12-2016

Taming tail latency for erasure-coded, distributed storage systems

Jingxian Fan
Purdue University

Graduate School Form
30 Updated 12/26/2015

PURDUE UNIVERSITY
GRADUATE SCHOOL
Thesis/Dissertation Acceptance

This is to certify that the thesis/dissertation prepared

By JINGXIAN FAN

Entitled
TAMING TAIL LATENCY FOR ERASURE-CODED, DISTRIBUTED STORAGE SYSTEMS

For the degree of Master of Science in Industrial Engineering

Is approved by the final examining committee:

VANEET AGGARWAL
Chair

CHRISTOPHER J. QUINN

GESUALDO SCUTARI

To the best of my knowledge and as understood by the student in the Thesis/Dissertation Agreement, Publication Delay, and Certification Disclaimer (Graduate School Form 32), this thesis/dissertation adheres to the provisions of Purdue University's "Policy of Integrity in Research" and the use of copyright material.

Approved by Major Professor(s): VANEET AGGARWAL

Approved by: ABHIJIT DESHMUKH 11/30/2016
Head of the Departmental Graduate Program Date

TAMING TAIL LATENCY FOR ERASURE-CODED, DISTRIBUTED STORAGE SYSTEMS

A Thesis

Submitted to the Faculty

of

Purdue University

by

Jingxian Fan

In Partial Fulfillment of the

Requirements for the Degree

of

Master of Science in Industrial Engineering

December 2016

Purdue University

West Lafayette, Indiana

For everyone that have supported and guided me during my graduate study life.

ACKNOWLEDGMENTS

This research on "Taming Tail Latency for Erasure-Coded, Distributed Storage Systems" has been given to me as part of the curriculum in my Master of Science Degree in Industrial Engineering.

I have explored and worked hard on this topic for months. During this time, I have been confronted with difficulties on theories and stepped into wrong directions several times. But finally with many helps I managed to work this out.

First I would like to thank my major professor Vaneet Aggarwal, without whom this research would never be started or completed. He gave me the chance to understand the model and figure out the problems. Not only he provided me with his insights on this research, but also he has been very supportive and encouraged me whenever a problem came out.

I would also like to thank my senior Wenqi Wang from our lab. Even though we are focused on different topics, he mentored me with usage of software and offered some technical helps during the implementation of simulation.

Last but not the least, the CLAN lab has been like a family to me all the time. I am grateful to have them in my graduate study life.

TABLE OF CONTENTS

Page

LIST OF TABLES

LIST OF FIGURES

SYMBOLS

r	Number of files in system by $i = 1, 2, ..., r$
m	Number of storage nodes
(n, k)	Erasure code for storing files
λ_i	Arrival rate of file i
π_{ij}	Probability of retrieving chunk from node j of file i
L_i	Latency of retrieving file i
x	Some constant of latency
Q_j	Sojourn Time of node j
X_j	Service Time of node j
μ_j	Mean sojourn time of node j
Λ_j	Arrival rate on node j
α_j	parameter of shifted exponential distribution of server j
β_j	parameter of shifted exponential distribution of server j
Γ_j	Second moment of sojourn time of node j
$\hat{\Gamma}_j$	Third moment of sojourn time of node j
$M_j(t)$	Moment Generating Function for the service time of 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
ω_i	weight of file i

ABBREVIATIONS

MDS	Maximum-Distance Separable
AMM	Alternating Minimization Method
WLTP	Weighted Latency Tail Probability Optimization
PEAP	Projected, Equal Access-Probability
BNW	Balanced Node Workload

ABSTRACT

Fan, Jingxian MS, Purdue University, December 2016. Taming Tail Latency For Erasure-Coded, Distributed Storage Systems . Major Professor: Vaneet Aggarwal.

Nowadays, in distributed storage systems, long tails of responsible time are of particular concern. Modern large companies like Bing, Facebook and Amazon Web Service show that 99.9th percentile response times being orders of magnitude worse than the mean. With the advantages of maintaining high data reliability and ensuring enough space efficiency, erasure code has become a popular storage method in distributed storage systems. However, due to the lack of mathematical models for analyzing erasure-coded based distributed storage systems, taming tail latency is still an open problem.

In this research, we quantify tail latency in such systems by deriving a closed upper bounds on tail latency for general service time distribution and heterogeneous files. Later we specified service time to shifted exponentially distributed. Based on this model, we developed an optimization problem to minimize weighted tail latency probability of deriving all files. We propose an alternating minimization algorithm for this problem. Our simulation results have shown significant reduction on tail latency of erasure-coded distributed storage systems with realistic environment workload.

1. INTRODUCTION

1.1 MOTIVATION: TRANSITION FROM FULL DATA REPLICATION TO ERASURE CODING

As the era of data explosion comes, emerging applications like big data analytics and cloud computing demand distributed storage systems to offer multiple petabyte level storage, and this need of storage is exponentially increasing. Traditionally, to avoid the data loss caused by servers breaking down, system failures or other unpredictable cause, distributed storage systems apply full data replication to achieve high reliability. Under this storage method, if a file is replicated n times, it can be recovered by accessing any of the n replicas. However, full data replication consumes times of more storage which is not favored by modern data driven companies.

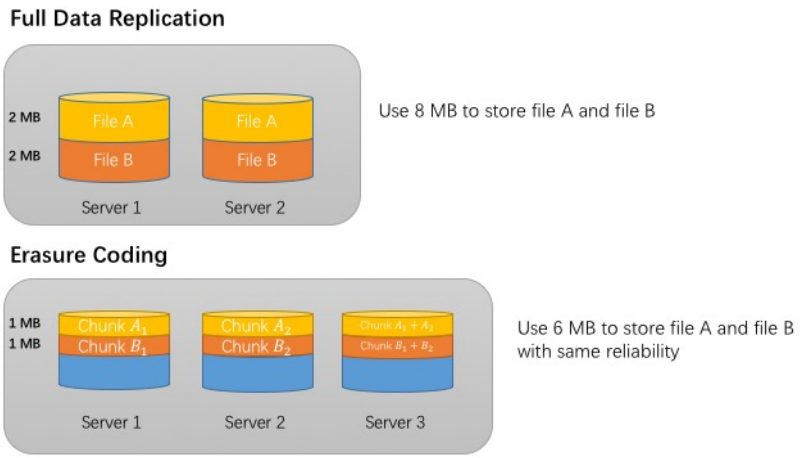

Figure 1.1. Full Data Replication Storage vs. Erasure Coding Storage

By comparison, erasure coding succeeds in reducing the cost of storage while maintaining high reliability. This emerging method has been widely studied for distributed storage systems and used by modern online applications such as Facebook and Google because of its advantages of space-optimal data redundancy for data protection. It has been showed that using erasure coding can efficiently reduce the cost of storage over 50% as a result of smaller storage space and data center footprint. For an erasure-coded system uses an (n, k) code, each file is encoded into n equal-size data chunks, allowing reconstruction from any subsets of $k < n$ chunks. Whenever the file is requested, the system needs to fetch k distinct chunks from different servers, which ensuring high reliability because even some of the servers breaks down, the file can still be recover from other k servers. And the storage is now n data chunks instead of n replication of files. Figure 1.1 gives the comparison of full data replication and erasure coding with an example of storing two files, file A and file B, both of 2 MB. It shows file A and B takes 8 MB to securely store with full data replication while it only takes 6 MB to store file A and B with the same reliability using a (3, 2) erasure coding.

However, there exists a key tradeoff for erasure coding, the delay of retrieving files. As data chunks are not stored in every server and the bandwidth between different servers is limited, a significant delay in data access becomes critical when retrieving the file, which can be perceived as poor quality of service. This long tail latency is of particular concern to modern web applications. Google and Amazon have published that every 500 ms extra delay causes a 1.2% user loss. Meanwhile, quantifying the tail latency in erasure-coded data storage system remains an open problem. Despite recent research effort working on mean service latency but less focus on tail latency, an analytical framework to quantify tail latency in distributed storage systems employing erasure codes is still a problem to explore.

Therefore, this research focuses on the tail latency of erasure-coded, distributed storage systems. Establishing a realistic system model, finding a valid upper bound for this tail latency in closed form and optimizing all related parameters in this system

to minimize tail latency for better industrial applications are the main goals of this research.

1.1.1 RESEARCH PROBLEM

Based on the previous motivation, the research problem of this thesis is defined as below:
Build an analytical framework to quantify tail latency in erasure-coded, distributed storage systems and optimize the tail latency.

1.1.2 RESEARCH QUESTIONS

To solve this problem, these research questions are defined:

Research Question 1: What is the mathematical system model for erasure-coded, distributed storage system?

Research Question 2: What is the weighted upper bound of tail latency for retrieving files from this system?

Research Question 3: After we have the upper bound, how to optimize it and what is the optimal result?

1.2 PREVIOUS RELATED WORK

Recent research effort provides bounds on mean service latency, however less is known on tail latency. To provide upper bounds on mean service latency of homogeneous files, there are two major analysis from prior work: *Fork-join queue analysis* and *Queuing-Theoretic Analysis*.

Fork-join queue analysis: The fork-join queue, [1] is one of the queuing models for erasure-coded storage. In [2] the authors provided a heuristic transmission scheme based on this model, in which a file request is forked to every storage node with

the file chunks. The file request exits the system when any first k chunks are processed. Under this way, the model adjusts coding parameters dynamically and thus improve latency performance. In [4], the authors applied this (n, k) fork-join queue to model the latency performance of erasure-coded storage, provided a closed-form upper bound of mean service latency in the case of systems with only homogeneous files and exponentially distributed service time. But the approach has problems being applied to heterogeneous file systems because each file has a separate folk-join queue and the queues of different files are highly dependent due to shared storage nodes and joint request scheduling. The authors of [3] proposed a self-adaptive policy that under dynamic workload status in erasure-coded storage systems, adjust chunk size and number of redundancy requests dynamically to minimize queuing latency in fork-join queues. Another work [5] used this fork-join queue to optimize threads allocation to each file request. But the proposed greedy/shared scheme could waste system resources because in fork-join queue there will always exist some threads with unfinished downloads as a result of redundant assignment.

In addition, in [6] the authors proposed a model that analyzed the (n, k) Fork-join queue model with heterogeneous files. From distinct classes under different scheduling policies, like preemptive, First-Come-First-Serve and non-preemptive priority scheduling policies, they derived lower and upper bounds for the average latency on jobs based on the analysis of mean and second moment of waiting time. But individual file request must be served by all n nodes or a set of pre-specified nodes under a folk-join queue, falling short to deal with dynamic load-balancing of heterogeneous files.

Queuing-Theoretic Analysis: , the authors in [7] proved asymptotic results for symmetric large scale systems that can be applied to provide a computable approximation for expected latency under an assumption of exponential service time distribution with homogeneous files. But the assumption that chunk placement is fixed and so is coding policy for all file requests is not true in reality. The authors in [8] present a block-one-scheduling policy only allowing the one request at the head of

the buffer to move forward. They provided an upper bound on the mean latency of storage system using queuing-theoretic analysis for erasure code scheme with fixed k. Later this approach is extended in [9] to general (n, k) erasure codes, still for homogeneous files. To provide numerical upper bounds on the mean latency, they proposed a family of *MDS-Reservation(t)* scheduling policy that block all except the first t of file requests. When t increases, the bound goes tighter but the number of states in the queueing-theoretic analysis increases exponentially.

1.3 ORGANIZATION OF THIS THESIS

In this thesis, chapter 1 introduce the motivation of this research and gives the layout of research problems. Chapter 1 also includes the related work on similar topics.

Chapter 2 describes the mathematical system model of erasure coded, distributed storage system. Chapter 3 focuses on a closed-formed upper bound of tail latency and formulates the optimization under the assumption that the service time is arbitrary distributed, which is the first stage and result of our analysis. Chapter 4 focuses on a tighter upper bound of tail latency and formulates the optimization under the assumption that the service time is shifted exponentially distributed, which explores a better optimization performance. In both chapter 3 and chapter 4, the algorithms for optimization and the simulation results is presented. In the last chapter, there are conclusions for this thesis and future works to do.

2. SYSTEM MODEL

2.1 DATA STORAGE AND ERASURE CODING

In a distributed storage system, there are heterogeneous servers to store data and process requests. Files are distributed among these servers and retrieved from them whenever needed. In this research we consider a data center of m heterogeneous storage servers, denoted by $M = 1, 2, ..., m$, also called storage nodes. This data center is distributively stored with a set of r files, indexed by $i = 1, 2, ...r$. Each file i is partitioned into k_i fixed-size data chunks and then it is encoded using an (n_i, k_i) MDS erasure code so it will generate n_i distinct chunks of the same size for file i. Then these encoded chunks are assigned to n_i distinct storage nodes, denoted by a set S_i of storage nodes, which should satisfy $S_i \subseteq M$ and $n_i = |S_i|$. With the use of (n_i, k_i) MDS erasure code, it enables the file to be rebuilt from any subset of k_i-out-of-n_i chunks. At the mean time, it introduces a redundancy factor of n_i/k_i. Thus, upon the arrival of each file request, k_i distinct data chunks are selected by a scheduler and retrieved to rebuild the desired file.

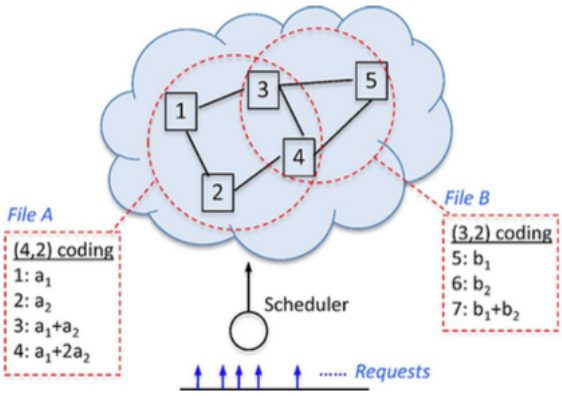

Figure 2.1. Erasure-coded storage of 2 files

Figure 2.1 is a erasure-coded storage data center of 2 files, A and B, partitioned into two blocks and encoded using (4, 2) and (3, 2) erasure code scheme respectively. Then encoded chunks are spread over all five storage nodes. File request for A and B should be processed by 2 different nodes with desired chunks. Node 3 and Node 4 are shared thus can process requests of both files.

2.2 PROBABILISTIC SCHEDULING

Prior work on erasure-coded storage systems focuses on mean latency with two main approaches, queuing-theoretic analysis and fork-join queue analysis. However when analyzing tail latency, both approaches appear weak to quantify because the states of the corresponding queuing model must encapsulate not only a snapshot of the current system with chunk placement and queue requests, but also the past history of how chunk requests have been processed by each storage nodes. As practical storage systems are required to handle a huge number of files and nodes, this can easily lead

to a state-explosion problem. If a simple scheduling policy that accesses available chunks with equal probability is applied, it will apparently lead to high tail latency resulted by hot storage nodes with worst performance. Meanwhile, a policy that load-balances the number of requests processed by each server does not necessarily optimize tail latency of all files, as files that employ different erasure codes causing different impact on service latency.

Since jointly request scheduling rule and the dependency of straggling fragment on popular storage nodes make tail latency even harder to quantify with current analysis, we use the Probabilistic Scheduling from [10] in this research. This probabilistic scheduling policy: 1) dispatches each batch of chunk requests corresponding to the same file request to a set of appropriate nodes with predetermined probabilities. This set of nodes is denoted by A_i of servers for file i and the predetermined probability is denoted by $P(A_i)$ for set A_i and file i; 2) each node buffers requests in a local queue and processes in order. The authors of [10] 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

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

The file request is completed if all its chunk requests have been processed by every node.

This probabilistic scheduling policy is used to provide an upper bound on mean service time when first proposed but in this research we extend this policy and propose an analytical model on tail latency allowing the optimization on tail latency.

2.3 QUEUING MODEL

Based on the distributed storage model system we now represent a queuing model of requests and processes in the system. We assume that the arrival of client requests for each file i follows 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 and file-size distribution. Under erasure codes, each file i can be retrieved from any k_i distinct nodes that store the file chunks. We model this by treating each file request as a batch of k_i chunk requests, so that a file request is served when all k_i chunk requests in the batch are processed by distinct storage nodes. All requests are buffered in a common queue of assumed infinite capacity.

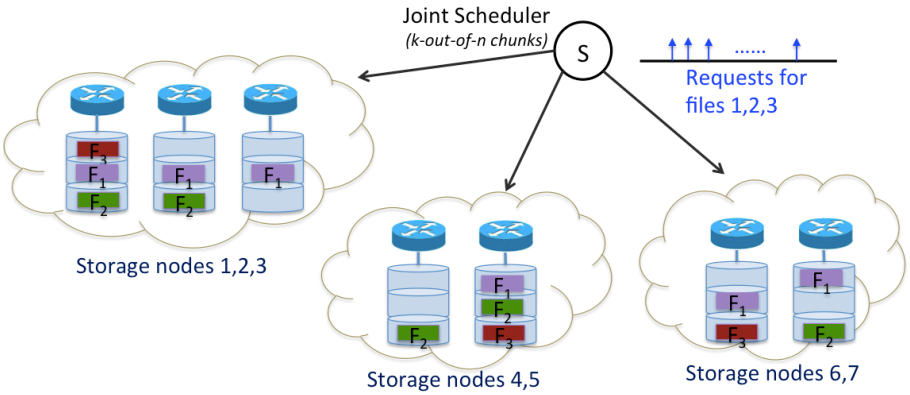

Figure 2.2. An illustration of a distributed storage system equipped with 7 nodes and storing 3 files using different erasure codes

Figure 2.2 is a distributed file system with 7 nodes. 3 files are stored in this system using $(6, 4)$, $(5, 3)$, and $(3, 2)$ MDS codes respectively. All file requests arriving are jointly scheduled to access k_i-out-of-n_i distinct chunks.

3. UNDER GENERAL SERVICE DISTRIBUTION: BOUNDS ON TAIL LATENCY, OPTIMIZATION AND SIMULATION RESULTS

3.1 UPPER 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, called sojourn time. Under probabilistic scheduling, the service time of a file-i, denoted by L_i request is determined by the maximum chunk service time at a randomly selected set A_i of storage nodes. 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 research 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. We consider an upper bound of tail latency on file i as

$$\begin{aligned}
\Pr(L_i \geq x) &\leq \Pr(L_{UB,i} \geq x) \\
&= \Pr_{A_i,Q_j}(\max_{j\in A_i} Q_j \geq x) \\
&= \Pr_{A_i,Q_j}(Q_j \geq x \text{ for some } j \in A_i) \\
&= \mathbb{E}_{A_i,Q_j}[\max_{j\in A_i} 1_{(Q_j\geq x)}] \\
&\leq \mathbb{E}_{A_i,Q_j} \sum_{j\in A_i} [1_{(Q_j\geq x)}] \\
&= \mathbb{E}_{A_i} \sum_{j\in A_i} [\Pr(Q_j \geq x)] \\
&= \sum_j \pi_{ij} [\Pr(Q_j \geq x)] \qquad (3.1)
\end{aligned}$$

Using Markov Lemma, $\Pr(Q_j \geq x)$ is bounded by $\frac{\mathbb{E}[Q_j^k]}{x^k}$ for any $k \geq 0$:

$$\Pr(Q_j \geq x) \leq \frac{\mathbb{E}[Q_j^k]}{x^k}$$

With these results, we can give a bound on file i from combinations of these moments with different k's.

$$\Pr(L_i \geq x) \leq \left(1 - \sum_{k=1}^{N} c_{i,k}\right) + \sum_{k=1}^{N} c_{i,k} \frac{\sum_j \pi_{ij} \mathbb{E}[Q_j^k]}{x^k} \tag{3.2}$$

for any $N > 0$, $c_{i,k} \geq 0$ for $k = 1, 2, \cdots, N$ with $\sum_{k=1}^{N} c_{i,k} \leq 1$. Further moments of Q_j are given by Pollaczek-Khinchine formula which gives the Laplace-Stieltjes transform of the waiting time in terms of that for the service time as

$$\widetilde{Q}_j(s) = \frac{(1-\rho_j)\widetilde{X_j}(s)s}{\Lambda_j \widetilde{X_j}(s) + s - \Lambda_j} \tag{3.3}$$

where $X_j(s)$ is Laplace-Stieltjes transform of the service time (with $\mathbb{E}[X_j] = \frac{1}{\mu_j}, \mathbb{E}[X_j^2] = \Gamma_j^2, \mathbb{E}[X_j^3] = \hat{\Gamma}_j^3$ and $\rho_j = \Lambda_j/\mu_j$), and $\rho_j = \Lambda_j/\mu_j$ is the request intensity at node j. Further,

$$\mathbb{E}[Q_j] = \frac{1}{\mu_j} + \frac{\Lambda_j \Gamma_j^2}{2(1-\rho_j)} \tag{3.4}$$

$$Var[Q_j] = {\sigma_j}^2 + \frac{\Lambda_j \hat{\Gamma}_j^3}{3(1-\rho_j)} + \frac{{\Lambda_j}^2 {\Gamma_j}^4}{4(1-\rho_j)^2} \tag{3.5}$$

$$\mathbb{E}[{Q_j}^2] = {\sigma_j}^2 + \frac{\Lambda_j \hat{\Gamma}_j^3}{3(1-\rho_j)} + \frac{{\Lambda_j}^2 {\Gamma_j}^4}{2(1-\rho_j)^2} + \frac{1}{{\mu_j}^2} + \frac{\Lambda_j \Gamma_j^2}{\mu_j(1-\rho_j)} \tag{3.6}$$

3.2 FORMULATION OF OPTIMIZED STORAGE SYSTEM

Let ω_i be the weight of file i, so ω_i is the fraction of file i requests, and upper bound of probability for average latency of all files is given by $\sum_i \omega_i \Pr(L_i \geq x)$. By adjusting weights ω_i, the proposed optimization allows us to explore a tail-latency tradeoff between different files and to offer elastic Service Level Agreements (SLA) to users with different tail latency preference. For instance, a large weight ω_i can be assigned to a video streaming application requiring quick responses, while a small ω_i might be appropriate for online data backup that is latency insensitive.

Thus, the problem of finding the upper bound for average latency over all files is now becoming a joint minimization problem:

$$\min \quad \sum_i \omega_i \left((1 - \sum_{k=1}^{N} c_{i,k}) + \sum_{k=1}^{N} c_{i,k} \frac{1}{x^k} \sum_j \pi_{ij} \mathbb{E}(Q_j^k) \right) \tag{3.7}$$

$$s.t. \quad \sum_{k=1}^{N} c_{i,k} = 1 \tag{3.8}$$

$$c_{i,k} \leq 0 \tag{3.9}$$

$$\sum_j \pi_{i,j} = K_i \tag{3.10}$$

$$\pi_{i,j} \in [0,1] \tag{3.11}$$

$$var. \quad \pi_{i,j}, c_{i,k} \tag{3.12}$$

Denote $U_{ik} = \frac{1}{x^k} \sum_j \pi_{ij} \mathbb{E}(Q_j^k)$, below follows the proof that U_{i1}, U_{i2} are both separately convex in $\pi_{i,j}$:

The following function, in which$X_j = (X_{1j}, X_{2J})$, X_{1j}, X_{2j} are functions of Λ_j defined as followed, is convex in Λ_j:

$$F(\Lambda_j) = \frac{\Lambda_j}{\hat{\lambda}}[X_j] \tag{3.13}$$

$$X_1 j = \frac{1}{\mu_j} + \frac{\Lambda_j \Gamma_j^2}{2(1-\rho_j)} \tag{3.14}$$

$$X_2 j = {\sigma_j}^2 + \frac{\Lambda_j \hat{\Gamma}_j^3}{3(1-\rho_j)} + \frac{{\Lambda_j}^2 {\Gamma_j}^4}{2(1-\rho_j)^2} + \frac{1}{{\mu_j}^2} + \frac{\Lambda_j \Gamma_j^2}{\mu_j (1-\rho_j)} \tag{3.15}$$

In order to prove that $F(\Lambda_j) = \frac{\Lambda_j}{\hat{\lambda}}[X_j]$ is convex in Λ_j,we already have the conclusion that we only need $\frac{\partial^2 X_j}{\partial \Lambda_j^2}$ to be positive.

As $X_{2j} = Var(Q_j) + X_{1j}^2$,according to the previous paper, [10]

$$\frac{\partial^2 X_{1j}}{\partial \Lambda_j^2} = \frac{\mu_j^2 \Gamma_j^2}{(\mu_j - \Gamma_j)^3} \geq 0 \tag{3.16}$$

$$\frac{\partial^2 X_{2j}}{\partial \Lambda_j^2} = \frac{\partial^2 Var(Q_j)}{\partial \Lambda_j^2} + \frac{\partial^2 (X_{1j}^2)}{\partial \Lambda_j^2} \geq 0 \tag{3.17}$$

So both U_1, U_2 are separately convex in Λ_j. As $\Lambda_j = \sum_i \lambda_i \pi_{i,j}$, so U_{i1}, U_{i2} are also separately convex in $\pi_{i,j}$.

Here we minimize the upper bound of average file tail latency probability over $\pi_{i,f}$ and $c_{i,k}$. Since we already have the convergence proof of U_{i1}, U_{i2}, we may find the minimization when $k = 2$.Then the joint minimization problem above is a convex minimization problem where the decision variables vector is split into two blocks. To solve this problem, we introduce the method of Alternating Minimization for Convex Programming for our case.

3.3 ALTERNATING MINIMIZATION METHOD

To solve the convex problem and present the algorithm for our case, we first introduce the Alternating Minimization Method as below:
For the following minimization problem:

$$min_{y\in\mathbb{R}^{n_1},z\in\mathbb{R}^{n_2}} H(\vec{y},\vec{z}) \equiv f(\vec{y},\vec{z}) + g_1(\vec{y}) + g_2(\vec{z}), \qquad (1.1)$$

The steps of Alternating-Minimization method to find the minimization. Below Table 3.1 shows the detailed steps of Alternating-Minimization method.

Table 3.1.
The Steps of Alternating Minimization Method

Step 1: Initialization

$\vec{y}_0 \in$ dom g_1,$\vec{z}_0 \in$ dom g_2 such that $\vec{z}_0 \in \text{argmin}_{\vec{z}\in\mathbb{R}^{n_2}} f(\vec{y}_0, \vec{z}) + g_2(\vec{z})$.

Step 2: General Step (k=0,1,...)

$$\vec{y}_{k+1} \in \underset{\vec{y}\in\mathbb{R}^{n_1}}{\text{argmin}}\, f(\vec{y}, \vec{z}_k) + g_1(\vec{y}),$$

$$\vec{z}_{k+1} \in \underset{\vec{z}\in\mathbb{R}^{n_2}}{\text{argmin}}\, f(\vec{y}_{k+1}, \vec{z}) + g_2(\vec{z})$$

Step 3: Decide the optimal set

The k-th iterate will be denoted by $\vec{x}_k = (\vec{y}_k.\vec{z}_k)$, and we also consider the sequence in between given by

$$\vec{x}_{k+\frac{1}{2}} = (\vec{y}_{k+1}.\vec{z}_k)$$

Since the generated sequence is monotone and satisfies:

$$H(\vec{x}_0) \geq H(\vec{x}_{\frac{1}{2}}) \geq H(\vec{x}_1) \geq H(\vec{x}_{\frac{3}{2}}) \geq \dots$$

When $H(\vec{x}_{k+\frac{1}{2}}) - H(\vec{x}_k) \to 0$,decide this $\vec{x}_k$ is the optimal arg.

Then we provide the proof that the Alternating Minimization Method can be applied to our minimization problem. To apply the Alt-Min Method on our minimization problem, first translate the constraints of the problem into the the function. Create function:

$$I(x) = \begin{cases} 0 & \text{if } x \geq 0 \\ \infty & \text{if } x < 0 \end{cases}$$

Change the problem into

$$\begin{aligned}\min_{c_{i,k},\pi_{ij}} \quad & \sum_i \omega_i \left((1 - \sum_{k=1}^{N} c_{i,k}) + \sum_{k=1}^{N} c_{i,k} U_{ik} \right) \\ & + \sum_i I(1 - \sum_k c_{i,k}) + \sum_i \sum_k I(c_{i,k}) \\ & + \sum_i I(K_i - \sum_j \pi_{ij}) + \sum_i I(\sum_j \pi_{ij} - K_i) \\ & + \sum_i \sum_j I(\pi_{ij}) + \sum_i \sum_j I(1 - \pi_{ij}) \end{aligned} \tag{3.18}$$

3.4 ALGORITHM

The gradient and projection are repeatedly used in the algorithm, so first we formulate them.

The gradient of $f(\boldsymbol{c}, \Pi(k))$ is :
when $\boldsymbol{c} = (1, 0, 0)$,

$$\nabla(f_0(\boldsymbol{c}, \Pi(k))) \;=\; \mathbf{0} \tag{3.19}$$

when $\boldsymbol{c} = (0, 1, 0)$,

$$\begin{aligned}\nabla(f_1(\boldsymbol{c}, \Pi(k))) &= \nabla(\sum_i w_i \frac{c_{i1}}{x} (\sum_j \pi_{ij} (\frac{1}{\mu_j} + \frac{\Lambda_j \Gamma_j^2}{2(1-\rho_j)}))) \\ &= \frac{\sum_i w_i}{x} \left[\frac{1}{\mu_j} + \frac{\mu_j \Gamma_j^2}{2} (\sum_j \frac{\Lambda_j \mu_j}{(\mu_j - \Lambda_j)^2} + \frac{\Lambda_j \mu_j - \Lambda_j^2}{(\mu_j - \Lambda_j)^2}) \right] \end{aligned} \tag{3.20}$$

when $\boldsymbol{c} = (0, 0, 1)$,

$$\begin{aligned}
\nabla(f_2(\boldsymbol{c}, \Pi(k))) &= \nabla(\sum_i w_i \frac{c_{i2}}{x^2}(\sum_j \pi_{ij}({\sigma_j}^2 + \frac{\Lambda_j \hat{\Gamma}_j^3}{3(1-\rho_j)} + \frac{{\Lambda_j}^2 {\Gamma_j}^4}{2(1-\rho_j)^2} + \frac{1}{{\mu_j}^2} + \frac{\Lambda_j \Gamma_j^2}{\mu_j(1-\rho_j)})) \\
&= \frac{\sum_i w_i}{x^2}\left[\sigma_j^2 + \frac{1}{\mu_j} + (\Gamma_j^2 + \frac{\hat{\Gamma}_j^3 \mu_j}{3})(\sum_j \frac{\Lambda_j \mu_j}{(\mu_j - \Lambda_j)^2} + \frac{\Lambda_j \mu_j - \Lambda_j^2}{(\mu_j - \Lambda_j)^2})\right] \\
&\quad + \frac{\sum_i w_i}{x^2}\left[\frac{\Gamma_j^4 \mu_j^2}{2}(\sum_j \frac{2\Lambda_j^2 \Pi_{ij}}{(\mu_j - \Lambda_j)^3} + (\frac{\Lambda_j}{\mu_j - \Lambda_j})^2)\right] \qquad (3.21)
\end{aligned}$$

The projection is

$$z = \Pi(k) - \alpha_k \nabla(f(\boldsymbol{c}, \Pi(k)) \qquad (3.22)$$

$$P_C(z) = \underset{y \in g_2}{\operatorname{argmin}} \|z - y\|^2 \qquad \forall y \in g_2 \qquad (3.23)$$

$$g_2(\boldsymbol{\pi}) = I(K_i - \sum_j \pi_{ij}) + I(\sum_j \pi_{ij} - K_i) + I(\pi_{ij}) + I(1 - \pi_{ij}) \qquad (3.24)$$

Once the value of z has been calculated,check if $g_2(z) == 0$,if true,$\Pi(k+1) = z$,else,use the convex function $g_2(\boldsymbol{\pi})$ with x to find the minimizer.

Algorithm 1 Tail Latency Optimization With Arbitrary Service Distribution

Require: $\boldsymbol{\mu}$, $\boldsymbol{\Gamma}$, $\boldsymbol{\sigma}$, $\boldsymbol{\lambda}$, x

Initialize $t = 0$, $\epsilon_1 > 0, \epsilon_2 > 0$.
Initialize feasible $\boldsymbol{c}$, $\boldsymbol{\pi} = \text{argmin}_{\mathbf{c}}\, f(\boldsymbol{c}, \boldsymbol{\pi}) + g_2(\boldsymbol{\pi})$
Initialize feasible $H^{(0)}$, $H^{(-\frac{1}{2})}$, $H(c, \pi) = \sum_i \omega_i Pr(L_i \leq x)$
while $H^{(t)} - H^{(t-\frac{1}{2})} > \epsilon$
    $i = 0$
    while(file i)
        $n = \text{argmin}_n\, U_{i,n}$
        $\boldsymbol{c}(n) = 1$
        $\boldsymbol{c}(otherthan' n') = 0$
        update $i = i + 1$
    end while
    update $H^{(t+\frac{1}{2})} = H(\boldsymbol{c}, \boldsymbol{\pi})$
    Initialize feasible $\Pi(1) = \boldsymbol{\pi}, \Pi(0) = 0$, $k = 1$,step-size $\alpha_k = constant/t$
    while $\Pi(k) - \Pi(k-1) < \epsilon_2$
        $n = \text{argmax}_n\, c(i, n)$
        $\boldsymbol{z} = \Pi(k) - \alpha_k \nabla(f_n(\boldsymbol{c}, \Pi(k)))$
        $\Pi(k+1) = Pc(\boldsymbol{z'})$
        update $k = k + 1$
    end while
    $\boldsymbol{\pi} = \Pi(k)$
    update $H^{(t+1)} = H(\boldsymbol{c}, \boldsymbol{\pi})$
    update $t = t + 1$
end while

Ensure: $\boldsymbol{c}$, $\boldsymbol{\pi}$, $H^{(t)}$

3.5 SIMULATION AND EVALUATION

With the above algorithm, we are able to implement a simulation on this tail latency optimization. In this simulation we use realistic parameters from Tahoe [11], which is an open-source, distributed file system. The system consists 12 nodes. Mean service rate of each node is set to 5.77, 4.22, 3.95, 4.76, 3.03, 3.66, 2.88, 5.45, 3.26, 4.62, 2.48, 2.52. Second moment of service rate is 0.014, 0.014, 0.014, 0.015, 0.015, 0.015, 0.013, 0.013, 0.013, 0.016, 0.016, 0.016. Third moment of service rate is 0.015, 0.015, 0.015, 0.016, 0.016, 0.016, 0.014, 0.014, 0.014, 0.017. Deviance of service rate is 0.83. 4 kinds of files are assumed to store in the system with distinct erasure codes of (7, 6), (8, 7), (8, 6), and (6, 4). Arrival rate for each kind of files is 0.0354, 0.0236, 0.0354, 0.0236. Weight for each kind of files is 0.1, 0.2, 0.3, 0.4. In all below figures, X represents the value which cuts the tail.

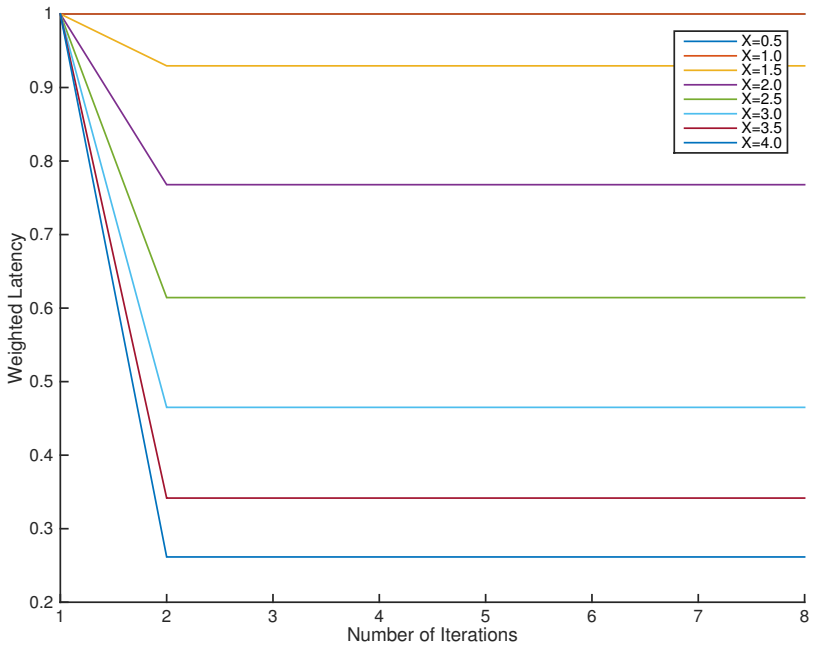

Figure 3.1. Convergence of Algorithm TLO with Arbitrary Service Distribution for varying X values from 0.5 to 4 seconds.

As under all parameters, the mean latency is around 1.0. The algorithm efficiently computes a solution within a few iterations.

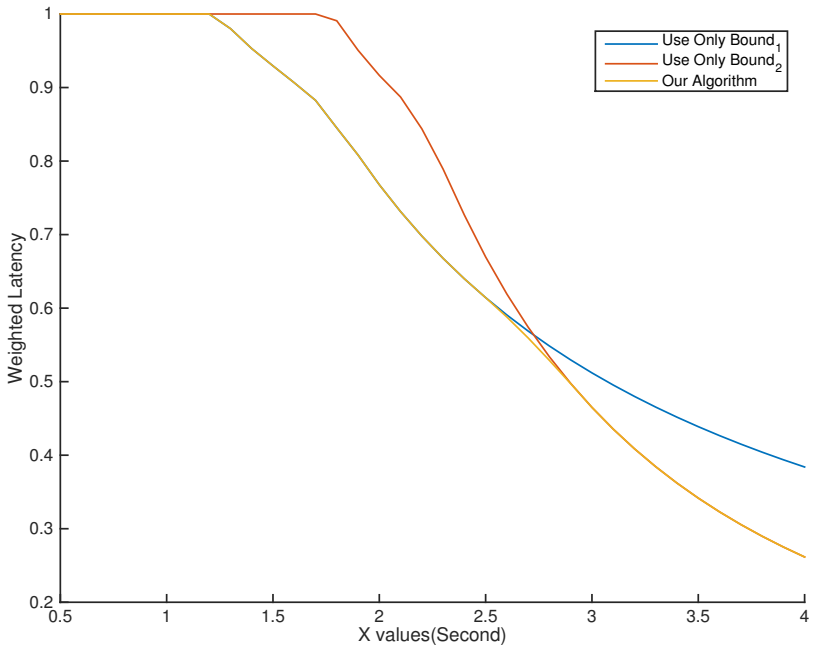

Figure 3.2. Plot of Weighted Latency and X values for 3 methods.

Here Bound 1 is the result if we only apply $U_{i,1}$ as the upper bound for file i, Bound 2 is the result if we only apply $U_{i,2}$ as the upper bound for file i. From figure 3.2, there are ranges when Bound 1 is lower while there are others when Bound 2 is lower. Our method is always at the lowest level as it applies these two's minimum. Between x in 2.5 to 2.9, these 2 bounds are combined to form our method's bound so that it is even lower than both.

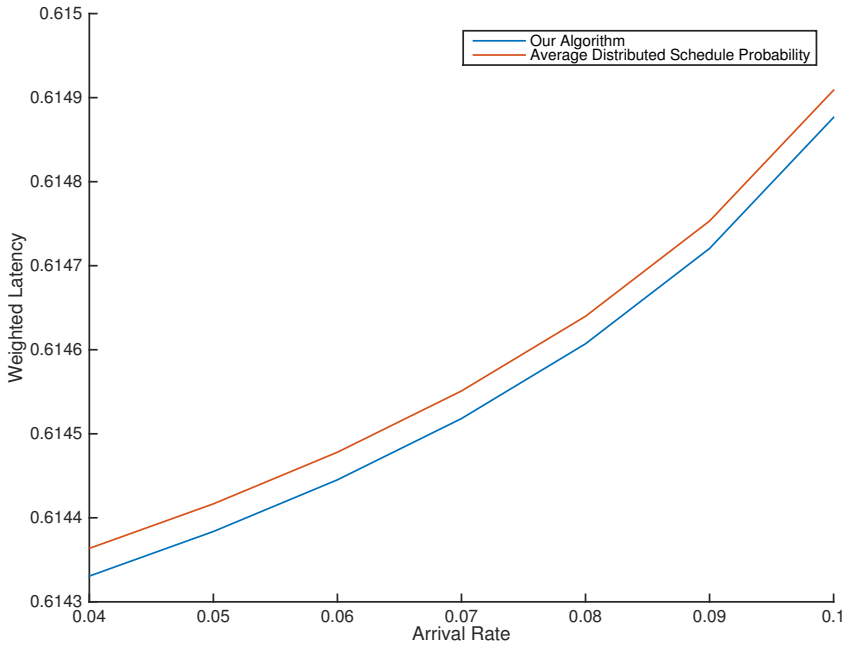

Figure 3.3. Trend of Weighted Latency and Arrival Rate of our algorithm and equal probability distributed schedule policy.

Both results are computed from the combination upper bound of all bounds. This shows our method provides lower latency over ranges of arrival rate.

3.6 SUMMARY

In this chapter, because of the limit that service time of each node is of arbitrary distribution, the only information for finding upper bound is the moments. Due to this deficiency, the final upper bound is optimized combination of all moments. With the Alternating Minimization Method we developed a proved efficient algorithm to examine our bounds.

The simulation results of this chapter show convergence and efficiency of our algorithm. Compared with simply applying one moment bound, our method is always tighter. However, when comparing our algorithm with equal probability distributed

schedule policy, our algorithm is tighter but the difference is not significant. This fact of results forwards us to go finding tighter upper bound for a more significant reduction on tail latency.

4. UNDER SHIFTED EXPONENTIAL SERVICE DISTRIBUTION: BOUNDS ON TAIL LATENCY, OPTIMIZATION AND SIMULATION RESULTS

4.1 UPPER BOUNDS ON TAIL LATENCY

In Chapter 3, we provide an upper bound generated by Markov Lemma with moments of service time. To simplify the optimization we only apply first two moments. However, the simulation results are not significant. Here in this chapter, to get a tighter and precise upper bound, we use the moment generating function of service time and find a more general form of upper bound with the Laplace Stieltjes Transform of $\mathbf{Q}_j$.

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

$$\mathbb{E}[e^{-sQ_j}] = \frac{(1-\rho_j)sM_j(-s)}{s-\Lambda_j(1-M_j(-s))}, \tag{4.1}$$

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. [12]. From Chapter 3 we already know that in order to get an upper bound for $\Pr(L_i \geq x)$, we need first use an upper bound on this and use that instead of $\Pr(L_i \geq x)$ in the objective. This upper bound has been proved as following:

$$\Pr(L_i \geq x) \;\; \leq \;\; \sum_j \pi_{ij} [\Pr\left(Q_j \geq x\right)]$$

In this chapter we use the exponential form of Markov Lemma:

$$\Pr\left(Q_j \geq x\right) \leq \frac{\mathbb{E}[e^{t_j Q_j}]}{e^{t_j x}}$$

In order to obtain $\mathbb{E}[e^{t_j Q_j}]$, we replace s in (4.1) with $-t$. Then we get

$$\Pr(Q_j \geq x) \leq \frac{(1-\rho_j)t_j M_j(t_j)}{e^{t_j x}(t_j - \Lambda_j(M_j(t_j)-1))}, \tag{4.2}$$

The expression is finite only when $\Lambda_j(M_j(t_j)-1) < t_j$. So our upper bound using Pollaczek-Khinchine formula for Laplace Stieltjes Transform is

$$\Pr(L_i \geq x) \leq \sum_j \frac{\pi_{ij}}{e^{t_j x}} \frac{(1-\rho_j)t_j M_j(t_j)}{t_j - \Lambda_j(M_j(t_j)-1)}, \tag{4.3}$$

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

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 use the results in chapter 3 to get the upper bound.

4.2 SHIFTED EXPONENTIAL SERVICE TIME DISTRIBUTION

Motivated by the Tahoe experiments [10] and Amazon S3 experiments, [5]. we consider the case when the service time distribution is a shifted exponential distribution. Let the service time distribution of server j has probability density function $f_{X_j}(x)$ as:

$$f_{X_j}(x) = \begin{cases} \alpha_j e^{-\alpha_j(x-\beta_j)}, & \text{for } x \geq \beta_j \\ 0, & \text{for } x < \beta_j \end{cases}. \tag{4.4}$$

Exponential distribution is a special case of shifted exponential when $\beta_j = 0$. Under shifted exponential distribution, the Moment Generating Function is now given as

$$M_j(t) = \frac{\alpha_j}{\alpha_j - t} e^{\beta_j t} \quad \text{for } t < \alpha_j. \tag{4.5}$$

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

$$\Pr(L_i \geq x) \leq \sum_j \frac{\pi_{ij}}{e^{t_j x}} \frac{(1-\rho_j)t_j M_j(t_j)}{t_j - \Lambda_j(M_j(t_j)-1)}, \tag{4.6}$$

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

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.

4.3 FORMULATION OF OPTIMIZED STORAGE SYSTEM

Similar to previous chapter, we consider ω_i as the weight of file i. Then we come up with the following Weighted Latency Tail Probability (WLTP) optimization problem over scheduling probabilities $\pi_{i,j}$ and parameter t_j, i.e.,

$$\min \quad \sum_i \omega_i \left(\sum_j \frac{\pi_{ij}}{e^{t_j x}} \frac{(1-\rho_j) t_j M_j(t_j)}{t_j - \Lambda_j(M_j(t_j)-1)} \right) \tag{4.7}$$

$$s.t. \quad \Lambda_j = \sum_i \lambda_i \pi_{ij} \tag{4.8}$$

$$M_j(t) = \frac{\alpha_j}{\alpha_j - t} e^{\beta_j t} \tag{4.9}$$

$$\rho_j = \frac{\Lambda_j}{\alpha_j} + \Lambda_j \beta_j \tag{4.10}$$

$$\sum_j \pi_{i,j} = k_i \tag{4.11}$$

$$\pi_{i,j} = 0, j \notin A_i \tag{4.12}$$

$$\pi_{i,j} \in [0,1] \tag{4.13}$$

$$t_j \geq 0 \tag{4.14}$$

$$t_j(t_j-\alpha_j+\Lambda_j)+\Lambda_j\alpha_j(e^{\beta_j t_j}-1) < 0 \tag{4.15}$$

$$var. \quad \pi_{i,j}, t_j \tag{4.16}$$

Here, Constraint (4.8) gives the aggregate arrival rate Λ_j for each node under give scheduling probabilities $\pi_{i,j}$ and arrival rates λ_i, Constraint (4.9) defines moment generating function with respect to parameter t_j, Constraint (4.10) defines the traffic intensity of the servers, Constraints (4.11-4.13) guarantees that the scheduling probabilities are feasible, and finally, the moment generating function exists due to the technical constraint in (4.15). If (4.15) 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 (4.14)) 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.

However, the proposed WLTP optimization is non-convex because constraint (4.15) is non-convex in both $\pi_{i,j}$ and t_j.

To develop an algorithmic solution to this non-convex optimization problem, we first prove that the problem is convex with respect to individual optimization variables, $\mathbf{t} = (t_1, t_2, \cdots, t_m)$and $\boldsymbol{\pi} = (\pi_{ij} \forall i = 1, \cdots, r, j = 1, \cdots, m)$, while the other one is fixed. With this convexity result we are able to propose an alternating optimization algorithm for this problem, which will be proven to be indeed optimal in later content.

4.3.1 CONVEXITY PROOF of OBJECTIVE FUNCTION IN t

The objective function, $\sum_i \omega_i \left(\sum_j \frac{\pi_{ij}}{e^{t_j x}} \frac{(1-\rho_j) t_j M_j(t_j)}{t_j - \Lambda_j (M_j(t_j) - 1)} \right)$ is convex in $\mathbf{t} = (t_1, t_2, \cdots, t_m)$ in the region where the constraints in (4.8)-(4.15) are satisfied. Below comes the detailed proof.

We note that inside the summation of (i, j), 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 subscript in the rest of this proof.

Let $f(\mathbf{x})$ and $g(\mathbf{x})$ be two non-negative differentiable convex functions of $\mathbf{x}$. If $\nabla f(\nabla g)^*$ is positive semi-definite, $F(\mathbf{x}) = f(\mathbf{x})g(\mathbf{x})$ is convex function of $\mathbf{x}$. We denote

$$\begin{aligned}
F(t) &= \frac{te^{-tx}M(t)}{t-\Lambda(M(t)-1)} && (4.17)\\
&= \frac{\alpha te^{(\beta-x)t}}{-t^2+(\alpha-\Lambda)t+\Lambda\alpha-\Lambda\alpha e^{\beta t}} && (4.18)\\
&= \frac{\alpha te^{(\beta-x)t}}{-t^2+(\alpha-\Lambda)t-\Lambda\alpha(e^{\beta t}-1)} && (4.19)\\
&= \frac{\alpha te^{(\beta-x)t}}{-t^2+(\alpha-\Lambda)t-\Lambda\alpha\sum_{u=1}^{\infty}\frac{(\beta t)^u}{u!}} && (4.20)\\
&= \frac{\alpha e^{(\beta-x)t}}{-t+(\alpha-\Lambda)-\Lambda\alpha\sum_{u=1}^{\infty}\frac{(\beta)^u t^{u-1}}{u!}} && (4.21)
\end{aligned}$$

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 (4.8)-(4.15) 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 $w'(t) \geq 0$.

$$\begin{aligned}
g(t) &= \frac{1}{h(t)}\\
g'(t) &= \frac{w(t)}{h^2(t)}\\
g''(t) &= \frac{h(t)w'(t)+2w^2(t)}{h^3(t)}\\
F''(t) &= f''(t)g(t)+f(t)g''(t)+2f'(t)g'(t)\\
&= \alpha e^{(\beta-x)t}\left(((\beta-x)^2g(t)+g''(t)+2(\beta-x)g'(t))\right)\\
&= \frac{\alpha e^{(\beta-x)t}}{h^3(t)}\left((\beta-x)^2h^2(t)+h(t)w'(t)+2w^2(t)\right.\\
&\quad \left.+2(\beta-x)w(t)h(t)\right)\\
&= \frac{\alpha e^{(\beta-x)t}}{h^3(t)}\left(2\left(\frac{(\beta-x)h(t)}{2}+w(t)\right)^2+h(t)w'(t)\right.\\
&\quad \left.+\frac{(\beta-x)^2h^2(t)}{4}\right)\\
&\geq 0, && (4.22)
\end{aligned}$$

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

4.3.2 CONVEXITY PROOF of OBJECTIVE FUNCTION IN π

The objective function, $\sum_i \omega_i \left(\sum_j \frac{\pi_{ij}}{e^{t_j x}} \frac{(1-\rho_j) t_j M_j(t_j)}{t_j - \Lambda_j (M_j(t_j)-1)} \right)$ is convex in $\boldsymbol{\pi} = (\pi_{ij} \forall (i,j))$. Below comes the detailed proof.

Since the sum of convex functions is convex, it is enough to show that

$$F_{i,j} = \pi_{ij} H_j, \tag{4.23}$$

where $H_j = \frac{1-\rho_j}{1-\Lambda_j (M_j(t_j)-1)/t_j}$ is convex w.r.t. $\boldsymbol{\pi}$. We first show that H_j is convex w.r.t. $\boldsymbol{\pi}$ with non-negative gradient. In order to see that, we first note that Λ_j is linear function of $\boldsymbol{\pi}$ with non-negative gradients. Since H_j depends on $\boldsymbol{\pi}$ only through Λ_j, it is enough to show that H_j is convex w.r.t. Λ_j. We note that H_j can be written as

$$H_j = \frac{1 - \Lambda_j C_1}{1 - \Lambda_j C_2}, \tag{4.24}$$

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

$$\frac{\delta}{\delta \Lambda_j} H_j = \frac{C_2 - C_1}{(1 - \Lambda_j C_2)^2} \geq 0 \tag{4.25}$$

$$\frac{\delta^2}{\delta \Lambda_j^2} H_j = 2 C_2 \frac{C_2 - C_1}{(1 - \Lambda_j C_2)^3} \geq 0. \tag{4.26}$$

Thus, $\nabla \pi_{ij}$ is convex w.r.t. $\boldsymbol{\pi}$ with non-negative gradients. π_{ij} is also convex w.r.t. $\boldsymbol{\pi}$, with non-negative gradients. To show that the product of these is convex, it is enough to show that $\nabla \pi_{ij} (\nabla H_j)^*$ is positive semi-definite, which is true here, as only one row is non-zero, and that row only has non-negative elements.

4.4 ALGORITHM

As proven in above section, the WLTP optimization problem is convex with respect to individual $\mathbf{t}$ and $\boldsymbol{\pi}$. In this section we propose an alternating minimization

algorithm to solve the WLTP problem. First we define two sub-problems as follows:

t-Optimization: Input $\boldsymbol{\pi}$

$$\begin{aligned} \min \quad & \sum_i \omega_i \left(\sum_j \frac{\pi_{ij}}{e^{t_j x}} \frac{(1-\rho_j) t_j M_j(t_j)}{t_j - \Lambda_j (M_j(t_j) - 1)} \right) \\ s.t. \quad & (4.8), (4.9), (4.10), (4.14), (4.15) \\ var. \quad & t_j \end{aligned}$$

π-Optimization: Input $\mathbf{t}$

$$\begin{aligned} \min \quad & \sum_i \omega_i \left(\sum_j \frac{\pi_{ij}}{e^{t_j x}} \frac{(1-\rho_j) t_j M_j(t_j)}{t_j - \Lambda_j (M_j(t_j) - 1)} \right) \\ s.t. \quad & (4.8), (4.9), (4.10), (4.11), (4.12), (4.13), (4.15) \\ var. \quad & \pi_{ij} \end{aligned}$$

As both these problems are proven to be convex, they can be solved using Projected Gradient Descent Algorithm. Using these two optimization as the building boxes, the proposed algorithm can be written as follows:

1. **Initialization:** Initialize π_{ij} and $t_j \ \forall \ (i, j)$ such that the choice is feasible for the problem.

2. **While Objective Converges:**

   - Run **t-Optimization** using current values of $\boldsymbol{\pi}$ to get new values of $\mathbf{t}$
   - Run **π-Optimization** using current values of $\mathbf{t}$ to get new values of $\boldsymbol{\pi}$

Since the constraint $t_j(t_j - \alpha_j + \Lambda_j) + \Lambda_j \alpha_j (e^{\beta_j t_j} - 1) < 0$ is non-convex in both $\pi_{i,j}$ and t_j, we consider a modified problem where this constraint is moved into our objective function. To do this, we add $LU(t_j(t_j - \alpha_j + \Lambda_j) + \Lambda_j \alpha_j (e^{\beta_j t_j} - 1) + \frac{1}{L^{1/10}})$ to the objective function where $L > 0$ and $U(x) = x^2$ where $x \geq 0$ and $U(x) = 0$ when $x < 0$. When $L \to \infty$, this is equivalent to $t_j(t_j - \alpha_j + \Lambda_j) + \Lambda_j \alpha_j (e^{\beta_j t_j} - 1) < 0$ as a constraint to the problem.

4.5 SIMULATION AND EVALUATION

4.5.1 NUMERIC SETTINGS

We denote our proposed latency optimization as Policy WLTP. To validate the proposed tail latency upper bound and optimization, we implement a simulation based on our policy and compare it with two other naive strategies. Below describe all three strategies we are going to compare in this section.

- Policy WLTP (*Weighted Latency Tail Probability optimization*): The joint scheduler is determined by the optimal solution that minimizes the weighted latency tail probabilities, with respect to our proposed tail latency bounds.

- Policy PEAP (*Projected, Equal Access-Probability*): For each file request, the joint request scheduler selects available chunks and nodes with equal probability. The equal access-probabilities are projected toward feasible region in (4.7) to ensure stability of the storage system.

- Policy BNW (*Balanced Node Workload*): The joint request scheduler is optimized to balance the workload of all storage nodes. This policy should minimize the chance of congested bottleneck in the storage system.

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 [5], we consider chunk service time that follows a shifted-exponential distribution with rate α_j and shift β_j.

Table 4.1.
Summary of parameters for 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	18.2295	24.0552	11.8750	17.0526	26.1912	23.9059
β_j	8.5368	13.6018	6.2756	9.5100	9.0524	12.1242
	Node 7	Node 8	Node 9	Node 10	Node 11	Node 12
α_j	27.0006	21.3812	9.9106	24.9589	26.5288	21.8067
β_j	12.3616	7.4950	9.9182	9.5646	11.1706	11.6750

As shown in Table 4.1, we have 12 heterogeneous storage nodes with different service speed and round-trip-time. The base arrival rates for the first 500 files are chosen as 0.02 s^{-1}, for the next 5000 files is chosen as 0.03 s^{-1}. The first 250 files are placed on first seven nodes, the next 250 files are placed on nodes 2 to 8, the next 250 files are placed on nodes 4 to 10, and the last 250 files are placed on nodes 6 to 12. This paper also considers different weights of the files - where the weights corresponding to the first 250 files are each chosen as $2/(15 \times 250)$, the weights corresponding to the next 250 files are chosen as $4/(15 \times 250)$, the weights corresponding to the next 250 files are chosen as $6/(15 \times 250)$, and the weights corresponding to the last 250 files are chosen as $3/(15 \times 250)$ such that the sum of weights of all files is 1. In order to initialize the algorithm, we choose $\pi_{ij} = k/n$ on the placed servers, all $t_j = .01$. But since these choices of π and $\mathbf{t}$ may not be feasible, we modify the initialization $\boldsymbol{\pi}$ to be the closest norm feasible solution to the above choice.

4.5.2 WEIGHTED LATENCY TAIL PROBABILITIES

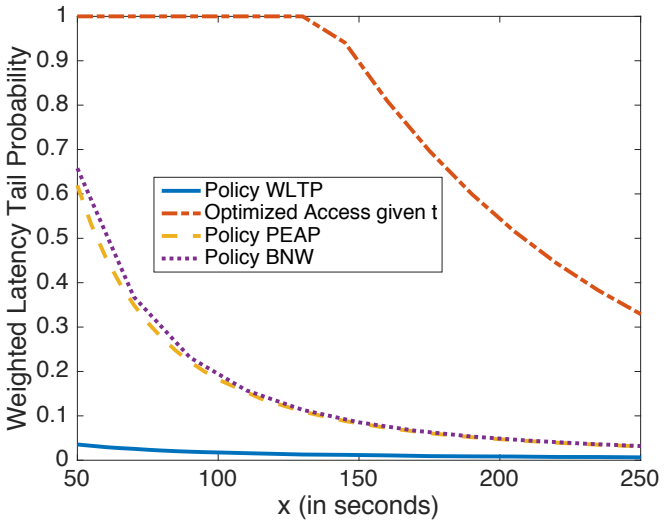

Figure 4.1. Weighted Latency Tail Probability vs x (in seconds) with Other Algorithms.

In Figure 4.1, 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, PEAP and BNW. Policy WLTP solves the optimal weighted latency tail probability via proposed alternative optimization algorithm over t_j and $\pi_{i,j}$. With fixed $\mathbf{t}$, Policy PEAP uses equal server access probabilities, projected toward the feasible region, while Policy BNW load-balances chunk requests across different servers. In particular, we have the first 250 files access the first 4 servers with equal probabilities, the last 250 files access the last 4 servers with equal probabilities, whereas files 251 to 500 access the 12 servers with probabilities [0 14 14 14 15 5 5 5 0 0 0 0]/18, and files 501 to 750 access the servers with probabilities [0 0 0 0 17 27 27 27 5 5 0 0]/27. With this choice, the aggregate arrival rate at the first server is 5, at the last two servers is 7.5 and the rest 9 servers is 8.8889. This achieves optimal load-balancing, because servers arrival rate at the first, 11th and 12th server can no longer be increased as each hosts only a single file.

We note that our proposed algorithm provides significant improvement over simple heuristics such as Policies PEAP and BNW, as weighted latency tail probability reduces by an orders of magnitude. For example, our proposed Policy WLTP decreases 90-percentile weighted latency (i.e., x such that $\sum_i \omega_i \Pr(L_i \geq x) \leq 0.1$) from 150 seconds to about 20 seconds. Uniformly accessing servers and simple load-balancing 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.

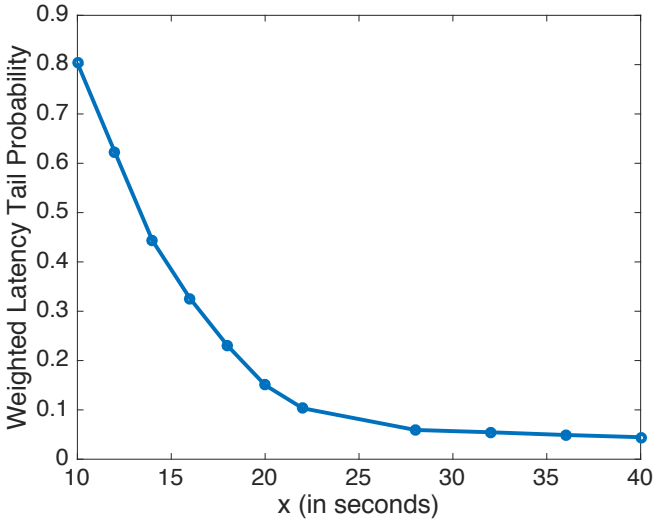

Figure 4.2. Weighted Latency Tail Probability vs x (in seconds) Near Mean Value.

Figure 4.2 further gives a zoomed-in plot for the weighted tail latency as a function of x when x varies from 10 to 40.

4.5.3 Tail Latency Reduction Speed of the Proposed Algorithm

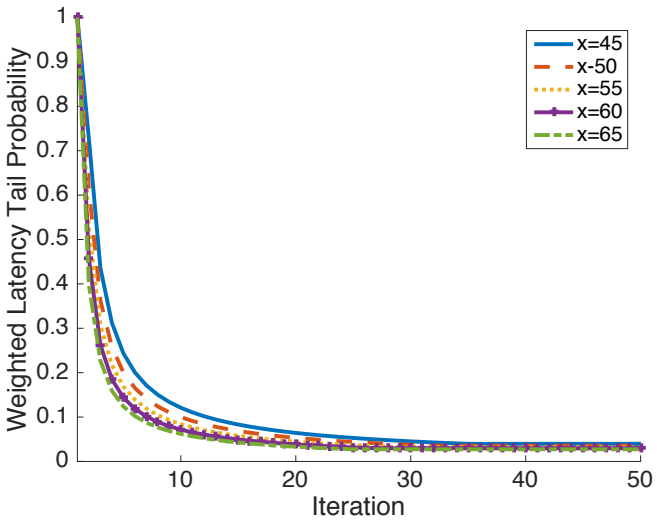

Figure 4.3. Reduction Speed of Weighted Latency Tail Probability.

Figure 4.3 shows the reduction speed of weighted latency tail probability and the number of iterations for different values of x ranging from 45 to 65 second in increments of 5 seconds to illustrate its tail latency reduction speed.

For 1000 files and 12 storage nodes, the weighted latency tail probability reduces very fast and within 40 iterations to a very low level, validating the efficiency of the proposed algorithm.

4.5.4 EFFECT OF ARRIVAL RATES

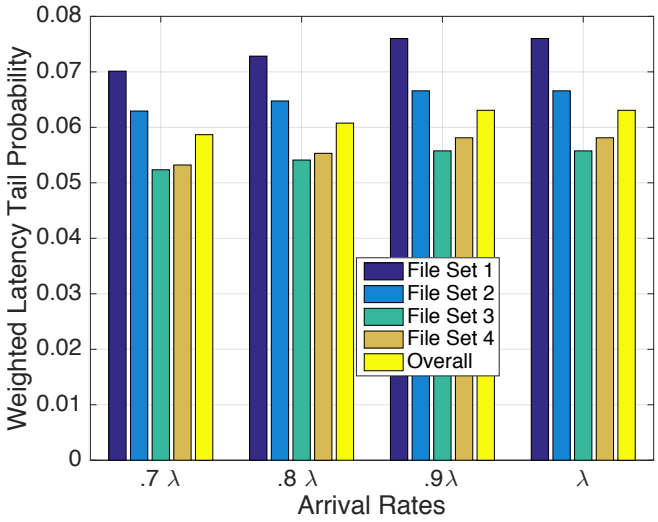

Figure 4.4. Weighted Latency Tail Probability for different file arrival rates .

We next want to see the impact of varying request arrival rates on the weighted latency tail probability. We choose $x = 25$ seconds and divide all files into 4 groups, each containing 250 consecutive files of equal weight. For λ as the base arrival rates, we increase arrival rate of all files from $.6\lambda$ to λ and plot the weighted latency tail probability for each group of files as well as the overall value in Figure 4.4.

While overall latency tail probability increases as arrival rate goes up, our algorithm assigns differentiated latency for different file groups. Here File Set 3 that has highest weight ω_3, which is the most tail latency sensitive set, always receive the minimum latency tail probability.

4.5.5 EFFECT OF NUMBER OF FILES

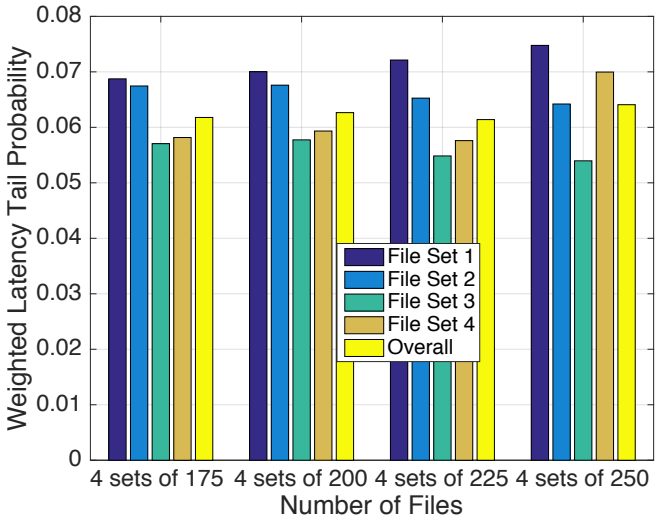

Figure 4.5. Weighted Latency Tail Probability for different number of files.

We then modify the number of files in each set from 250 in the base case to values such as 175, 200, and 225, as shown in Figure 4.5.

Weighted latency tail probabilities increases with the number of files, which brings in more workload, meaning higher arrival rates. From Figure 4.5 we can tell that even the number of files increase from 4 sets of 175 to 4 sets of 225, the overall weighted latency tail probabilities increases very little. Our optimization algorithm optimizes new files along with existing ones to keep overall latency tail probability at a very low level.

4.5.6 EFFECT OF FILE SIZES

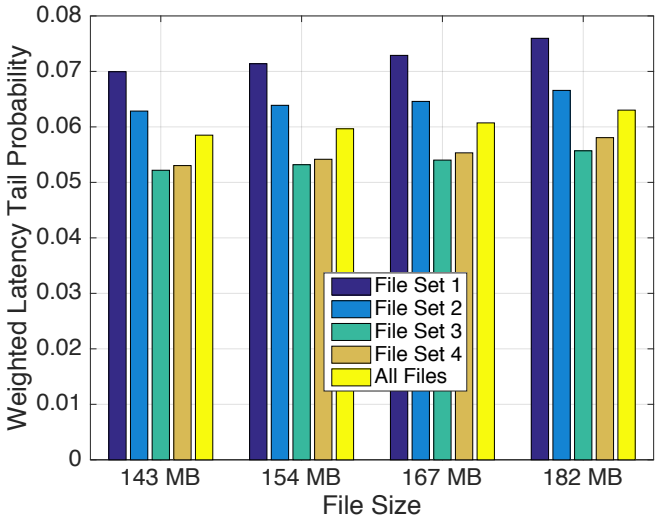

Figure 4.6. Weighted Latency Tail Probability for different size of files.

File size is varied in our simulation as 143 MB, 154 MB, 167 MB, and 182 MB. We then plot the optimal weighted latency tail probability with file size in Figure 4.6. In order to capture the effect of file size as compared to a file size of 200 MB, the value of α increases in proportion to the chunk size, and the value of β decreases in proportion to the chunk size accordingly.

Increasing file size results in higher tail latency for all 4 groups of files. While File Set 1 that are assigned the lowest weight ω_1, which is the least tail latency sensitive, suffers most as file size increases. With the sacrifice of File Set 1, the tail latency of the rest of file sets increases a little. And thus our optimization algorithm manages to minimize the overall weighted tail latency even when file size increases.

4.6 SUMMARY

In this chapter we use a more detailed upper bound on tail latency and in order to get deeper and specific sense of tail latency bounds we apply the shifted exponential

service time distribution according to the results of previous experiments. Due to these improvements, the optimization problem becomes a non-convex problem and more complicated. But by reformulating the optimization problem and modifying our algorithm, we managed to provide a heuristic algorithm solution. Based on the above, we also perform thorough simulations to validate our algorithm. Simulation results show significant reduction of tail latency for erasure-coded storage systems with realistic workload.

5. CONCLUSIONS AND FUTURE WORK

This chapter summarizes the entire master thesis and discusses about future possible works of this current research.

5.1 CONCLUSIONS

In response to the data storage method change from full data replication to erasure coding in distributed storage systems, this thesis works on building analytical models under this transition and finds upper bound in closed form of tail latency to optimize the performance of distributed storage systems. Furthermore, to validate the proposed upper bounds and the efficiency of algorithm, this thesis performs simulations and numeric evaluations. To solve the research problem of building an analytical framework to quantify tail latency in erasure-coded, distributed storage systems and izing the tail latency, three research questions have been defined and answered.

For research question 1, we present a system model with erasure coding data storage. This thesis explains how erasure coding works with k out of n encoded replication chunks and how these chunks are placed among different storage nodes in a distributed storage system. With the basic environment build-up, this thesis introduces probabilistic scheduling policy for how to model the process of retrieve data chunks. After pointing out the flaw of previous approaches working on tail latency, we reason the probabilistic scheduling policy is suitable and workable for our analysis goals. Then we consider the actual file request and process situation to formulate a queuing model into the system. Arrival of client requests for individual files is assumed to follow Poisson process while the service time of server nodes is initially set to arbitrary distribution to find general results. Later in chapter 4, motivated by the Tahoe experiments and Amazon S3 experiments, shifted exponential service time

distribution is introduced into our model and specify our upper bounds then thus enable our research to give significant optimization results.

For research question 2, this thesis provides weighted tail latency upper bounds in closed form in different forms. On the first stage, we deduce a brief form with moments of service time using Markov Inequality. In order to get the tightest bound possible, we introduce a new set of parameters as $c_{i,k}$ to adjust the upper bound to the minimum.

On the second stage, we update our upper bound to a more general form by using exponential form of Markov Inequality to put moment generating function of Q_j given by Pollaczek-Khinchine formula directly into our bound, which is a Laplace Stieltjes transform. Thus in the result of our weighted latency we have the moment generating function and Laplace variable. Due to this change, in order to make the upper bound valid, we also have more constraints on all variables. With the improved form for any service time distribution, we formulate further upper bound for shifted exponential service time distribution.

For research question 3, the main method we apply on our optimization problem is the Alternating Minimization Method [13]. The optimization problem of the first stage is proven to be a convex problem while the problem of the second stage is non-convex due to one of the constraints but we manage to modify the objective function to fulfill the requirements of using the Alternating Minimization Method. Before use the method we first examine the five assumptions and then prove its optimality. The detailed proof is presented in the appendix. The numeric evaluations show that our results of the first stage is convergence and valid, but lack the significance of reduction on tail latency compared to other strategies.

For the second stage of this thesis, we performed more evaluations to see the effect of other factors on the weighted tail latency optimized from our results. We note that in the simulation results of the second stage of our research, our algorithm significantly reduce the weighted tail latency compared with other naive strategies and reduces weighted latency to a very low level within reasonable iterations. And

we analyze the effect of arrival rates, number of files and file size, all the results show that our algorithm is effectively minimize the weighted latency and maintain it at a low level.

5.2 FUTURE WORK

With the current results of this thesis, we intend to have further work to do in the future. Since the simulation results of chapter 4 have shown that our algorithm has efficiently reduced the weighted tail latency, we intend to implement our algorithm in the realistic practice. In the future we want to perform an experiment on the Tahoe Testbed with our algorithm.

The *Tahoe Testbed* is an open-source, distributed filesystem based on the *zfec* erasure coding library. It provides three special instances of a generic node: 1) *Tahoe Introducer:* It keeps track of a collection of storage servers and clients and introduces them to each other. 2) *Tahoe Storage Server:* It exposes attached storage to external clients and stores erasure-coded shares. 3) *Tahoe Client:* It processes upload/download requests and connects to storage servers through a Web-based REST API and the Tahoe-LAFS (Least-Authority File System) storage protocol over SSL.

Our algorithm requires customized erasure code, chunk placement, and server selection algorithms. While Tahoe uses a default (10, 3) erasure code, it supports arbitrary erasure code specification statically through a configuration file. In Tahoe, each file is encrypted and then broken into a set of segments, where each segment consists of k blocks. Each segment is then erasure-coded to produce n blocks using an (n, k) encoding scheme and then distributed to n distinct storage servers. The set of blocks on each storage server constitute a chunk. Thus the file equivalently consists of k chunks that are encoded into n chunks and each chunk consist of multiple blocks. The Tahoe client randomly selects a set of available storage servers with enough storage space to store n chunks. For server selection during file retrievals the client first asks all known servers for the storage chunks they might have. Once it knows

where to find the needed k chunks from the k servers that respond the fastest, it downloads at least the first segment from those servers. This means that it tends to download chunks from the "fastest" servers purely based on round-trip times.

If the realistic experiment succeeds to validate our algorithm, our algorithm can work on erasure-coded, distributed storage systems for companies who are bothered with long tail latency problems.

REFERENCES

REFERENCES

[1] Adam Shwartz Franois Baccelli, Armand M. Makowski. The fork-join queue and related systems with synchronization constraints: Stochastic ordering and computable bounds. *Advances in Applied Probability*, 21(3):629–660, 1989.

[2] Guanfeng Liang and Ulaş C. Kozat. Fast cloud: Pushing the envelope on delay performance of cloud storage with coding. *IEEE/ACM Trans. Netw.*, 22(6):2012–2025, December 2014.

[3] Guanfeng Liang and Ulas C. Kozat. TOFEC: achieving optimal throughput-delay trade-off of cloud storage using erasure codes. *CoRR*, abs/1307.8083, 2013.

[4] Gauri Joshi, Yanpei Liu, and Emina Soljanin. On the delay-storage trade-off in content download from coded distributed storage systems. *CoRR*, abs/1305.3945, 2013.

[5] Shengbo Chen, Yin Sun, Ulas C. Kozat, Longbo Huang, Prasun Sinha, Guanfeng Liang, Xin Liu, and Ness B. Shroff. When queueing meets coding: Optimal-latency data retrieving scheme in storage clouds. *CoRR*, abs/1404.6687, 2014.

[6] Akshay Kumar, Ravi Tandon, and T. Charles Clancy. On the latency of erasure-coded cloud storage systems. *CoRR*, abs/1405.2833, 2014.

[7] Virag Shah and Gustavo de Veciana. Performance evaluation and asymptotics for content delivery networks.

[8] L. Huang, S. Pawar, H. Zhang, and K. Ramchandran. Codes Can Reduce Queueing Delay in Data Centers. *ArXiv e-prints*, February 2012.

[9] Nihar B Shah, Kangwook Lee, and Kannan Ramchandran. The mds queue: Analysing latency performance of codes and redundant requests. 2013.

[10] Yu Xiang, Tian Lan, Vaneet Aggarwal, and Yih-Farn Robin Chen. Joint latency and cost optimization for erasure-coded data center storage. *CoRR*, abs/1404.4975, 2014.

[11] B Warner, Z Wilcox-O?Hearn, and R Kinninmont. Tahoe-lafs docs, 2015.

[12] L. Kleinrock. *Queueing Systems: Theory*. Number v. 1 in A Wiley-Interscience publication. Wiley, 1976.

[13] Amir Beck. On the convergence of alternating minimization for convex programming with applications to iteratively reweighted least squares and decomposition schemes. *SIAM Journal on Optimization*, 25(1):185–209, 2015.