Optimizing Differentiated Latency in Multi-Tenant, Erasure-Coded Storage

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

***Abstract*—Erasure codes are widely used in distributed storage systems since they provide space-optimal data redundancy to protect against data loss. Despite recent progress on quantifying average service latency when erasure codes are employed, there is very little work on providing differentiated latency among multiple tenants that may have different latency requirements. This paper proposes a novel framework for providing and optimizing differentiated latency in erasure-coded storage by investigating two policies, weighted queue and priority queue, for scheduling tenant requests. For both policies, we quantify service latency for different tenant classes for homogeneous files with arbitrary placement and service time distributions. We develop an optimization framework that jointly minimizes differentiated latency over three decision spaces: 1) data placement; 2) request scheduling; and 3) resource management. Efficient algorithms harnessing bipartite matching and convex optimization techniques are developed to solve the proposed optimization. Our solution enables elastic service-level agreements to meet heterogeneous application requirements. We further prototype our solution with both queuing models applied in an open-source, cloud storage deployment that simulates three geographically distributed data centers through bandwidth reservations. Experimental results validate our theoretical delay analysis and show significant joint latency reduction for different classes of files, providing valuable insights into service differentiation and elastic quality of service in erasure-coded storage systems.**

***Index Terms*—Differentiated services, distributed storage, erasure codes, weighted queue, priority queue, quality of service, service-level agreements.**

I. Introduction

ERASURE coding has been increasingly adopted by storage systems such as Microsoft Azure and Google Cloud Storage due to better space efficiency, while achieving the same or better level of reliability compared to full data replication. As a result, the effect of coding on content retrieval latency in data-center storage system is drawing more and more significant attention these days. Google and Amazon have published that every 500 ms extra delay means a 1.2% user loss [2]. Optimizing cloud storage to meet heterogeneous service level objectives for latency among multiple tenants is an important and challenging problem. Yet, due to the lack of analytic latency models for erasure-coded storage for differentiated services, most of the literature is limited to the analysis and optimization of average service latency of all tenants, e.g., [8], [9], [13], and [16], failing to recognize cloud applications' heterogeneous preferences.

Providing the same level of latency to all applications is unsatisfactory - cloud tenants may find it either inadequate or too expensive to fit their specific requirements, which is shown to vary significantly [2]. In fact, elastic QoS is an integral part of cloud computing and one of its most attractive premises. To our best knowledge, however, the optimization of differentiated service latency in an erasure-coded storage system is an open problem. First, latency is largely affected by queueing, and the existence of multiple tenants causes queueing for all requests to share the same underlying infrastructure. Second, using (n, k) erasure coding, an access request can be served by any k-out-of-n data chunks. It requires solving a chunk placement and dynamic scheduling problem in line with tenants' latency requirements. In this paper, we study erasure-coded storage under two request management policies, priority queuing and weighted queuing. By quantifying service latency of these policies, we are able to propose a novel optimization framework that provides differentiated service latency to meet heterogeneous application requirements and to enable Elastic Service-level Agreements (SLAs) in cloud storage.

Much of the prior work on improving storage latency in an erasure-coded system is limited to the easier problem of analysing and optimizing average latency over all cloud tenants, regardless of their different latency preferences. For homogeneous files, Huang *et al.* [8] and Shah *et al.* [9] proposed a *block-t-scheduling* policy that only allows the first t requests at the head of the buffer to move forward in order to gain tractability. A separate line of work was developed using the fork-join queue [10]–[12] and provides different bounds of average service latency for erasure-coded storage systems [13]–[15]. Recently, a new approach to analyzing average latency in erasure-coded storage was proposed in [16]. It harnesses order statistic analysis and a new probabilistic scheduling policy to derive an upper bound of average latency in closed-form. Not only does this result supersede previous latency analysis [8], [9], [13] by incorporating multiple non-homogeneous files and arbitrary service time distribution, its closed-form quantification of service latency also enables a

Manuscript received May 5, 2016; revised October 12, 2016; accepted January 23, 2017. Date of publication January 24, 2017; date of current version March 9, 2017. This work was supported in part by the National Science Foundation under Grant no. CNS-1618335 and CSR-1320226. This paper was presented in part at the 35th International Conference on Distributed Computing Systems (ICDCS) 2015 [1]. The associate editor coordinating the review of this paper and approving it for publication was V. Fodor.

Y. Xiang and Y.-F. Chen are with AT&T Labs-Research, Bedminster, NJ 07921 USA (e-mail: yxiang@research.att.com; chen@research.att.com).

T. Lan is with the Department of ECE, George Washington University, Washington, DC 20052 USA (e-mail: tlan@gwu.edu).

V. Aggarwal is with the School of IE, Purdue University, West Lafayette, IN 47907 USA (e-mail: vaneet@purdue.edu).

Digital Object Identifier 10.1109/TNSM.2017.2658440

joint latency and storage cost minimization that can be efficiently solved via an approximate algorithm [16], [17]. The probabilistic scheduling policy has also been shown to be asymptotically optimal for tail latency index of heavy-tailed files [18].

However, all these prior works are focused on analyzing and optimizing average service latency, which is undesirable for a multi-tenant cloud environment where each tenant has a different latency requirement for accessing files in an erasure coded, online cloud storage. While customizing elastic service latency for the tenants is undoubtedly appealing, it also comes with great technical challenges and calls for a new framework for quantifying, optimizing, and delivering differentiated service latency in general erasure-coded storage. In this paper, we propose two classes of policies for providing differentiated latency and elastic QoS in multi-tenant, erasure-coded storage systems: non-preemptive priority queue and weighted queue for scheduling tenant requests. Both of which partition tenants into different service classes based on their delay requirement and apply differentiated management policy to file requests generated by tenants in each service class.

In particular, our first policy is modeled as a non-preemptive priority queue. That is, a file request generated by a tenant in high priority class (i.e., having more stringent delay requirements) can move ahead of all the low priority requests (i.e., generated by tenants in low priority class) waiting in the queue of storage nodes, but low priority requests in service are not interrupted by high priority requests. The second policy is modeled as a weighted queue, where file requests submitted by tenants in each class are buffered and processed in a separate first-come-first-serve queue at each storage node. The service rate of these queues are tunable with the constraints that they are non-negative, and sum to at most the service rate of the server. Tuning these weights allows us to provide differentiated service rate to tenants in different classes, therefore assigning differentiated service latency to tenants. In this paper, we apply probabilistic scheduling in [16] and derive closed-form latency bounds for the service policies that use non-preemptive priority queue and weighted queue, under arbitrary data placement and service time distributions.

The goal of this paper is to quantify service latency under these two policies and to develop an optimization framework to minimize differentiated service latency for multiple tenants. Using the latency analysis, we propose a novel optimization framework for minimizing differentiated service latency. More precisely, for priority queue policy, we formulate a joint latency minimization problem for all tenants over two dimensions, i.e., the placement of erasure-encoded file chunks on distributed storage nodes and the scheduling of chunk access requests, while the joint optimization for weighted queue policy further minimizes latency over weights assigned to each service class. We show that both problems are mixed-integer optimization and difficult to solve in general. To develop efficient algorithmic solutions, we identify that each problem can be decoupled into two sets of subproblems: one equivalent to a bipartite matching and the other proven to be convex. We develop a greedy algorithm to solve these subproblems optimally in an iterative fashion, therefore jointly minimizing services latency of all tenants in different service classes. This optimization gives rise to new challenges that cannot be addressed by scheduling algorithms in Map-reduce type of framework [3]–[6], because chunk placement in erasure-coded storage must be solved jointly with a request scheduling problem that selects k_i-out-of-n_i distinct chunks/servers with available data to service any request.

To validate our theoretical analysis and joint latency optimization for different tenants, we provide a prototype of the proposed algorithms in *Tahoe* [20], which is an open-source, distributed file system 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. One additional storage client was deployed to issue storage requests. From the experiment results, we first find that the service time distribution is proportional to the bandwidth of the server, which validates an assumption used in the analysis of the weighted queue latency. Further, the experiment results validate fast convergence of our differentiated latency optimization algorithms. We see that our algorithms efficiently reduce latency both with the priority and the weighted queues, and the results from the experiments are reasonably close to the given latency bounds for both the models. Finally, we note that priority queuing could lead to unfairness since the low priority tenants only share residual service rates left over by high priority tenants, while weighted queuing is able to balance service rates by optimizing weights assigned to each service class.

II. System Model

In this section we introduce the system model and the two queuing models that are used in this paper. We consider a data center consisting of m heterogeneous servers, denoted by $\mathcal{M} = \{j = 1, 2, \ldots, m\}$. A set of tenants store r heterogeneous files among those m servers. Each file is encoded with different erasure code as follows. We divide file i into k_i fixed-size chunks then encode it using an (n_i, k_i) Maximum Distance Separable (MDS) erasure code [19] to generate n_i distinct encoded chunks of equal size. Each encoded chunk is then stored on a different storage node. Thus, we denote $\mathcal{S}_i$ as the set of storage nodes hosting file i chunks, which satisfy $\mathcal{S}_i \subseteq \mathcal{M}$ and $n_i = |\mathcal{S}_i|$. Thus, each selected server will have only one chunk of file i. When processing a file-retrieving request, the (n_i, k_i) MDS erasure code allows the content to be reconstructed from any subset of k_i-out-of-n_i chunks, so we need to access only k_i chunks of file i to recover the complete file, and we have to find out which k_i of the n_i chunks should be accessed. We assume that the parameters k_i and n_i for each file are fixed. The value of k_i is determined by content size and the choice of chunk size.

We assume that the tenants' files are divided into 2 service classes, $\mathcal{R}_1$ for delay-sensitive files and $\mathcal{R}_2$ for non-delay-sensitive files. There are a series of file requests for the r files, such that requests for each file (or content) i arrive as a Poisson process with rate λ_i, $i = 1, \ldots, r$. For a given placement $(\mathcal{S}_i)$ for each file, and erasure-coded parameters (n_i, k_i), we will use

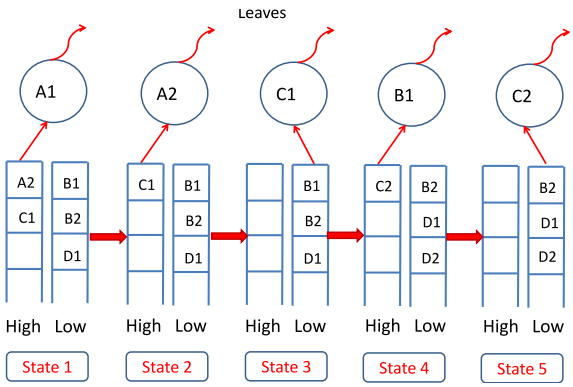


Fig. 1. System evolution for high/Low priority queuing.

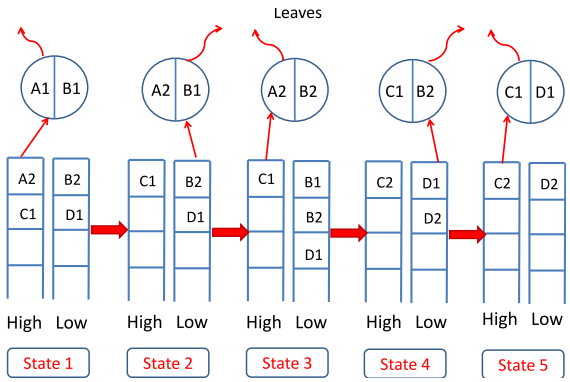


Fig. 2. System evolution for weighted queuing.

probabilistic scheduling to select k_i file chunks when a file is requested. There are n_i-choose-k_i options, and each option is chosen with certain probability. This scheduling strategy was proposed in [16], and an outer bound on the latency was provided. The probabilistic scheduling for all these n_i-choose-k_i options was shown in [16] to be equivalent to choosing each of the storage nodes in $\mathcal{S}_i$ with certain probability, say $\pi_{i,j}$. More formally, let $\pi_{i,j} \in [0, 1]$ be the probability that a request for file i will be forwarded to and served by storage node j, i.e.,

$$\pi_{i,j} = \mathbb{P}\Big[j \in \mathcal{S}_i \text{ is selected} \;\Big|\; k_i \text{ nodes are selected}\Big]. \quad (1)$$

It is easy to see that $\pi_{i,j} = 0$ if no chunk of content i is placed on node j, i.e., $j \notin \mathcal{S}$. Further, $\sum_j \pi_{i,j} = k_i$ since exactly k_i distinct storage nodes are needed to process each request. Next, we will describe the two queuing models that are used in this paper.

A. Priority Queuing

We assign a high priority for delay-sensitive files in $\mathcal{R}_1$ and a low priority for non-delay-sensitive files in $\mathcal{R}_2$. In order to serve the tenants with different priorities, priority queues involve having two sets of queues at each storage node - high priority queues and low priority queues. The requests made by the high priority tenant enter the high priority queues and the requests made by the low priority tenant enters the low priority queues. A chunk is serviced from high priority queue as long as there is a chunk in the queue, and a chunk is serviced from the low priority queue only if there is no chunk in the high priority queue. We assume a non-preemptive priority queue, where if a high priority request arrives during the waiting time of a low priority request, the arriving high priority request will be served first while the request which is already in service will not be affected by the later arrival of high priority requests.

An example is shown in Fig. 1, where a server has two queues, one for high priority requests and the other for low priority requests, which we denote as Q_h and Q_l respectively. At state 1, Q_h has two jobs A2 and C1 and Q_l has three jobs B1, B2 and D2 in queue, with job A1 in service. Since priority queues are used, a low priority request will only be served when the high priority queue is empty and thus after serving A1, A2 followed by C1 will be served, and all jobs in low priority queue will wait as depicted in state 2. In state 3, after C1 leaves, there is no new high priority request and thus B1 from Q_l will be served. State 4 shows a new high priority request C2 arrives during when B1 is being served. Due to the use of non-preemptive priority queue, the system will not disrupt service for B1. After $B1$ is served, C2 will be served before other low priority requests as depicted in State 5.

B. Weighted Queuing

Weighted queuing apportions service rate among different service classes in proportion to given weights. Tenants with higher weights receive more service rates, while tenants with lower weights can still receive their fair share if the weights are properly balanced. Assume that the total bandwidth B allocated to a server is divided among two service classes by weights w_1 and w_2, such that all tenants' files in class k gets bandwidth of $B_k = Bw_k$ for $k \in \{1, 2\}$, satisfying $\sum_{k=1}^{2} w_k = 1$. Thus, the system is equivalent to dividing each physical server into two logical servers, one with service bandwidth $B_1 = Bw_1$ and other with $B_2 = Bw_2$. We assume that the service time is inversely proportional to bandwidth such that service time with $B/2$ bandwidth will be twice as compared to that with a bandwidth of B. Unlike priority queuing, each server now is able to serve two requests from different classes at the same time, offering different service bandwidths for different classes. In this case, low-priority class may still get a small portion of bandwidth, without having to wait for all the high-priority jobs to finish.

To compare with the priority queuing model, we consider an example as shown in Fig. 2. We have the same set of jobs as in Fig. 1. We have A1, A2, C1, C2 as class 1 jobs, and B1, B2, D1, D2 as class 2 jobs, where class 1 is the class with larger weighted bandwidth resources and thus equivalent to a higher priority. Let the jobs in service at state 1 be A1 and B1. Each time a logical server for either class is free, the first job of that class in queue will be served at the service bandwidth of Bw_k, $k \in \{1, 2\}$, as shown in all states in Fig. 2. We can see from Fig. 2 that all class one requests are served during the 5 states, and 3 of the class 2 jobs are served. However, priority queuing has to wait for class one requests to be served before class 2, and thus fewer requests were served in the previous example. Thus, weighted queuing provides more fairness to low priority class customers.

III. Upper Bound: Probabilistic Scheduling

In this section, we will derive a latency upper bound for each file in the two service classes.

We consider two types of delay, *Queuing delay* which is denoted by $\mathbf{Q}_j$ and *Connection delay*, which is denoted by $\mathbf{N}_j$. Queuing delay is the waiting time a chunk request receives in node j due to sharing of network and I/O bandwidth, and is determined by service rates and arrival rates of chunk request at each storage node according to our queuing models. The connection delay includes the overhead to set up a connection from a tenant to node j in a certain data-center. We assume that the connection delay has a known mean η_j and variance ξ_j^2, and is independent of $\mathbf{Q}_j$. It is easy to see that if a subset $\mathcal{A}_i \subseteq \mathcal{S}_i$ of k_i nodes are selected to process a file-i request, its latency is determined by the maximum of queuing plus connection delay of the k_i nodes in $\mathcal{A}_i$. Therefore, under probabilistic scheduling we find average latency of file i as

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

where the expectation is taken over all system dynamics including queuing at each storage node and the random selection of $\mathcal{A}_i$ for each request with respect to probabilities $\pi_{i,j}\ \forall j$.

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 $\mathbb{E}[\mathbf{X}_\mathbf{j}^2] - \mathbb{E}[\mathbf{X}_\mathbf{j}^2] = \sigma_j^2$, second moment $\mathbb{E}[\mathbf{X}_\mathbf{j}^2] = \Gamma_{j,2}$, third moment $\mathbb{E}[\mathbf{X}_\mathbf{j}^3] = \Gamma_{j,3}$. Xiang *et al.* [16] gave an upper bound on $\bar{T}_i$ using its mean and variance as follows.

Lemma 1 [16]: The expected latency $\bar{T}_i$ of file i is tightly upper bounded by

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

where $\mathbf{D}_j = \mathbf{N}_j + \mathbf{Q}_j$ is the aggregate delay on node j with mean $\mathbb{E}[\mathbf{D}_j]$ and variance $\mathrm{Var}[\mathbf{D}_j]$.

We now consider the cases when different storage nodes manage chunk requests using priority or weighted queue models and derive upper bounds on service latency.

A. Latency Analysis With Priority Queues

According to our system model, we consider two priority queues for each storage node, one for requests of file in service class $\mathcal{R}_1$ (high priority class), and the other for requests in service class $\mathcal{R}_2$ (low priority class). Each file is either high priority or low priority and thus all chunk requests of the same file have the same priority level k, where $k = 1, 2$. We note that the queuing delay for different priority class requests is different. This is because low priority class requests have to wait for the queue of high priority class to finish. Thus, we need to find an upper bound to the expected latency of the files in the different priority classes using probabilistic scheduling. Let $\Lambda_{jk} = \sum_{i\in\mathcal{R}_k}\lambda_i\pi_{ij}$ be the aggregate arrival rate of class k requests on node j and $\rho_{jk} = \Lambda_{jk}/\mu_j$ the corresponding service intensity. We analyze non-preemptive priority queues on each node and use the result in Lemma 1 to obtain an upper bound on service latency for each file in the two service classes as follows.

Theorem 1: For non-preemptive priority queues, the expected latency $\bar{T}_{i,k}$ of file i of class k is upper bounded by $\bar{T}_i$ in (2), where $\mathbb{E}([\mathbf{D}_j]$ and $\mathrm{Var}[\mathbf{D}_j]$ for class k are denoted by $\mathbb{E}([\mathbf{D}_{jk}]$ and $\mathrm{Var}[\mathbf{D}_{jk}]$, respectively, and are given as follows.

$$\mathbb{E}\left(\left[\mathbf{D}_{j1}\right]\right) = \eta_j + \frac{1}{\mu_j} + \frac{\left(\Lambda_{j1}+\Lambda_{j2}\right)\Gamma_{j,2}}{2\left(1-\rho_{j1}\right)}, \quad (3)$$

$$\mathrm{Var}\left[\mathbf{D}_{j1}\right] = \xi_j^2 + \sigma_j^2 + \frac{\left(\Lambda_{j1}+\Lambda_{j2}\right)\Gamma_{j,3}}{3\left(1-\rho_{j1}\right)} + \frac{\left(\Lambda_{j1}+\Lambda_{j2}\right)^2\Gamma_{j,2}^2}{4\left(1-\rho_{j1}\right)^2}, \quad (4)$$

$$\mathbb{E}\left[\mathbf{D}_{j2}\right] = \eta_j + \frac{1}{\mu_j} + \frac{\left(\Lambda_{j1}+\Lambda_{j2}\right)\Gamma_{j,2}}{2\left(1-\rho_{j1}\right)\left(1-\rho_{j1}-\rho_{j2}\right)}, \quad (5)$$

$$\mathrm{Var}\left[\mathbf{D}_{j2}\right] = \xi_j^2 + \sigma_j^2 + \frac{\left(\Lambda_{j1}+\Lambda_{j2}\right)\Gamma_{j,3}}{3\left(1-\rho_{j1}\right)^2\left(1-\rho_{j2}\right)} + \frac{\left(\Lambda_{j1}+\Lambda_{j2}\right)^2\Gamma_{j,2}^2}{4\left(1-\rho_{j1}\right)^2\left(1-\rho_{j2}\right)^2} + \frac{\Lambda_{j1}\left(\Lambda_{j1}+\Lambda_{j2}\right)\Gamma_{j,2}^2}{2\left(1-\rho_{j1}\right)^3\left(1-\rho_{j2}\right)} \quad (6)$$

Theorem 1 quantifies the mean and variance of the delay for retrieving a file chunk of class k from each server j and then employs the order statistics bound in Lemma 1 to derive a bound on the maximum latency to retrieve k_i chunks from distinct servers. In our non-preemptive queuing model, any requests belonging to high priority class $\mathcal{R}_1$ are moved to the head of line and processed as soon as the storage server becomes free. Thus, these high priority requests are processed as if low priority class requests do not exist, except for a residual service time to complete any request that has already started on the server. For low priority class $\mathcal{R}_2$, a new request is served only if there is no high priority request in the queue. The service time (i.e., queuing delay) consists of four parts: the residual service time to complete any active request, the total service time of high priority requests that are already in queue, the total service time of the low priority requests that are already in queue before target request's arrival, and the total service time of the high priority requests arrived during target request's waiting time. The detailed expressions of the moments of the queuing delays are presented in Appendix A.

B. Latency Analysis for Weighted Queuing

We consider the weighted queue policy, where each storage node employs a separate queue for each service class. Queuing delay $\mathbf{Q}_j^k$ for class k requests on node j depends on the queuing weights w_{jk} since service bandwidth on each storage node is shared among all queues in proportion to their assigned weights, whereas connection delay due to protocol overhead is independent of bandwidth allocation and remains unchanged.

Let μ_j be the overall service rate on node j. According to our weighted queuing model, class k requests on node j receive w_{jk} fraction of service bandwidth and therefore have an average service time $1/(w_{jk}\mu_j)$ per chunk request. Due to Poisson property of request arrivals, each weighted queue can be modeled as a M/G/1 queue, whose mean and variance can be found in closed-form.

Theorem 2: For weighted queues, the expected latency $\bar{T}_{i,k}$ of file i of class k is upper bounded by $\bar{T}_i$ in (2), where $\mathbb{E}([\mathbf{D}_j]$ and $\text{Var}[\mathbf{D}_j]$ for class k are denoted by $\mathbb{E}([\mathbf{D}_{jk}]$ and $\text{Var}[\mathbf{D}_{jk}]$, respectively, and are given as follows.

$$\mathbb{E}\big[\mathbf{D}_{jk}\big] = \eta_j + \frac{1}{\mu_j} + \frac{\Lambda_{kj}P_{kj}^2\Gamma_{j,2}}{2w_{jk}\big(w_{jk} - \Lambda_{kj}P_{kj}/\mu_j\big)} \tag{7}$$

$$\begin{aligned}\text{Var}\big[\mathbf{D}_{jk}\big] = {} & \xi_j^2 + \sigma_j^2 \\ & + \sum_{k=1}^{2} P_{kj}\left(\frac{\Lambda_{kj}P_{kj}\Gamma_{j,3}}{3w_{jk}^2\big(w_{jk} - \Lambda_{kj}P_{kj}/\mu_j\big)}\right. \\ & \left. + \frac{\Lambda_{kj}^2P_{kj}^2\Gamma_{j,2}^2}{4w_{jk}^2\big(w_{jk} - \Lambda_{kj}P_{kj}/\mu_j\big)^2}\right),\end{aligned} \tag{8}$$

where $\Lambda_{kj} = \sum_{i\in\mathcal{R}_k}\lambda_i\pi_{ij}$ is the arrival rate of class k files at the storage node k, $P_{kj} = \frac{\Lambda_{kj}}{\sum_k \Lambda_{kj}}$ is the proportion of class k files at storage node j.

With weighted queues, the two queues for different classes are M/G/1 queues with different arrival rates and service rates since we assumed that service time is proportional to the bandwidth. Thus, we can look at each queue separately and the analysis as in [16] can be done for each class. The mean and variance with weighted queues can be found by summing the mean and variance of each of the classes by weighing the sums with the probability that a file belongs to that particular class. The detailed proof is given in Appendix B.

IV. Joint Latency Optimization for Differentiated Services Classes

In this section we jointly optimize differentiated service latency for files in all service classes over three dimensions. First, we need to decide the chunk placement of each file i across the m servers based on the erasure code parameter (n_i, k_i). Second, when a client requests a file, a subset of k_i nodes hosting desired file chunks must be selected according to probabilities $\pi_{i,j}$ $\forall j$, i.e., the access probabilities for each chunk.[1] Finally, in the case of weighted queues, we also need to decide on the weights w_{jk} used for sharing service bandwidth among different classes of files. In this section, we will formulate the service latency optimization problem in erasure-coded storage using priority and weighted queues, and then propose the two algorithms, namely Algorithm JLOP (Joint Latency Optimization in Priority Queuing) and JLOW (Joint Latency and Weight Optimization), for solving the latency minimization for differentiated services classes in the model described in previous section. In an online environment with dynamic file arrivals and removals, we can solve Problems JLOP and JLOW repeatedly using the proposed algorithms to update system configurations on the fly.

[1]Note that it is possible to have $\pi_{i,j} = 0$ even if a chunk of file i is placed on node j, i.e., the chunk is only a cold copy to ensure fault-tolerance and is never accessed in latency optimization.

A. Latency Optimization for Priority Queues

For priority queues, we give preference to high-priority tenants by allowing them to occupy as many system resources as needed without consideration for low-priority requests, while the low-priority tenants share the remaining resources given decisions of high-priority tenants. Thus, we propose a two-stage optimization problem as follows. First, we jointly optimize the chunk placement and access probabilities for all files in the high-priority class to minimize service latency they receive. Then, latency for the low-priority files is minimized based on the existing traffic generated by the high-priority files.

Let $\hat{\lambda}_k = \sum_{i \text{ is file of priority class k}} \lambda_i$ be the total arrival rate for high priority requests, and thus $\lambda_i/\hat{\lambda}_k$ is the fraction of file i requests among the class k priority files. The average latency of all files in class k is given by $\sum_{i\in\mathcal{R}_k}(\lambda_i/\hat{\lambda}_k)\bar{T}_{ik}$, for $k = 1, 2$. Our goal is to minimize the latency of high priority files over their chunk placement and access probabilities regardless of low priority requests, and based on the decision, to update optimization variables of low priority files to minimize their latency under existing high priority file traffic. We formulate problem JLOP, which consists of 2 sequential optimization problems for $k = 1, 2$ as follows.

JLOP:

$$\min \sum_{i\in\mathcal{R}_k}\frac{\lambda_i}{\hat{\lambda}_k}\bar{T}_{ik} \tag{9}$$

$$\text{s.t.} \sum_{j=1}^{m}\pi_{i,j} = k_i, \quad \forall i \tag{10}$$

$$\pi_{i,j} \in [0,1],\ \pi_{i,j} = 0, \quad \forall k, \quad \forall j \notin \mathcal{S}_i, \tag{11}$$

$$|\mathcal{S}_i| = n_i \text{ and } \mathcal{S}_i \subseteq \mathcal{M} \tag{12}$$

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

We can see this optimization problem for $k = 1, 2$ is a mixed integer optimization as we have fixed n servers to select for chunk placement for each request. It is hard to solve in general, thus we propose to break this problem into two sub-problems: placement and scheduling, which have an easier-to-handle structure and can be solved efficiently.

We consider the sub-problem for optimizing scheduling probabilities and recognize that for fixed chunk placements, Problem JLOP is convex in π_{ij}. The scheduling algorithm for high priority class is convex, since the low priority class does not exist as seen by the high priority class, so the problem becomes the optimization with only one priority class, which can be easily proved that $\bar{T}_{ik}$ is convex over Λ_{jk} as shown in [16], and Λ_{jk} is a linear combination of π_{ij}, thus the optimization problem is convex in π_{ij} when other parameters are fixed. For low priority class, we have the following lemma.

Lemma 2 (Convexity of the Scheduling Sub-Problem for Low Priority Class): When $\{z, S_i\}$ are fixed, Problem JLPO is a convex optimization over probabilities $\{\pi_{i,j}\}$.

Now we consider the placement sub-problem that optimizing over S_i for each file request with fixed z, π_{ij} for all classes in priority queuing. We first rewrite latency bound $\bar{T}_{ik}$ as

$$\bar{T}_{ik} = z + \sum_{j \in S_i} \frac{\pi_{ij}}{2} F\big(z, \Lambda_{jk}\big) \tag{13}$$

where $F(z, \Lambda_{jk}) = A_{jk} + \sqrt{A_{jk}^2 + B_{jk}}$ is an auxiliary function with $A_{jk} = \mathbb{E}([\mathbf{D}_{jk}] - z$ and $B_{jk} = \mathrm{Var}[\mathbf{D}_{jk}]$. To show that the placement subproblem can be cast into a bipartite matching, we consider the problem of placing file i chunks on m available servers. It is easy to see that placing the chunks is equivalent to permuting the existing access probabilities $\{\pi_{i,j},\ \forall j\}$ on all m servers, because $\pi_{i,j} > 0$ 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 service latency, which can be solved via a matching problem between the set of existing scheduling probabilities $\{\pi_{ij},\ \forall j\}$ and the set of m available servers, with respect to their load excluding the contribution of file i itself. Let $\Lambda_{jk}^{-i} = \Lambda_{jk} - \lambda_i \pi_{ij}$ when request for file i is of priority class k, 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

$$D_{u,v} = \sum_i \frac{\lambda_i \pi_{iu}}{2\hat{\lambda}_k} F\Big(z, \Lambda_{vk}^{-i} + \lambda_i \pi_{iu}\Big), \quad \forall u, v \tag{14}$$

which quantifies the contribution to overall latency (in objective value (9) by assigning existing π_{iu} to server v that has an existing load Λ_{vk}^{-i}. It is easy to see that a minimum-weight matching on $\mathcal{G}_r$ finds an optimal β to minimize

$$\sum_{u=1}^{m} D_{u,\beta(u)} = \sum_{u=1}^{m} \sum_i \frac{\lambda_i \pi_{iu}}{2\hat{\lambda}_k} F\Big(z, \Lambda_{vk}^{-i} + \lambda_i \pi_{iu}\Big),$$

which is exactly the optimization objective of Problem JLOP (except for a constant) if a file chunk with access probability $\pi_{i,u}$ to a rack with existing load $\Lambda_{i,\beta(u)}^{-i}$.

Now we can propose our algorithm JLPO (described in Fig. 3), which solves the placement and scheduling sub-problems iteratively for each of the 2 priority classes. Algorithm JLPO finds a solution for the joint optimization by iteratively solving its sub-problems, i.e., requires scheduling and chunk placement for the two service classes. In Step 1, we solve a convex optimization to determine the optimal request scheduling for high priority class with all other decision variables fixed. Next, for given scheduling probabilities, Step 2 computes an optimal matching between file chunks and storage serves to minimize overall access latency. The same process is repeated in Steps 3 and 4 for the low priority service class, which also requires to solve a convex scheduling optimization (due to Lemma 2) and optimal matching for chunk placement. Finally, latency bounds for both high and low priority classes are updated via an optimization of z. The proposed algorithm is guaranteed to converge to a local optimal point of the joint optimization, since no local changes (in scheduling or placement alone) in any sub-problems is able to further improve the objective. The algorithm is guaranteed to converge because the optimality in solving each sub-problem ensures a sequence of monotonically decreasing latency objectives.

Algorithm JLOP :

Initialize $t = 0$, $\epsilon > 0$.
Initialize feasible $\{z(0), \pi_{i,j}(0), S_i(0)\}$.
while $O(t) - O(t-1) > \epsilon$
  // Solve scheduling and placement for high priority
  // class with given $\{z(t), \pi_{i,j}^1(t), \pi_{i,j}^2(t), S_i^1(t), S_i^2(t)\}$
  Step 0:for all high priority jobs
    // Solve scheduling for high priority class with given
    //$\{z(t), S_i(t),\}$
    Step 1: $\pi_{i,j}^1(t+1) = \arg\min_{\pi_{i,j}}$ **(9) s.t. (12), (11).**
    // Solve placement for high priority class with given
    // $\{z(t), \pi_{i,j}(t+1)\}$
    Step 2: for $i = 1, \ldots, r$
      Calculate $\Lambda_{i,j}^{-i}$ using $\{\pi_{i,j}(t+1)\}$.
      Calculate D_{jk} from (15).
      $(\beta(j) \forall j \in \mathcal{N})$=$Hungarian\_Algorithm(\{D_{jk}\})$.
      Update $\pi_{i,\beta(j)}(t+1) = \pi_{i,j}(t)\ \forall i, j$.
      Initialize $S_i^1(t+1) = \{\}$.
      for $j = 1, \ldots, N$
        if $\exists i$ s.t. $\pi_{i,j}^1(t+1) > 0$
          Update $S_i(t+1) = S_i(t+1) \cup \{j\}$.
        end if
      end for
    end for
  end for
  // Solve scheduling and placement for low priority
  // class with given $\{z(t), \pi_{i,j}^1(t), S_i^1(t)\}$
  Go To Step 0
  Step 3:for all low priority jobs
    // Solve scheduling for low priority class with given
    $\{z(t), S_i^1(t+1)\}$
    Go To Step 1
    // Solve placement for low priority class with given
    $\{z(t), \pi_{i,j}^1(t+1)\}$
    Go To Step 2
    Update $S_i^2(t+1)$ and $\pi_{i,j}^2(t+1)$
  end for
  // Update bound for both high/low priority class given
  // $\{\pi_{i,j}(t+1), S_i(t+1)\}$
  $z(t+1) = \arg\min_{z \in \mathbb{R}}(9)$.
  Update objective value $O^{(t+1)}$.
  Update $t = t + 1$.
end while
Output: $\{\mathcal{S}_r(t), \pi_{i,j} r^{(t)}, w_{i,j}(t)\}$

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

B. Latency Optimization for Weighted Queues

In the presence of weighted queues, we consider a joint optimization of all files in different service classes by minimizing a weighted aggregate latency. Let $\hat{\lambda}_k = \sum_{i \in \mathcal{R}_k} \lambda_i$, $\hat{\lambda} = \sum_k \hat{\lambda}_k$, and $\tilde{T}_{ik}$ be given by the upper bound on $\bar{T}_{i1}$ in Theorem 2. Then, we want to optimize the following.

$$\min \ \ C_1 \tilde{T}_1 + C_2 \tilde{T}_2 \tag{15}$$

$$\text{s.t.} \ \ \tilde{T}_k = \frac{\hat{\lambda}_k}{\hat{\lambda}} \sum_{i \text{ is file of class } i} \tilde{T}_{ik}, \tag{16}$$

$$\sum_{j=1}^{m} \pi_{i,j} = k_i, \ \ \pi_{i,j} \in [0, 1], \ \ \pi_{i,j} = 0 \tag{17}$$

$$\forall j \notin \mathcal{S}_i, \tag{18}$$

Algorithm JLWO :

Initialize $t = 0$, $\epsilon > 0$.
Initialize feasible $\{z(0), \pi_{i,j}(0), S_i(0), w_{jk}\}$.
while $O(t) - O(t-1) > \epsilon$
  // Solve bandwidth allocation for given $\{z(t), \pi_{i,j}(t), S_i(t)\}$
  $w_{jk}(t+1) = \arg\min_{w_{jk}}$ (15) s.t. (19).
  // Solve scheduling for given $\{z(t), S_i(t), w_{jk}(t+1)\}$
  $\pi_{i,j}(t+1) = \arg\min_{\pi_{i,j}}$ (15) s.t. (16), (17).
  // Solve placement for given $\{z(t), w_{jk}(t+1), \pi_{i,j}(t+1)\}$
  for $i = 1, \ldots, r$
    Calculate $\Lambda_{i,j}^{-r}$ using $\{\pi_{i,j}(t+1)\}$.
    Calculate D_{jk} from (14).
    $(\beta(j) \forall j \in \mathcal{N})$=$Hungarian\_Algorithm(\{D_{jk}\})$.
    Update $\pi_{i,\beta(j)}(t+1) = \pi_{i,j}(t)$ $\forall i, j$.
    Initialize $S_i(t+1) = \{\}$.
    for $j = 1, \ldots, N$
      if $\exists i$ s.t. $\pi_{i,j}(t+1) > 0$
        Update $S_i(t+1) = S_i(t+1) \cup \{j\}$.
      end if
    end for
  end for
  // Update bound for given $\{w_{jk}(t+1), \pi_{i,j}(t+1), S_i(t+1)\}$
  $z(t+1) = \arg\min_{z \in \mathbb{R}}$. (15).
  Update objective value $O^{(t+1)}$=(15).
  Update $t = t + 1$.
end while
Output: $\{\mathcal{S}_i(t), \pi_{i,j}(t), w_{jk}(t)\}$

Fig. 4. Algorithm JLWO: Our proposed algorithm for solving Problem JLWO.

$$\sum_{i=1}^{N} \sum_{j \neq i} w_{jk} = 1. \tag{19}$$

$$|\mathcal{S}_i| = n_i, \text{ and } \mathcal{S}_i^k \subseteq \mathcal{M} \tag{20}$$

$$\text{var. } \mathcal{S}_i, \ \pi_{i,j}, \ w_{jk}, \quad \forall i, j, k.$$

As we have to choose fixed n servers for chunk placement according to the (n, k) erasure code applied, problem JLWO is a mixed integer optimization and hard to have a computational solution. In order to solve this optimization problem, we first break Problem JLWO into three sub-problems: (i) a weight sub-problem for optimizing service bandwidth among different queues by choosing weights w_{jk}, (ii) a scheduling sub-problem for determining accessing probabilities $\pi_{i,j}$, and (iii) a placement sub-problem that select a subset S_i nodes to host encoded chunks of file i.

We first recognize the scheduling sub-problem is convex. It can be easily proven using the convexity of $\bar{T}_k$ over Λ_{jk} as shown in [16], and due to the fact that Λ_{jk} is a linear combination of π_{ij}. Second, as for the placement problem we again cast it into a matching, similar to the one proposed for priority queuing. It results in a bipartite matching that can be solved efficiently. Finally, we show that the weight sub-problem is convex with respect to w_{jk} in the following lemma.

Lemma 3: (Convexity of the bandwidth allocation sub-problem): When $\{z, \pi_{i,j}, S_i\}$ are fixed, Problem JLWO is a convex optimization over weights $\{w_{jk}\}$.

We propose Algorithm JLWO (described in Fig. 4) to solve the differentiated latency optimization in weighted queuing by iteratively computing optimal solutions to the three sub-problems. In particular, Step 1 finds optimal bandwidth allocation for fixed chunk placement and request scheduling via a convex optimization (due to Lemma 3) of weights w_i. Next, Step 2 solves optimal scheduling probabilities, and Step 3 solves optimal chunk placement using Bipartite matching. Since each step finds an optimal solution for a sub-problem, the iterative algorithm is guaranteed to generate a monotonic sequence of objective values and is converged to a local optimal solution.

V. Implementation and Evaluation

A. Tahoe Testbed

To validate our proposed algorithms for priority and weighted queuing models, and to evaluate their performance, we implemented the algorithms in *Tahoe* [20], which is an open-source, distributed file-system 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 experiment is done on a Tahoe testbed consists of three separated hosts in an Openstack cluster running Havana, where we reserved bandwidths between these hosts in different availability zones (using a bandwidth control tool from our Cloud QoS platform) to simulate three separate data centers. The virtual hosts (VM's) on the same host are simulated as real hosts that belong to the same availability zone and in one data center. Out of the virtual hosts distributed across the three simulated data centers, we ran 12 storage servers on 12 virtual hosts, and we ran one introducer on a separate virtual host and 1 extra-large client node (with sufficient cpu and memory capacities) as a virtual host to initiate all client requests. The 12 storage servers are evenly distributed onto the three data-centers; i.e., we have four VM instances(storage servers) in each of the three Openstack availability zones in our cluster. We had to use an extra-large client node since we had to initiate a large number of requests through the client, using multiple Tahoe ports to simulate multiple clients on this node. All VMs have a 100GB volume attached for storing chunks in the case of storage servers and clients, or meta information in the case of the introducer. Our Tahoe testbed is shown in Fig. 5.

B. Basic Experiment Setup

We use (7,4) erasure code in the Tahoe testbed throughout the experiments described in the implementation section; however, we have two experiment setups for the two different queuing models.

For priority queuing, Algorithm Priority provides chunk placement and request scheduling for both high priority and low priority classes, which we also denote as class 1 (high priority) and class 2 (low priority). We assign the arrival rates of the two classes and generate file requests of class 1 and class 2 one by one based on the arrival rates. File requests are divided into chunk requests and then dispatched to the servers. When chunk requests arrive at the server, class 2 requests enter at the end of the queue while class 1 requests enter at the

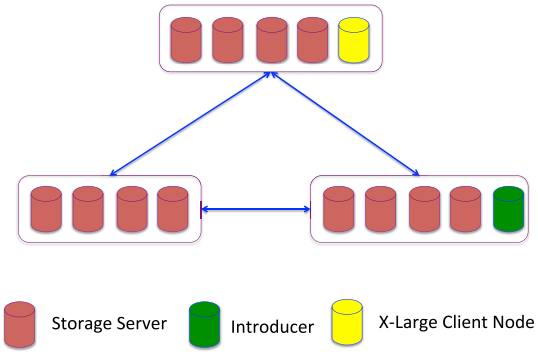


Fig. 5. Our Tahoe testbed with three hosts and each has 4 VMs as storage servers.

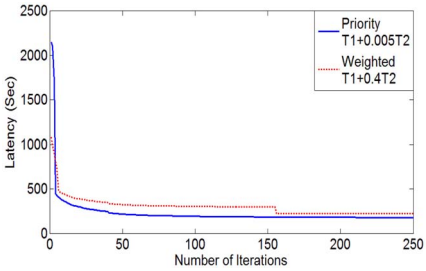


Fig. 6. Convergence of Algorithm Priority and Algorithm Weighted for heterogeneous files on our 12-node testbed. Both algorithms efficiently compute the solution in 175 iterations.

end of class 1 requests, before any class 2 request. Since we are using non-preemptive priority queuing, class 2 requests that are already in service before a class 1 request enters the queue will be allowed to be completed with no interference from class 1 requests. Chunk placement and request scheduling decisions that will minimize average latency for both classes will come from Algorithm JLOP.

For weighted queuing, requests are generated based on arrival rates for the two classes, then the system calls a bandwidth reservation tool to reserve the assigned bandwidth $Bw_{i,j}$ based on weights of each server from the algorithm, and then submit all the requests of the two classes. The system will calculate the chunk placement, request scheduling and weight assignment decisions from Algorithm JLWO.

C. Experiments and Evaluation

Convergence of Algorithm: We implemented Algorithm JLOP and Algorithm JLWO in MATLAB, where the convex optimization sub-problems are solved using MOSEK, a commercial optimization solver (even though one can implement them with any standard algorithm). For 12 distributed storage servers in our testbed, Figure 6 demonstrates the convergence of our algorithms, which optimizes the latency for the two classes in both queuing models over: chunk placement $\mathcal{S}_i$, request scheduling $\pi_{i,j}$ and bandwidth weights distribution $w_{i,j}$ (for weighted queuing model). As we have proven that algorithm JLOP and JLWO are both convex and the optimization is solvable earlier, now we see the convergence of the proposed optimized queuing algorithms in Fig. 6. Our algorithms for the two models efficiently solve the optimization

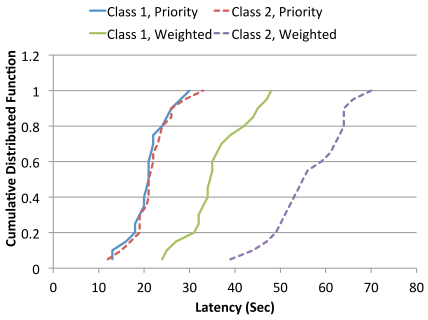


Fig. 7. Actual service time distribution for both classes of priority queuing and weighted queuing; each has $r = 1000$ files, each of size $100MB$ using erasure code (7,4) with the aggregate request arrival rate set to $\lambda_i = 0.28$ /sec in each model.

problem with $r = 1000$ files. For weighted queuing we set $C_1 = 1$, $C_2 = 0.4$ in the objective function. For priority queuing, we plot latency for the two classes in such a way that the latency for class 2 is multiplied by a factor of 0.05; this is necessary to have T_1 and T_2 on the same scale since class 2 experiences a high latency. It is observed that the normalized objective converges within 175 iterations for a tolerance $\epsilon = 0.01$. To achieve dynamic file management, our optimization algorithm can be executed repeatedly upon file arrivals and departures.

Validation of Experiment Setup: 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 for each class in both priority queuing and weighted queuing models on our testbed. We upload $r = 1000$ files of size $100MB$ file using a $(7, 4)$ erasure code. All the experiments for access were run to account for a 10-hour duration, which would account for efficient averaging. The experiment for priority queuing used an aggregate request arrival rate for both classes at 0.28/sec, out of which the arrival rate for class 1 request is 0.0122/sec and arrival rate for class 2 is 0.2678/sec. Thus, the ratio of class1/class2=1/22. This ratio is such that the interval between class 1 arrivals is similar to the chunk service time so that class 2 requests get a chance to be serviced. For weighted queuing, the aggregate request arrival rate for both classes is 0.28/sec, with each class having a rate of 0.14/sec, ratio of class1/class2 requests =1/1. Then we measure the chunk service time of class 1 and class 2 at the servers of both models. Bandwidth weights are 0.6 for class 1 and 0.4 for class 2 for weighted queuing. Figure 7 depicts the Cumulative Distribution Function (CDF) of the chunk service time for each class in each queuing model. We note that both class 1 and class 2 for priority queuing have a mean chunk service time of around 22 sec, and class 1 for weighted queuing has a mean chunk service time of 35 sec and class 2 for weighted queuing has a mean chunk service time of 60 sec, which means the chunk service time is proportional to the bandwidth reservation based on weight assignments for M/G/1 queues, thus validating our assumption for the analysis of weighted queues.

Validation of algorithms and joint optimization: In order to validate that the algorithms work for the two queuing models, we

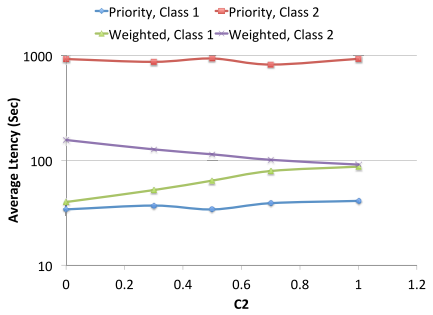


Fig. 8. $r = 1000$ files, each of size $100MB$, aggregate request arrival rate for both classes is 0.28/sec for both priority/weighted queuing; varying C_2 to validate our algorithms, weighted queuing provides more fairness to class 2 requests.

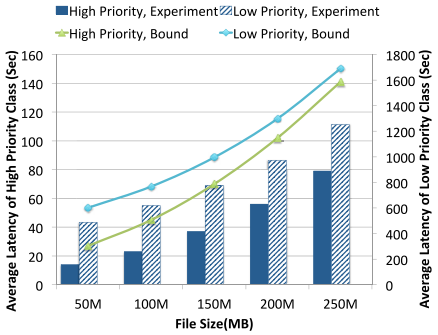


Fig. 9. Evaluation of different file sizes in priority queuing. Both experiment and bound statistics are using the secondary axis. Latency increases quickly as file size grows due to the queuing delay of both classes in priority queuing. Our analytic latency bound taking both network and queuing delay into account tightly follows actual service latency.

choose $r = 1000$ files of size $100MB$, and aggregate arrival rate 0.28/sec. For weighted queuing, we use ratio class1/class2=1:1, i.e., arrival rate of 0.14/sec for each class. We then upload heterogeneous files of size 100MB to obtain the service time statistics for class 1 and class 2 (including mean, variance, and second, third moment) at all storage nodes when running the two queuing models. We fix $C_1 = 1$ and vary C_2 from 0 to 1 and run both Algorithm Priority (not affected by C_2) and Algorithm JLOP and JLWO to generate the optimal solution for both priority queuing and weighted queuing. The algorithms provide chunk placement, request scheduling for the two models and weight assignment (for weighted queuing). We then find the average retrieval latency of the r files. For priority queuing, the aggregate request arrival rate is 0.28/sec with ratio class1/class2=1:22. We repeat the experiment for priority queuing using placement and scheduling decisions from algorithm JLOP, and take the average latency to get enough data points for both priority classes in Fig. 8, even though C_2 does not affect latency in priority queuing. From Fig. 8 we can see for weighted queuing, latency of class 2 increases as C_2 decreases; i.e., when class 2 becomes even less important. Also, the average latency of class 1 requests decreases as C_2 decreases. This shows the expected result that when class 2 becomes more important, more weight is allocated to class 2, and since C_2 is always smaller than C_1, class 1 gets more bandwidth. For priority queuing we can see even with a much lower arrival rate of class 1 as compared to weighted queuing model, class 2 requests rarely get a chance to be served due to the priority queuing policy. Thus, they experience extremely long latency as compared to class 1 requests, and even when they are compared to both classes in weighted queuing. Fig. 8 shows that weighted queuing provides much more fairness for class 2 requests than priority queuing.

Evaluation of the performance of our solution–Priority queuing: To demonstrate the effectiveness of algorithm JLOP for priority queuing, we vary file size in the experiments from 50MB to 250MB with an aggregate request arrival rate at 0.28, ratio of high priority/low priority=1/22, and erasure code (7,4). We choose $r = 1000$ files, with 200 files of each file size, i.e., 200 files of 50MB, 200 of 100MB, etc. We initiate retrieval requests of these files and plot the average latency for files of each size. We also showed our analytic latency upper bound

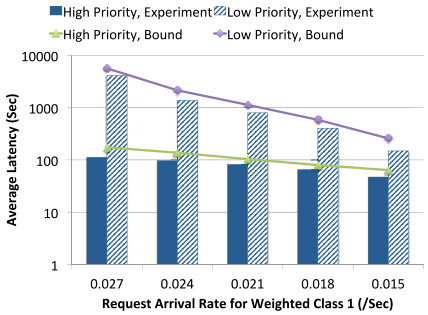


Fig. 10. Evaluation of different request arrival rates in priority queuing. Fixed $\lambda_2 = 0.14/sec$ and varying λ_1. As arrival rates of high priority class increase, latency of low priority requests shows logarithm growth.

in Figure 9. We see that the average latency increases almost linearly with file size for high priority class since the service time for each class increases linearly, and queuing delay depends only on high priority requests. However, for requests of low priority latency increases more than linearly with file sizes since the latency depends mainly on whether a request get a chance to be served or not, i.e., queuing delay dominates and the service time is much smaller as compared to queuing delay. We also observe that our analytic latency bound tightly follows actual average service latency for both classes.

Next, we fixed the arrival rate of low priority requests to 0.14/sec and varied the file request arrival rate of high priority class from $\lambda_1 = 0.027$ /sec to $\lambda_1 = 0.015$ /sec with file size $200MB$. Actual service delay and our analytic bound for each class is shown by a bar plot in Figure 10. Our bound provides a close estimate of service latency as shown in the figure. As arrival rates for high priority increases, latency of low priority class shows logarithmic growth, which means the probability that a low priority request gets served becomes dramatically less as the arrival rate of high priority requests increases, which leads to extremely long queuing delay for low priority requests. This shows extreme unfairness for class 2 requests.

Evaluation of the performance of our solution–Weighted queuing: For weighed queuing, we design a similar experiment as in the case of priority queuing to compare the results.

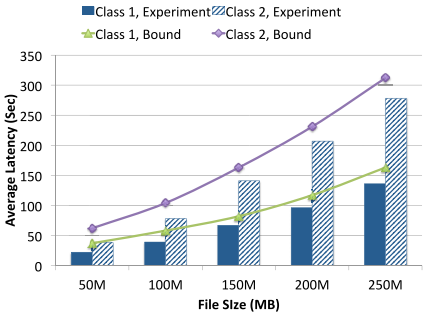


Fig. 11. Evaluation of different file sizes in weighted queuing. Latency increase shows more fairness for class 2 requests. Our analytic latency bound taking both network and queuing delay into account tightly follows actual service latency for both classes.

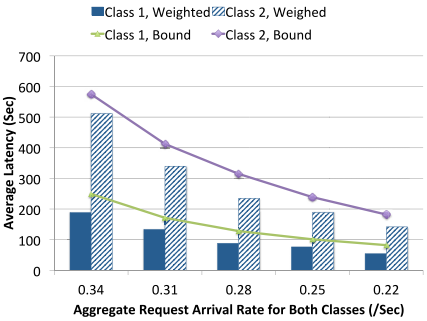


Fig. 12. Evaluation of different request arrival rates in weighted queuing. As the arrival rate increases, latency increase shows more fairness for class 2 requests compared to priority queuing.

First, we varied the file size from 50MB to 250MB with aggregate request arrival rate of 0.28/sec, and the ratio of class1/class2=1/1. We have 1000 heterogeneous files in total, out of which every 200 files are of the same file size. A (7,4) erasure code is applied, $C_1 = 1$, and $C_2 = 0.4$. We used the same workload combination as in the file size experiment for priority queuing. As shown in Fig. 11, we can see for both classes, latency increases as file size increases, but class 2 increases much faster than class 1. This is because the class 2 requests typically get a small portion of service bandwidth, thus increasing file size will increase the service time and thus the queuing time is a lot more than class 1 requests, which get more bandwidth. Also the analytic bound for both classes tightly follows the actual latency as shown in the figure as well. However, even though class 2 requests are experiencing longer latency than class 1 requests in this case, the latency is still within a proper range unlike that in priority queuing. Thus, we can see that weighted queuing provides more fairness to different classes.

Next we varied the aggregate arrival rate of both class 1 and class 2 requests from 0.34/sec to 0.22/sec as shown in Figure 12 with file size 200*MB*, while keeping the ratio of class1/class2=1. Actual service delay and our analytic bound for each class is shown in Figure 10, where the bound provides a close estimate of service latency as shown in the figure. We can see as the arrival rate for class 1 increases, latency of class 2 also increases much faster than that of class 1. This is because increasing the arrival rate for class 1 will give more bandwidth to class 1 in order to reduce their latency and thus further decreases the bandwidth to class 2. Also, increasing the workload could not be completely compensated by increasing the bandwidth, and thus latency of class 1 increases too.

VI. Conclusion

Relying on a novel probabilistic scheduling policy, this paper develops an analytical upper bound on average service delay of multi-tenant, erasure-coded storage with arbitrary number of files and any service time distribution using weighted queuing or priority queuing to provide differentiated services to different tenants. An optimized distributed storage system is then formalized using these queues. Even though only local optimality can be guaranteed due to the non-convex nature of the problems, the proposed algorithm significantly reduces the latency. Both our theoretical analysis and algorithm design are validated via a prototype in Tahoe, an open-source, distributed file system, in an open-source, cloud storage deployment that simulates three geographically distributed data centers through bandwidth reservations. Note that this paper focuses on two service classes only, and an extension to a general number of service classes is still an open research problem.

Appendix A
Theorem 1

Proof: Following the results in [21, Ch. 3] and [22, Ch. 7], we have that the mean queuing delay for the high priority class at server j is given as

$$\mathbb{E}\big[Q_{j1}\big] = \frac{\mathbb{E}[R]}{1-\rho_{j1}}$$

where $\mathbb{E}[R]$ is the residual service time, given as

$$\mathbb{E}[R] = \sum_{k=1}^{2} \frac{\Lambda_{jk}\Gamma_j^2}{2},$$

and $\Lambda = \sum_{k=1}^{2} \Lambda_{jk}$ be the aggregate request arrival rate of the two priority classes at server j. The mean queuing delay for the low priority class at server j is given as

$$\mathbb{E}\big[Q_{j2}\big] = \frac{\mathbb{E}[R]}{\big(1-\rho_{j1}\big)\big(1-\rho_{j1}-\rho_{j2}\big)}$$

The second moment of the queuing delay for the high priority class at server j is

$$\mathbb{E}\Big[Q_{j1}^2\Big] = 2\big(\mathbb{E}[Q_{j1}]\big)^2 + \frac{\rho_{j1}\mathbb{E}\big[R^2\big]}{1-\rho_{j1}},$$

where $\mathbb{E}[R] = \frac{\mathbb{E}[X_j^2]}{2\mathbb{E}[X_j]}$ and $\mathbb{E}[R^2] = \frac{\mathbb{E}[X_j^3]}{3\mathbb{E}[X_j]}$. The variance of the queuing delay for the high priority class at server j is

$$\begin{aligned}\mathrm{Var}\big[Q_{j1}\big] &= \mathbb{E}\Big[Q_{j1}^2\Big] - \big(\mathbb{E}\big[Q_{j1}\big]\big)^2 \\ &= \frac{\rho_{j1}}{\big(1-\rho_{j1}\big)^2}\Big(\rho_{j1}\big(\mathbb{E}[R]\big)^2 + \big(1-\rho_{j1}\big)\mathbb{E}[R^2]\Big) \qquad (21)\end{aligned}$$

The second moment of the queuing delay for the low priority class at server j is

$$\mathbb{E}\Big[Q_{j2}^2\Big] = 2\big(\mathbb{E}\big[Q_{j2}\big]\big)^2 + \frac{(\rho_{j1}+\rho_{j2})\mathbb{E}\big[R^2\big]}{\big(1-\rho_{j1}\big)^2\big(1-\rho_{j2}\big)},$$

and the variance of the queuing delay for the high priority class at server j is

$$\begin{aligned}\mathrm{Var}\big[Q_{j2}\big] &= \mathbb{E}\Big[Q_{j1}^2\Big] - \big(\mathbb{E}\big[Q_{j1}\big]\big)^2 \\ &= \frac{\rho_{j1}+\rho_{j2}}{\Big(1-\rho_{j1}^2\big(1-\rho_{j2}\big)^2\Big)}\Big(\big(1-\rho_{j2}\big)\mathbb{E}[R^2] \\ &\quad + \left[\big(\rho_{j1}+\rho_{j2}\big) + \frac{\rho_{j1}\big(1-\rho_{j2}\big)}{1-\rho_{j1}}\right](\mathbb{E}[R])^2\Big) \end{aligned} \quad (22)$$

This further gives

$$\begin{aligned}\mathrm{Var}\big[Q_{j2}\big] &= \frac{\big(\Lambda_{j1}+\Lambda_{j2}\big)\Gamma_{j,3}}{3\big(1-\rho_j 1\big)^2\big(1-\rho_{j2}\big)} + \left(\frac{\big(\Lambda_{j1}+\Lambda_{j2}\big)\Gamma_{j,2}}{2\big(1-\rho_{j1}\big)\big(1-\rho_{j2}\big)}\right)^2 \\ &\quad + \frac{\Lambda_{j1}\big(\Lambda_{j1}+\Lambda_{j2}\big)\Gamma_{j,2}^2}{2\big(1-\rho_{j1}\big)^3\big(1-\rho_{j2}\big)} \end{aligned} \quad (23)$$

These will further give the result as shown in Theorem 1 after simplification. ■

Appendix B
Theorem 2

Proof: From the weighted queuing model we know that each of them will be served at different service rate, thus service time for each weighted queue with weight w_{jk} would be $\mathbb{E}[X]/w_{jk}$, with probability P_{jk}, thus the expected service time would be $\sum_{k=1}^{n}\frac{P_{jk}\mathbb{E}[X]}{w_{jk}}$, variance of service time should be $\mathbb{E}[X_{jk}^2] - (\mathbb{E}[X_{jk}])^2]$, thus we have $\mathrm{Var}[X_{jk}]$ as the second item in $\mathbb{E}Z_j$. We also have

$$\mathbb{E}\big[Q_j\big] = \sum_{k=1}^{n} P_{jk}\mathbb{E}\big[Q_{jk}\big],$$

where $\mathbb{E}[Q_{jk}]$ is the expected waiting time for requests in the M/G/1 queue with weight w_{jk}, which is the same as what we derived for single priority class in priority queuing, with different service rates:

$$\mathbb{E}\big[Q_{jk}\big] = \frac{\Lambda_{jk}P_{jk}\mathbb{E}\big[X^2\big]/w_{jk}^2}{2-\Lambda_{jk}P_{jk}\mathbb{E}[X]/w_{jk}}$$

Then we can obtain:

$$\mathbb{E}\big[Q_j\big] = \sum_{k=1}^{n} P_{jk}\frac{\Lambda_{jk}P_{jk}\mathbb{E}\big[X^2\big]/w_{jk}^2}{2-\Lambda_{jk}P_{jk}\mathbb{E}[X]/w_{jk}}$$

After simplification, we obtain the third item in the statement of the lemma. For the variance, we apply the results we get for single priority class model with weighted service rates and take average with the distribution probability for each weighted queue to get:

$$\begin{aligned}Q_{jk} = \sum_{k=1}^{n} P_{jk}\Big(&\frac{\Lambda_{jk}P_{jk}\mathbb{E}\big[X^3\big]/w_{jk}^3}{3\big(1-\Lambda_{jk}P_{jk}\mathbb{E}[X]/w_{jk}\big)} \\ &+ \frac{\Lambda_{jk}^2P_{jk}\mathbb{E}\big[X^4\big]/w_{jk}^4}{4\Big(1-\Lambda_{jk}P_{jk}\mathbb{E}[X]/w_{jk}^2\Big)}\Big)\end{aligned} \quad (24)$$

A simplification will lead to the equation in Theorem 2. ■

Appendix C
Lemma 2

Proof: As shown in Equation (13), $\bar{T}_{ik}$ depends on $F(z, \pi_{ij})$ for weighted queuing. Since we have:

$$F\big(z, \Lambda_{jk}\big) = A_{j2} + \sqrt{A_{j2}^2 + B_{j2}}$$

We find the second order derivatives of A_{j2} with respect to Λ_{j2}.

$$\frac{\partial^2 A_{jk}}{d\Lambda_{j2}^2} = \frac{\Gamma_{j,2}}{\big(\rho_{j1}-1\big)\big(\rho_{j1}\mu_j+\Lambda_{j2}-\mu_j\big)^3} \quad (25)$$

We note that $\rho_{j1}-1<0$, and $(\rho_{j1}\mu_j+\Lambda_{j2}-\mu_j)^3$ is negative as long as $\rho_{j2}<1-\rho_{j1}$, since Λ_{j2} is a linear combination of π_{ij} for low priority class, i.e., A_{j2} is convex in π_{ij} as long as $\Lambda_{j2}<\frac{1-\rho_{j1}}{\mu_j}$. We find the second order derivatives of B_{j2} with respect to Λ_{j2}.

$$\begin{aligned}\frac{\partial^2 B_{j2}}{d\Lambda_{j2}^2} &= \frac{2\Gamma_{j,3}\mu_j\big(\Lambda_{j1}+\mu_j\big)}{3\big(1-\rho_{j1}\big)^2\big(\mu_j-\Lambda_{j2}\big)^3} \\ &\quad + \frac{\Gamma_{j,2}^2\mu_j^2\big(\Lambda_{j1}+\mu_j\big)\big(3\Lambda_{j1}+\mu_j+2\Lambda_{j2}\big)}{2\big(1-\rho_{j1}\big)^2\big(\mu_j-\Lambda_{j2}\big)^4} \\ &\quad + \frac{\Lambda_{j1}\Gamma_{j,2}^2\mu_j\big(\Lambda_{j1}+\mu_j\big)}{\big(1-\rho_{j1}\big)^3\big(\mu_j-\Lambda_{j2}\big)^3}\end{aligned} \quad (26)$$

Here, all the three terms are positive as long as $\Lambda_{jk}<\mu_j$. Thus, both A_{j2} and B_{j2} are convex in π_{j2}. Now we prove that F is convex in A_{j2} and B_{j2}:

$$\begin{aligned}\frac{\partial^2 F}{\partial\Lambda_{j2}^2} &= \frac{\partial^2 A_{j2}}{\partial\Lambda_{j2}^2} + \frac{A_{j2}\frac{\partial^2 A_{j2}}{\partial\Lambda_{j2}^2} + B_{j2}\frac{\partial^2 B_{j2}}{\partial\Lambda_{j2}^2}}{\Big(A_{j2}^2+Bj2^2\Big)^{1/2}} \\ &\quad + \frac{\Big(A_{j2}\frac{\partial B_{j2}}{\partial\Lambda_{j2}} + B_{j2}\frac{\partial A_{j2}}{\partial\Lambda_{j2}}\Big)^2}{\Big(A_{j2}^2+B_{j2}^2\Big)^{3/2}}\end{aligned} \quad (27)$$

As we have proved that $\frac{\partial^2 A_{j2}}{d\Lambda_{j2}^2}>0$ and $\frac{\partial^2 A_{j2}}{d\Lambda_{j2}^2}>0$ then it easy to see from equation (27) that $\frac{\partial^2 F}{\partial\Lambda_{j2}^2}>0$. Since F is increasing and convex in A_{j2} and B_{j2}, and A_{j2} and B_{j2} are both convex in Λ_{j2}, we conclude that their composition $F(z, \Lambda_{j2})$ is also convex in Λ_{j2}. This completes the proof. ■

APPENDIX D
LEMMA 3

Proof: As shown in Equation (13), $\bar{T}_1, \bar{T}_2$ depends on $F(z, \pi_{ij}, w_{jk})$ for weighted queuing. Since we have:

$$F\big(z, \Lambda_{jk}, w_{jk}\big) = A_j + \sqrt{A_{jk}^2 + B_{jk}}$$

We find the second order derivatives of A_{jk} with respect to w_{jk}.

$$\frac{\partial^2 A_{jk}}{dw_{jk}^2} = \sum_{k=1}^{2} \frac{\Lambda_{jk} P_{jk}^2 \mu_j \Gamma_{j,2} \left(\left(\frac{3\mu_j w_{jk}}{2} - \Lambda_{jk} P_{jk} \right)^2 + \frac{3\mu_j^2 w_{jk}^2}{4} \right)}{w_{jk}^3 \big(\mu_j w_{jk} - \Lambda_{jk} P_{jk}\big)^3} \tag{28}$$

where the numerator is positive since it's the sum of two squares $(\frac{3\mu_j w_{jk}}{2} - \Lambda_{jk} P_{jk})^2$ and $\frac{3\mu_j^2 w_{jk}^2}{4}$. Denominator is positive as long as $w_{jk} > \frac{\Lambda_{jk} P_{jk}}{\mu_j}$.

The second order derivatives of B_{jk} with respect to w_{jk} is given as

$$\frac{\partial^2 B_{jk}}{dw_{jk}^2} = \frac{\Lambda_{jk}^2 P_{jk}^3 \Gamma_{j,2}^2}{2w_{jk}^2 \left(w_{jk} - \frac{\Lambda_{jk} P_{jk}}{\mu_j} \right)^2} \times \left(\frac{3}{w_{jk}^2} + \frac{4}{w_{jk} \left(w_{jk} - \frac{\Lambda_{jk} P_{jk}}{\mu_j} \right)} + \frac{3}{\left(w_{jk} - \frac{\Lambda_{jk} P_{jk}}{\mu_j} \right)^2} \right), \tag{29}$$

which is positive as long as $w_{jk} > \frac{\Lambda_{jk} P_{jk}}{\mu_j}$. Thus, both A_{jk} and B_{jk} are convex in w_{jk}. As proved in the proof of Lemma 2, F is convex in A_{jk} and B_{jk}. Since F is increasing and convex in A_{jk} and B_{jk}, and A_{jk} and B_{jk} are both convex in w_{jk}, we conclude that their composition $F(z, \Lambda_{jk}, w_{jk})$ is also convex in w_{jk}. This completes the proof. ■

REFERENCES

[1] Y. Xiang, T. Lan, V. Aggarwal, and Y.-F. R. Chen, "Multi-tenant latency optimization in erasure-coded storage with differentiated services," in *Proc. ICDCS*, Columbus, OH, USA, Jun./Jul. 2015, pp. 790–791.

[2] E. Schurman and J. Brutlag, "The user and business impact of server delays, additional bytes, and HTTP chunking in Web search," in *Proc. O'Reilly Velocity Web Perform. Oper. Conf.*, San Jose, CA, USA, Jun. 2009.

[3] G. Ananthanarayanan, A. Ghodsi, S. Shenker, and I. Stoica, "Effective straggler mitigation: Attack of the clones," in *Proc. USENIX Conf. Netw. Syst. Design Implementation*, Lombard, IL, USA, 2013, pp. 185–198.

[4] H. Xu and W. C. Lau, "Optimization for speculative execution in big data processing clusters," *IEEE Trans. Parallel Distrib. Syst.*, vol. 28, no. 2, pp. 530–545, Feb. 2017.

[5] G. Ananthanarayanan *et al.*, "Reining in the outliers in map-reduce clusters using Mantri," in *Proc. USENIX Conf. Oper. Syst. Design Implementation*, Vancouver, BC, Canada, 2010, pp. 265–278.

[6] D. Wang, G. Joshi, and G. Wornell, "Using straggler replication to reduce latency in large-scale parallel computing," *ACM SIGMETRICS Perform. Eval. Rev.*, vol. 43, no. 3, pp. 7–11, 2015.

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

[8] L. Huang, S. Pawar, H. Zhang, and K. Ramchandran, "Codes can reduce queueing delay in data centers," in *Proc. IEEE Int. Symp. Inf. Theory* , Cambridge, MA, USA, 2012, pp. 2766–2770.

[9] N. Shah, K. Lee, and K. Ramachandran, "The MDS queue: Analyzing latency performance of codes and redundant requests," in *Proc. IEEE Int. Symp. Inf. Theory*, Honolulu, HI, USA, Jul. 2014, pp. 861–865.

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

[11] R. Pedarsani, J. Walrand, and Y. Zhong, "Robust scheduling in a flexible fork-join network," in *Proc. 53rd IEEE Conf. Decis. Control*, Los Angeles, CA, USA, 2014, pp. 3669–3676.

[12] E. Özkan and A. R. Ward, "On the control of fork-join networks," *arXiv:1505.04470v2*, Jun. 2016.

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

[14] S. Chen *et al.*, "When queueing meets coding: Optimal-latency data retrieving scheme in storage clouds," in *Proc. IEEE Infocom*, Toronto, ON, Canada, Apr./May 2014, pp. 1042–1050.

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

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

[17] Y. Xiang, V. Aggarwal, Y.-F. R. Chen, and T. Lan, "Differentiated latency in data center networks with erasure coded files through traffic engineering," *IEEE Trans. Cloud Comput.*, to be published.

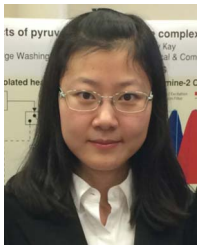
[18] V. Aggarwal and T. Lan, "Tail index for a distributed storage system with Pareto file size distribution," *arXiv:1607.06044*, Jul. 2016.

[19] F. J. MacWilliams and N. J. Sloane, *The Theory of Error-Correcting Codes*. Amsterdam, The Netherlands: North-Holland, 1977.

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

[21] L. Kleinrock, *Queueing Systems Volume 2: Computer Applications*. New York, NY, USA: Wiley, 1976.

[22] J. Sztrik, *Basic Queueing Theory*. Saarbrücken, Germany: GlobeEdit and OmniScriptum GmbH & Co, KG, 2016.

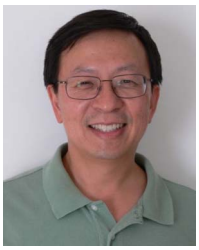


Yu Xiang received the B.A.Sc. degree from the Harbin Institute of Technology in 2010, and the Ph.D. degree from George Washington University in 2015, both in electrical engineering. She is currently a Senior Inventive Scientist with AT&T Labs-Research. Her current research interests are in cloud resource optimization, distributed storage systems, and cloud storage charge-back.

Tian Lan (S'03–M'10) received the B.A.Sc. degree from Tsinghua University, China, in 2003, the M.A.Sc. degree from the University of Toronto, Canada, in 2005, and the Ph.D. degree from Princeton University in 2010. He is currently an Associate Professor of Electrical and Computer Engineering with George Washington University. His research interests include cloud resource optimization, mobile networking, storage systems, and cyber security. He was a recipient of the 2008 IEEE Signal Processing Society Best Paper Award, the 2009 IEEE GLOBECOM Best Paper Award, and the 2012 INFOCOM Best Paper Award.

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

He was a Senior Member of Technical Staff Research with AT&T Labs-Research, NJ, USA, and an Adjunct Assistant Professor with Columbia University, NY, USA. He is currently an Assistant Professor with Purdue University, West Lafayette, IN, USA. His research interests are in applications of statistical, algebraic, and optimization techniques to distributed storage systems, machine learning, and wireless systems. He was a recipient of the Princeton University's Porter Ogden Jacobus Honorific Fellowship in 2009, the AT&T Key Contributor Award in 2013, the AT&T Vice President Excellence Award in 2012, and the AT&T Senior Vice President Excellence Award in 2014. He is serving on the Editorial Board of the IEEE TRANSACTIONS ON COMMUNICATIONS and the IEEE TRANSACTIONS ON GREEN COMMUNICATIONS AND NETWORKING.



Yih-Farn (Robin) Chen received the B.S. degree in electrical engineering from National Taiwan University, the M.S. degree in computer science from the University of Wisconsin, Madison, and the Ph.D. degree in computer science from the University of California at Berkeley. He is a Director of Inventive Science, leading the Cloud Platform Software Research Department with AT&T Labs-Research. His current research interests include cloud computing, software-defined storage, mobile computing, distributed systems, World Wide Web, and IPTV. He is an ACM Distinguished Scientist and a member of the International World Wide Web Conferences Steering Committee (IW3C2). He also serves on the editorial board of the *IEEE Internet Computing*.