - \lambda_i \pi_{ij}$ when request for file i, be the total request rate at server j, excluding the contribution of file i. We define a complete bipartite graph $\mathcal{G}_r = (\mathcal{U}, \mathcal{V}, \mathcal{E})$ with disjoint vertex sets $\mathcal{U}, \mathcal{V}$ of equal size m and edge weights given by

$$D_{u,v} = \sum_i \frac{\lambda_i \pi_{iu}}{e^{t_u x}} F\left(\Lambda_v^{-i} + \lambda_i \pi_{iu}\right), \ \forall u, v, \quad (39)$$

which quantifies the contribution to overall latency tail probability by assigning existing π_{iu} to server v that has an existing load Λ_j^{-i}. It can be shown that a minimum-weight matching on $\mathcal{G}_r$ finds an optimal β to minimize

$$\sum_{u=1}^m D_{u,\beta(u)} = \sum_{u=1}^m \sum_{i=1}^r \frac{\lambda_i \pi_{iu}}{e^{t_j x}} F\left(\Lambda_{\beta(u)}^{-i} + \lambda_i \pi_{iu}\right), \quad (40)$$

The proposed algorithm for solving latency tail probability problem is shown in Algorithm 1, where the order of files whose placements are optimized one after the other are chosen at random. Note that the order of the files whose decisions are changed make a difference. In the proposed algorithm, we take a single pass over the files since there is an outer loop of alternating minimization.

Since the objective is non-negative and the objective is non-increasing in each iteration, the algorithm converges.

VI. NUMERICAL RESULTS

To validate the proposed tail latency bound and tail latency optimization, we employ a hybrid simulation method, which generates chunk service times based on real system measurements on Tahoe and Amazon S3 servers in [13, 18, 25], and compare the performance of our proposed latency optimization, denoted as WLTP Policy, with five baseline strategies. The proposed strategy and the other baseline strategies are described below.

- Proposed Approach-Optimized Placement, i.e., WLTP (*Weighted Latency Tail Probability*) Policy: The joint scheduler is determined by the proposed solution that minimizes the weighted latency tail probabilities, with respect to the three sets of variables: chunk placement on the servers $\mathcal{S}$, auxiliary variables $\mathbf{t}$, and the scheduling policy $\boldsymbol{\pi}$.
- Proposed Approach-Random Placement, i.e., WLTP-RP (*WLTP - Random Placement*) Policy: The chunks are placed uniformly at random. With this fixed placement, the weighted latency tail probability is optimized over the remaining two sets of variables: auxiliary variables $\mathbf{t}$, and the scheduling policy $\boldsymbol{\pi}$.
- WLTP-RP-Fixed $\mathbf{t}$ Policy: The chunks are placed uniformly at random, and all the auxiliary variables t_j are set as 0.01. The weighted latency tail probability is optimized over the scheduling access probabilities $\boldsymbol{\pi}$.
- PEAP (*Projected, Equal Access-Probability*) Policy: For each file request, the joint request scheduler selects available nodes with equal probability. This choice of $\pi_{i,j} = k_i/n_i$ may not be feasible and thus the access probabilities are projected toward feasible region in (16) for all $t_j = .01$ for a uniformly randomly placed files to ensure stability of the storage system. With these fixed access probabilities, the weighted latency tail probability is optimized over the remaining two sets of variables: chunk placement on the servers $\mathcal{S}$, and the auxiliary variables $\mathbf{t}$.
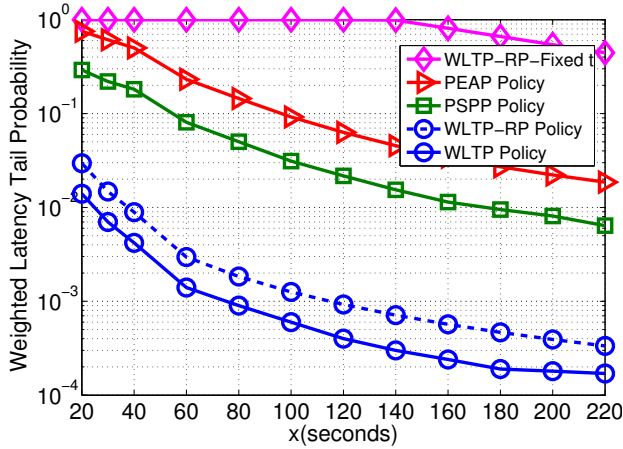- PEAP-RP Policy: As compared to the PEAP Policy, the chunks are placed uniformly at random. The weighted latency tail probability is optimized over the choice of auxiliary variables $\mathbf{t}$.
- PSPP (*Projected Service-Rate Proportional Allocation*) Policy: The joint request scheduler chooses the access probabilities to be proportional to the service rates of the storage nodes, i.e., $\pi_{ij} = k_i \frac{\mu_j}{\sum_j \mu_j}$. This policy assigns servers proportional to their service rates. These access probabilities are projected toward feasible region in (16) for a uniformly random placed files to ensure stability of the storage system. With these fixed access probabilities, the weighted latency tail probability is optimized over the remaining two sets of variables: chunk placement on the servers $\mathcal{S}$, and the auxiliary variables $\mathbf{t}$.

Fig. 2. Weighted Latency Tail Probability vs x (in seconds).

Fig. 3. Convergence of Weighted Latency Tail Probability.

- PSPP-RP (*PSPP - Random Placement*) Policy: As compared to the PSPP Policy, the chunks are placed uniformly at random. The weighted latency tail probability is optimized over the choice of auxiliary variables $\mathbf{t}$.

In the simulations, we consider $r = 1000$ files, all of size 200 MB and using $(7,4)$ erasure code in a distributed storage system consisting of $m = 12$ distributed nodes. Based on [13, 18, 25], we consider chunk service time that follows a shifted-exponential distribution with rate α_j and shift β_j. As shown in Table II, we have 12 heterogeneous storage nodes with different service rates α_j and shifts β_j. The base arrival rates for the first 250 files is $2/150$ s^{-1}, for the next 250 files are $4/150$ s^{-1}, for the next 250 files are $6/150$ s^{-1}, and for the last 250 files is $3/150$ s^{-1}. This paper also considers the weights of the files proportional to the arrival rates.

In order to initialize the algorithm, we choose $\pi_{ij} = k/n$ on the placed servers, all $t_j = .01$. However, since these choices of π and $\mathbf{t}$ may not be feasible, we modify the initialization π to be the closest norm feasible solution to the above choice.

TABLE II
SUMMARY OF PARAMETERS FOR THE 12 STORAGE NODES IN OUR SIMULATION (SHIFT β IN MS AND RATE α IN 1/S).

	Node 1	Node 2	Node 3	Node 4	Node 5	Node 6
α_j	20.0015	26.1252	14.9850	17.0526	27.1422	22.8919
β_j	10.5368	15.6018	8.2756	10.0120	12.8544	13.6722

	Node 7	Node 8	Node 9	Node 10	Node 11	Node 12
α_j	30.0000	21.3812	11.9106	25.1599	28.8188	23.8067
β_j	12.6616	9.9156	10.7872	8.6166	13.8721	10.8964

Weighted Latency Tail Probabilities: In Figure 2, we plot the decay of weighted latency tail probability $\sum_i \omega_i \Pr(L_i \geq x)$ with x (in seconds) for Policies WLTP, WLTP-RP, PSPP, PEAP, and WLTP-RP-Fixed $\mathbf{t}$. Notice that WLTP Policy solves the optimal weighted latency
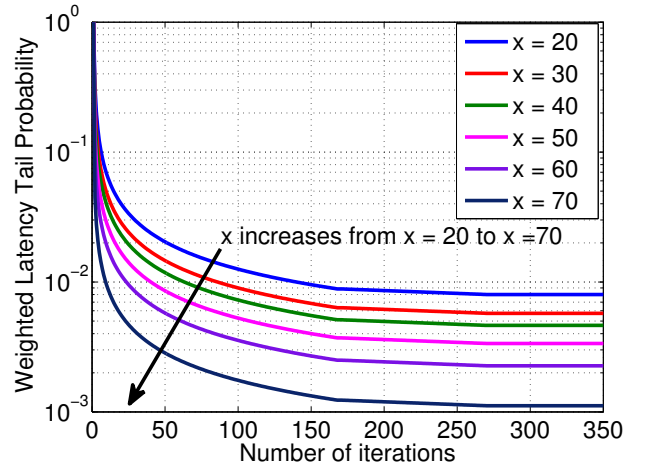
tail probability via proposed alternative optimization algorithm over π, $\mathbf{t}$, and $\mathcal{S}$. With optimized $\mathbf{t}$ and placement, Policy PEAP uses equal server access probabilities, projected toward the feasible region, while Policy PSPP assign chunk requests across different servers proportional to their service rates. The values of $\mathbf{t}$ are then found optimally for the above given values of $\pi_{i,j}$. Note that this figure also represents the complementary cumulative distribution function (ccdf) of the WLTP, WLTP-RP, WLTP-RP-Fixed $\mathbf{t}$, PSPP, and PEAP. For instance, We observe that $\Pr(x \geq 20) \approx 0.01$ for our proposed policy WLTP which is significantly lower as compared to the other strategies.

We note that our proposed algorithm for jointly optimizing π, $\mathbf{t}$ and $\mathcal{S}$ provides significant improvement over simple heuristics such as Policies WLTP-RP-Fixed $\mathbf{t}$, PSPP, and PEAP, as weighted latency tail probability reduces by orders of magnitude. For example, our proposed Policy WLTP decreases 99-percentile weighted latency (i.e., x such that $\sum_i \omega_i \Pr(L_i \geq x) \leq 0.01$) from above 160 seconds in the baseline policies to about 20 seconds using WLTP. We also notice that chunk placement optimization reduces the latency tail probability for all x, as can be seen from Figure 2 among the policies WLTP and WLTP-RP. Uniformly accessing servers and simple service-rate-based scheduling are unable to optimize the request scheduler based on factors like chunk placement, request arrival rates, different latency weights, thus leading to much higher tail latency. Since the policy WLTP-RP-Fixed $\mathbf{t}$ performs significantly worse than the other considered policies, we do not include this policy in the rest of the paper.

Convergence of the Proposed Algorithm: We have shown that the proposed algorithm converges within about 350 iterations to the optimal objective value, validating the efficiency of the proposed optimization algorithm. To illustrate its convergence speed, Figure 3 shows the con-
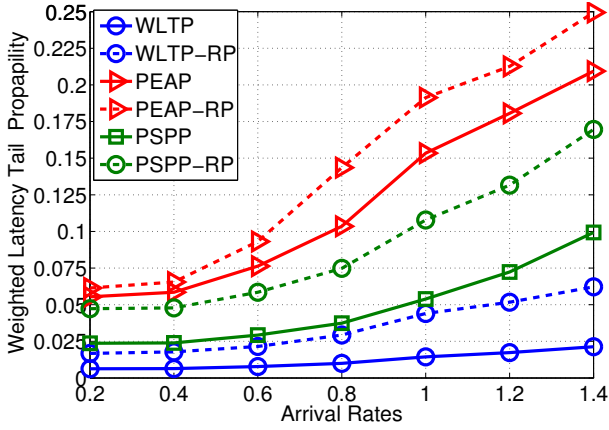
Fig. 4. Weighted Latency Tail Probability for different file arrival rates. We vary the arrival rate of file i from $0.2 \times \lambda$ to $1.4 \times \lambda$, where λ is the base arrival rate.

Fig. 5. Weighted Latency Tail Probability for different number of files.

Fig. 6. Weighted Latency Tail Probability for different file sizes.

vergence of objective value vs. the number of iterations for different values of x ranging from 20 to 70 seconds in increments of 10 seconds. In the rest of the results, 350 iterations will be used to get the required results.

Effect of Arrival Rates: We next see the effect of varying request arrival rates on the weighted latency tail probability. We choose $x = 50$ seconds. For λ as the base arrival rates, we increase arrival rate of all files from $.2\lambda$ to $1.4 \times \lambda$ and plot the weighted latency tail probability in Figure 4. While latency tail probability increases as arrival rate increases, our algorithm assigns differentiated latency for different files to maintain low weighted latency tail probability. We compare our proposed algorithm with the different baseline policies and notice that the proposed algorithm outperforms all baseline strategies.

Since the weighted latency tail probability is more significant at high arrival rates, we observe significant improvement in latency tail by about a multiple of 9 (approximately 0.025 to about 0.22) at the highest arrival rate in Figure 4 between PEAP and WLTP policies. Our proposed policy always receives the minimum latency. Thus, efficiently reducing the latency of the high arrival rate files reduces the overall weighted latency tail probability.

Effect of Number of files: Figure 5 demonstrates the impact of varying the number of files from 200 to 1200 on the weighted latency tail probability. Weighted latency tail probabilities increases with the number of files, which brings in more workload (i.e., higher arrival rates). Our optimization algorithm optimizes new files along with existing ones to keep overall weighted latency tail probability at a low level. We note that the proposed optimization strategy effectively reduces the tail probability and outperforms the considered baseline strategies. Thus, joint optimization over all three variables $\mathcal{S}$, π, and t helps reduce the tail probability significantly.
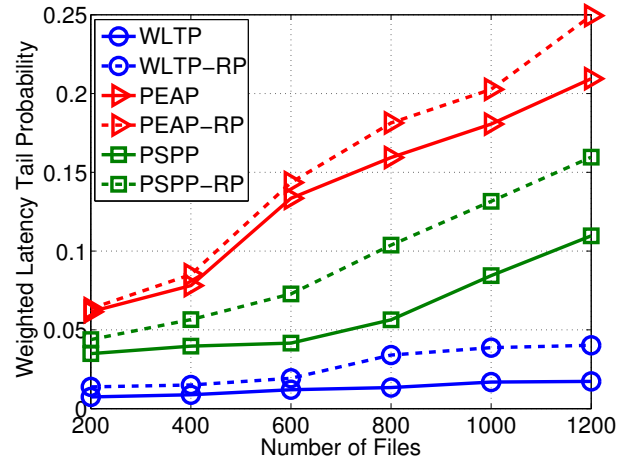
Effect of File Sizes: We vary the file size in our simulation from 200MB to 700MB, and plot the optimal weighted latency tail probability with varying file size in Figure 6. In order to capture the effect of increased file size as compared to a default size of 200 MB, we increase the value of parameters α and β proportionally to the chunk size in the shifted-exponential service time distribution. While increasing file size results in higher weighted tail latency probability for files, we compare our proposed algorithm with the baseline policies and verified that the proposed optimization algorithm offers significant reduction in tail latency.

Effect of the Tail Latency Weights: We next show the effect of varying the weights (i.e., w_i's) on the weighted tail latency probability for the proposed WLTP policy. We choose $x = 40$ seconds. Recall that the arrival rate of files was divided into four groups, each with different arrival rates. We vary the arrival rate of all files from $.4\lambda$ to λ with a step of 0.2λ and plot the weighted latency tail probability for each group of 250 files as well as the overall value in
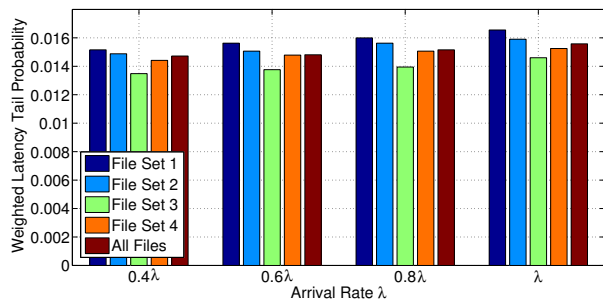
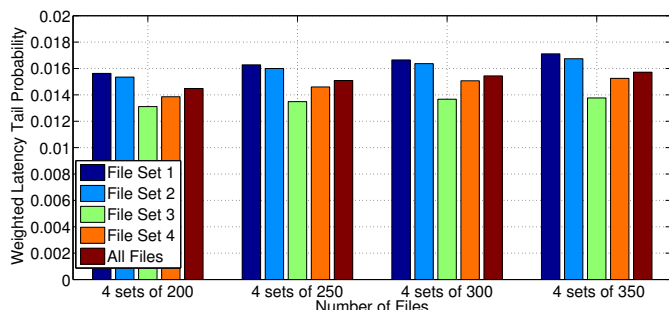Fig. 7. Weighted Latency Tail Probability for different file arrival rates.

Fig. 8. Weighted Latency Tail Probability for different file arrival rates.

Figure 7. While weighted latency tail probability increases as arrival rate increases, our algorithm assigns differentiated latency for different file groups. Group 3 that has highest weight ω_3 (i.e., most tail latency sensitive) always receive the minimum latency tail probability even though these files have the highest arrival rate. Thus, efficiently reducing the latency of the high arrival rate files reduces the overall weighted latency tail probability. We note that efficient access probabilities π help in differentiating file latencies as compared to the strategy where minimum queue-length servers are selected to access the content obtaining lower weighted tail latency probability.

Figure 8 shows the effect of the file weights on the weighted latency tail probability for varying number of files. In particular, we modify the number of files in each group from 250 in the base case to values such as 250, 300, and 350, as shown in Figure 8. Apparently, the weighted tail latency probability increases with the number of files, since that increases the workload in terms of the arrival rates. Our optimization algorithm optimizes the placement and access of the files to keep weighted tail latency probability lower. As noted earlier, the higher arrival rate group has lower tail latency probability thus reducing the overall objective.

VII. ACKNOWLEDGMENT

VIII. CONCLUSIONS

This paper provides bounds on latency tail probabilities for distributed storage systems using erasure coding. These bounds are used to formulate an optimization to jointly minimize weighted latency tail probability of all files over request scheduling and data placement. The non-convex optimization problem is solved using an efficient alternating optimization algorithm. Simulation results show significant reduction of tail latency for erasure-coded storage systems with realistic workload.

Following this work, the probabilistic scheduling used in this paper has been shown to be optimal for tail index in [31]. However, the model of file size is different as compared to this paper. Finding more general scenarios where such scheduling strategy is optimal, or improving the strategy to show optimality for wider classes is an important research problem.

REFERENCES

[1] V. Aggarwal, J. Fan, and T. Lan, "Taming tail latency for erasure-coded, distributed storage systems," in *Proc. IEEE Infocom*, Jul 2017.

[2] M. Sathiamoorthy, M. Asteris, D. Papailiopoulos, A. G. Dimakis, R. Vadali, S. Chen, and D. Borthakur, "Xoring elephants: Novel erasure codes for big data," in *Proceedings of the 39th international conference on Very Large Data Bases.*, 2013.

[3] C. Huang, H. Simitci, Y. Xu, A. Ogus, B. Calder, P. Gopalan, J. Li, and S. Yekhanin, "Erasure coding in windows azure storage," in *Proceedings of the 2012 USENIX Conference on Annual Technical Conference*, ser. USENIX ATC'12. USENIX Association, 2012.

[4] A. Fikes, "Storage architecture and challenges (talk at the google faculty summit)," http://bit.ly/nUylRW, Tech. Rep., 2010.

[5] J. Dean and L. A. Barroso, "The tail at scale," in *Communications of the ACM*, 2013.

[6] Y. Xu, Z. Musgrave, B. Noble, and M. Bailey, "Bobtail: Avoiding long tails in the cloud," in *10th USENIX Symposium on Networked Systems Design and Implementation (NSDI 13)*, 2013.

[7] G. Liang and U. C. 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, Dec. 2014. [Online]. Available: http://dx.doi.org/10.1109/TNET.2013.2289382

[8] L. A. Barroso, "Warehouse-scale computing: Entering the teenage decade," in *Proceedings of the 38th Annual International Symposium on Computer Architecture*, ser. ISCA '11. New York, NY, USA: ACM, 2011, pp. –. [Online]. Available: http://dl.acm.org/citation.cfm?id=2000064.2019527

[9] L. Huang, S. Pawar, H. Zhang, and K. Ramchandran, "Codes can reduce queueing delay in data centers," in *Information Theory Proceedings (ISIT), 2012 IEEE International Symposium on*, July 2012, pp. 2766–2770.

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

[11] K. Lee, N. B. Shah, L. Huang, and K. Ramchandran, "The mds queue: Analysing the latency performance of erasure codes," *IEEE Transactions on Information Theory*, 2017.

[12] Y. Xiang, T. Lan, V. Aggarwal, and Y.-F. R. Chen, "Joint latency and cost optimization for erasure-coded data center storage," in *Proceedings of IFIP WG 7.3 Performance 2014*, 2014.

[13] Y. Xiang, T. Lan, V. Aggarwal, and Y. F. R. Chen, "Joint latency and cost optimization for erasure-coded data center storage," *IEEE/ACM Transactions on Networking*, vol. 24, no. 4, pp. 2443–2457, Aug 2016.

[14] K. Gardner, S. Zbarsky, M. Velednitsky, M. Harchol-Balter, and A. Scheller-Wolf, "Understanding response time in the redundancy-d system," *ACM SIGMETRICS Performance Evaluation Review*, vol. 44, no. 2, pp. 33–35, 2016.

[15] 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*, p. 629660, 1989.

[16] G. Liang and U. Kozat, "Fast cloud: Pushing the envelope on delay performance of cloud storage with coding," *Networking, IEEE/ACM Transactions on*, vol. 22, no. 6, pp. 2012–2025, Dec 2014.

[17] ——, "Tofec: Achieving optimal throughput-delay trade-off of cloud storage using erasure codes," in *Proceedings of IEEE Infocom*, 2014.

[18] S. Chen, Y. Sun, U. Kozat, L. Huang, P. Sinha, G. Liang, X. Liu, and N. Shroff, "When queuing meets coding: Optimal-latency data retrieving scheme in storage clouds," in *Proceedings of IEEE Infocom*, 2014.

[19] A. Kumar, R. Tandon, and T. Clancy, "On the latency and energy efficiency of distributed storage systems," *IEEE Transactions on Cloud Computing*, vol. PP, no. 99, pp. 1–1, 2015.

[20] B. Li, A. Ramamoorthy, and R. Srikant, "Mean-field-analysis of coding versus replication in cloud storage systems," in *Computer Communications, IEEE INFOCOM 2016-The 35th Annual IEEE International Conference on*. IEEE, 2016, pp. 1–9.

[21] Y. Xiang, V. Aggarwal, Y.-F. Chen, and T. Lan, "Taming latency in data center networking with erasure coded files," in *Cluster, Cloud and Grid Computing (CCGrid), 2015 15th IEEE/ACM International Symposium on*, May 2015, pp. 241–250.

[22] Y. Xiang, T. Lan, V. Aggarwal, and Y.-F. Chen, "Multi-tenant latency optimization in erasure-coded storage with differentiated services," in *Distributed Computing Systems (ICDCS), 2015 IEEE 35th International Conference on*, June 2015, pp. 790–791.

[23] Y. Xiang, V. Aggarwal, Y.-F. Chen, and T. Lan, "Differentiated latency in data center networks with erasure coded files through traffic engineering," *IEEE Transactions on Cloud Computing*, 2017.

[24] V. Aggarwal, Y.-F. Chen, T. Lan, and Y. Xiang, "Sprout: A functional caching approach to minimize service latency in erasure-coded storage," in *Distributed Computing Systems (ICDCS), 2016 IEEE 36th International Conference on*, June 2016.

[25] ——, "Sprout: A functional caching approach to minimize service latency in erasure-coded storage," *Submitted to IEEE/ACM Transactions on Networking, available at arXiv:1609.09827*, Sept 2016.

[26] H. Weatherspoon and J. Kubiatowicz, "Erasure coding vs. replication: A quantitative comparison," in *Peer-to-Peer Systems*, ser. Lecture Notes in Computer Science, P. Druschel, F. Kaashoek, and A. Rowstron, Eds. Springer Berlin Heidelberg, 2002, vol. 2429, pp. 328–337.

[27] A. Abdelkefi and Y. Jiang, "A structural analysis of network delay," in *Communication Networks and Services Research Conference (CNSR), 2011 Ninth Annual*, May 2011, pp. 41–48.

[28] 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 Conference on File and Storage Technologies*, ser. FAST'04. Berkeley, CA, USA: USENIX Association, 2004, pp. 1–1. [Online]. Available: http://dl.acm.org/citation.cfm?id=1973374.1973375

[29] B. Calder et al., "Windows azure storage: A highly available cloud storage service with strong consistency," in *Proceedings of the Twenty-Third ACM Symposium on Operating Systems Principles*, ser. SOSP '11. New York, NY, USA: ACM, 2011, pp. 143–157. [Online]. Available: http://doi.acm.org/10.1145/2043556.2043571

[30] L. Kleinrock, *Queueing Systems: Theory*, ser. A Wiley-Interscience publication. Wiley, 1976, no. v. 1. [Online]. Available: https://books.google.com/books?id=rUbxAAAAMAAJ

[31] V. Aggarwal and T. Lan, "Tail index for a distributed storage system with pareto file size distribution," *CoRR*, vol. abs/1607.06044, 2016. [Online]. Available: http://arxiv.org/abs/1607.06044