Privacy and Optimization for Distributed Machine Learning and Storage

by

Yaochen Hu

A thesis submitted in partial fulfillment of the requirements for the degree of

Doctor of Philosophy

in

Computer Engineering

Department of Electrical and Computer Engineering

University of Alberta

Abstract

We study the problem of applying optimization methods under realistic and challenging constraints such as privacy, efficiency, etc., and demonstrate our design over two different cases: distributed machine learning with distributed features and load balancing for erasure-coded cloud storage systems.

First, we study an important distributed machine learning problem where features are inherently distributed or vertically partitioned among multiple parties, and sharing of raw data or model parameters among parties is prohibited due to privacy concerns. We propose an asynchronous stochastic gradient descent (SGD) algorithm for such a feature distributed machine learning (FDML) problem, with theoretical convergence guarantees under bounded asynchrony. We also propose an alternating direction method of multipliers (ADMM) sharing framework, a more robust approach over SGD method especially for data set with high dimensional feature spaces and establish convergence and iteration complexity results under non-convex loss. Besides, we introduce a novel technique from differential privacy for both algorithms to further protect the data privacy.

Second, we study the load balancing and tail latency reduction problem for erasure coded cloud storage systems, which lacks the flexibility of load balancing. We provide a new perspective by proactively and intelligently launching degraded reads and propose a variety of strategies on when and where to launch the degraded reads. We also solve the load balancing problem by block migration for statistical load balancing and propose a local block migration scheme with theoretical approximation ratio.

To my parents and Tong Li.

“Everything with form is unreal.”
Shakyamuni

“Do or do not, there is no try.”
Master Yoda

Acknowledgements

First of all, I would like to thank Professor Di Niu, my PhD supervisor for the last 6 years and a half. He is sharp on finding cutting edge research topics and efficient in bringing ideas into works. I was deeply affected by his passion and perfection towards the writing in every paper and I have learned valuable knowledge on how to be professional as a researcher. Besides, he is also a good mentor for developing a professional career and gave me valuable suggestions on how to be successful in lifetime. He provided lots of opportunities to collaborate with industry from which I gained valuable experience on how to apply theory to solve real world problems.

I would like to thank the colleagues in DSC, MIG, Tencent. Specifically, I am especially grateful to Jianming Yang and Gang Luo. They raised interesting industry problems, from which I came up with several ideas that contributed significantly to my research works. I would like to thank Jianchao Tu, who provided help in constructing the prototype system in my research work. I would like to thank Dong Wu, who was a very productive person and helped me on collecting and preprocessing the data. Besides I would like to thank Tengjiao Guo, Feiquan Huang and Shan Zhu for their support on my research. I would like to thank Zheng Cai, Lei Chen, Xue Chen, Xiawei Duan, Tengjiao Guo, Feiquan Huang, Beibei Kong, Gang Luo, Chi Ma, Zhiwei Shen, Lianzhi Tan, Dong Wu, Boyi Yang, Chao Yang, Dan Yi, Liting Zhang, Zhaoyong Zhang and Shan Zhu for our precious friendship and their support to my internship in Tencent. I would also like thank Jianming Yang, Yiwei Pan, Shengping Zhou, Jian Xiong and Lin Nie for their kind support.

I would like to thank Professor Zongpeng Li for his supervising in one of my early works. I was impressed by his enthusiasm and agile mind to research. I would like to thank Professor Linglong Kong and Peng Liu from mathematical department, they are professional mathematician who gave me important support in one of my theoretical research works. I would like to thank Dr. Cheng Huang from Microsoft reach for his valuable suggestions in my work on load balancing problem. Moreover, I would like to thank Dr. Majid Khabbazian and Dr. Marek Reformat for being the member in my PhD supervisor committee.

I would like to thank the members I have met in Professor Niu's group for our precious friendship and collaborations. These include Haolan chen, Fred X Han, Chenglin Li, Bang Liu, Yan Liu, Qikai Lu, Keith Mills, Dashun Wang, Shuopeng Zhang, Ting Zhang, Xv Zhang, Mingjun Zhao and Rui Zhu. They are excellent and intelligent people and I have gained much knowledge and wonderful ideas from the discussion with them. Specifically, I have collaborated with Bang Liu, Shuopeng Zhang and Rui Zhu, who have provided precious suggestions and insights to our works. I enjoyed discussing interesting math problems that yields contra-intuitive results with Xv Zhang. I learned a lot from the conversation with Haolan Chen.

Lastly, I would like to thank my family for their love, support and encouragement to me.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Optimization methods have been widely studied since most modern applications or some of their submodules can be modeled and improved as an optimization problem. However, there are usually some critical gaps between theory and application to apply the optimization methods in real problems. In this dissertation, we study two different problems with challenging optimization constraints. The first one is to train a machine learning model over distributed features while keeping the features local and confidential. Unlike most existing distributed machine learning works that focuses on speeding up the training by parallel computing, we design the distributed system and algorithm to utilize the distributed features which is not sharable due to privacy issue. The second one is to reduce the tail latency and balance the load for erasure coded cloud storage system, which is lack of flexible load direction means. We provide several different optimization approaches in order to deal with the challenging constraints, such as computation overhead, probing overhead, content placement feasibility, content migration overhead, etc.

1.1 Privacy and Distributed Machine Learning

Machine learning is one of the most popular topics as it has been proved to be a powerful method to extract useful information from vast data. Most machine learning methods rely on minimizing some loss or cost functions, which is essentially optimization problems. The effectiveness of a machine learning model highly depends on availability of data, especially high-quality features. However, vast amount of data is naturally distributed and cannot be shared due to privacy issues or regulation. Examples include a user's behavioural data logged by multiple apps, a patient's record stored at different hospitals and clinics, a user's investment behavior logged by multiple financial institutions and government agencies and so forth. To fully extract the value of those distributed data, we need to find some method which keeps the locality of data and minimize the information leakage other than those directly correlated to target applications. The major challenge here is to solve the optimization problem during training while maintaining the privacy of the data.

To solve the above challenges, we design, implement and extensively evaluate a practical

Feature Distributed Machine Learning (FDML) system based on real-world datasets. For any supervised learning task, e.g., classification, our system enables each party to use an arbitrary model (e.g., logistic regression, factorization machine, SVM, and deep neural networks) to map its local feature set to a local prediction, while different local predictions are aggregated into a final prediction for classification via a "hyper-linear structure", which is similar to softmax.

We propose two different algorithms to train model in FDML. One is a mini-batched stochastic gradient descent (SGD) algorithm performed in the sense of stale synchronous parallel (SSP) [Ho *et al.*, 2013], i.e., different parties are allowed to be at different iterations of parameter updates up to a bounded delay. We theoretically establish a convergence rate of $O(\frac{1}{\sqrt{T}})$ for the proposed asynchronous FDML algorithm under certain assumptions (including the bounded delay assumption[Ho *et al.*, 2013]), where T is the number of iterations on (the slowest) party, which matches the standard convergence rate of fully centralized synchronous SGD training with a convex loss as well as that known for asynchronously distributed data-parallel SGD in SSP[Ho *et al.*, 2013]. The other algorithm is alternating direction method of multipliers (ADMM) sharing based algorithm. We establish theoretical convergence guarantees and iteration complexity results under the non-convex loss in a fully parallel setting, whereas previously, the convergence of ADMM sharing algorithm for non-convex losses is only known for the case of sequential (Gauss-Seidel) execution[Hong *et al.*, 2016]. Experiment results shows the effectiveness of both algorithms. Specifically, the SGD-based algorithm is more versatile to different local models while the ADMM-based algorithm is more robust and scalable to large number of features.

A highlight of our FDML system is that during each training iteration, every party is solely responsible for updating its own local model parameters (local net) using its local feature sets, and for each record, only needs to share its local prediction to the central server (or to other parties directly in a fully decentralized scenario). Since neither the original features nor the local model parameters of a party are transferred to any external sites, the FDML system preserves data locality and is much less vulnerable to model inversion attacks[Hitaj *et al.*, 2017] targeting other collaborative learning algorithms[Shokri and Shmatikov, 2015; Zhou *et al.*, 2016] that share model parameters between parties. Moreover, we further enhance the data privacy by adopting a differential-privacy-based method[Dwork, 2008; Dwork *et al.*, 2014; Shokri and Shmatikov, 2015] by adding some perturbations to the shared local predictions.

1.2 Load Balancing in Erasure Coded Storage Systems

Cloud storage systems, such as Hadoop Distributed File System (HDFS)[Borthakur, 2008], Google File System (GFS)[Ghemawat *et al.*, 2003] and Windows Azure Storage (WAS)[Calder *et al.*, 2011], store large amounts of enterprise-level and personal data. Since these systems rely on commodity servers in datacenters, content must be replicated (e.g., for 3 times in

HDFS) for fault-tolerance. Erasure coding, e.g., an (k, r) systematic Reed-Solomon (RS) code, is further adopted in many production systems, e.g., Windows Azure Storage[Huang *et al.*, 2012], Google's ColossusFS, Facebook's HDFS, to offer significantly higher reliability than data replication at a much lower storage cost[Weatherspoon and Kubiatowicz, 2002; Khan *et al.*, 2012]. In these system, the uncoded block is stored on only one server and reading requests are directed to those blocks primarily unless the data block is unavailable due to server congestion or server failure. Degraded reads that reconstruct the content by retrieving information from several coded blocks will be launched for unavailable data. Unlike the traditional 3-replica system that has convenient ad-hoc means to balance the load, the unbalanced load and long tail latency is more likely to happen in a erasure coded system.

The problem of latency reduction and load balancing problem can be modeled as optimization problem in both the request direction phase and the content placement phase. The challenges lies in several aspects. In the load direction phase, the time complexity is critical for decision making since it may eventually increase the latency. Moreover, different strategies incur different levels of probing overhead that measures the queuing status of candidate servers. Therefore the balance between the accuracy and efficiency of the optimization methods is critical. In the content placement phase, the solution to the related optimization problem will influence the placement of content, and the overhead of migrating content blocks cannot be ignored. Therefore we need to find the solution efficiently in terms of time complexity and/or the overhead imposed to the real system.

For load direction, we propose to *proactively and intelligently* launch degraded reads in order to shift loads away from hotspots and prevent potential congestion early. Intuitively speaking, if a hot object is attracting a large number of requests which may potentially congest its original storage node, we may serve some of these requests through degraded reads in the first place, without waiting for normal read timeouts. Although proactive degraded reads may reduce the longest queues, they may flood the system with more reading tasks and affect the service latencies for other requests in general. Therefore, we must carefully decide: 1) for which request a degraded read should be performed, and 2) should a degraded read be used, from which storage nodes the degraded read should be served. Toward these objectives, we propose a variety of load balancing approaches to reduce latencies in erasure coded storage systems, including statistical optimization that can globally coordinate different requests and per-request optimal decisions. A first approach is an efficient optimization framework that intelligently maintains a load direction table between all requests and storage nodes, based on periodically sampled demand and queue statistics. This approach is sub-optimal since it only updates direction decisions periodically, failing to utilize instantaneous load information. We then naturally turn to per-request optimal decisions, one of which is *least latency first*, that is to serve each request with the normal read or a degraded read, whichever minimizes the current request latency. However, this may lead to an excessive number of degraded reads and increase overall system burden, affecting future requests. To solve this issue, we introduce the key notion of *marginal load* and propose a novel *least-marginal-load-first* policy which judiciously and lazily launches degraded reads for load balancing based on server queue length statistics, without flooding the system. To reduce the server queue length probing overhead, we further adapt the power-of-two sampling idea to our per-request optimal load balancing in coded storage systems. We show that the per-request optimal decision is essentially the optimal solution to the statistical optimization problem for each single request with a specific objective function.

For content placement, we place blocks with *anti-correlated* demands on a same server to benefit from statistical multiplexing and prevent certain hot blocks from congesting a specific server. We formulate the content placement optimization to minimize the expected average waiting time of all incoming requests, based on the mean and covariance of demands for different blocks, which can be readily measured according to recent request history. To avoid globally shuffling blocks across the system, we require all block migration to be *local*, and move as few blocks as possible with respect to the current placement to reduce the migration overhead. Our statistical content placement problem is similar to the Min-k-Partition problem, a well-known NP-complete problem[Kann *et al.*, 1997; Karp, 1972], which aims to divide a graph into k partitions to minimize the sum of all intra-partition edge weights. Yet, our problem turns out to be even more challenging, since we also need to handle an additional constraint that no blocks from the same coded group can be placed on the same server, which is needed to maintain the promised reliability of an (k, r) RS code. We introduce a novel technique to convert this constraint into carefully designed weights in the objective function and propose a time-efficient local search algorithm which only moves the block that reduces the latency objective the most at a time. We prove that our algorithm always produces a feasible solution that satisfies the special constraint and theoretically derive the worst-case approximation ratio of our algorithm with respect to the global optimality. We characterize such a ratio as a function of demand statistics; the larger the demand variation among blocks, the better the approximation.

1.3 Thesis Organization and Contributions

We first introduce feature distributed machine learning problem and the SGD based method to train the machine learning model in Chapter 2 [Hu *et al.*, 2019]. Our contributions are

- Proposing an effective feature distributed machine learning (FDML) model that utilize the distributed features while keeping them local and confidential. (Sec. 2.2)
- Proposing a stale synchronized SGD based training algorithm. (Sec. 2.3, Sec. 2.4)
- Theoretically deriving the convergence rate of $O(1/\sqrt{T})$, T being the number of training iterations. (Sec. 2.5)

We introduce the ADMM based method to train the FDML model in Chapter 3 and our contributions are

- Proposing a parallel ADMM sharing algorithm to train the FDML model. (Sec. 3.3)
- Deriving the convergence of the proposed algorithm as well as the iteration complexity for non-convex loss functions, where the existing work only analyzes the case of sequential (Gauss-Seidel) update. (Sec. 3.3)
- Combining technique from differential privacy into the proposed algorithm and derived the privacy loss. (Sec. 3.4)

We introduce the tail latency reduction problem for erasure coded cloud storage system and our proposed efficient optimization method in Chapter 4 [Hu *et al.*, 2017]. Our contributions are

- Proposing proactive degraded reading approach to shift loads away from hotspots and prevent potential congestion early. (Sec. 4.2)
- Proposing varieties of proactive degraded reading strategies including statistical optimization, per-request optimization by least loaded first or least marginal load first and power-of-two sampling. (Sec. 4.2)
- Implementing a prototype system and verifying our strategies over a real data trace collected from Windows Azure Storage. (Sec. 4.3, Sec. 4.4)

In Chapter 5, we introduce another optimization method to the load balancing problem via a local block migration algorithm [Hu and Niu, 2016]. Our contributions are

- Proposing to reduce the access latency in coded storage systems through a new approach of block placement adjustment and controlled block migration. (Sec. 5.2)
- Modeling the problem statistically and transforming the problem into min-k-partition problem by eliminating the challenging constraints of placement feasibility with a penalty term. (Sec. 5.2)
- Proposing the local block migration algorithm with theoretical guarantee of feasibility and approximation ratio. (Sec. 5.3)

Chapter 6 concludes the dissertation and proposes promising future directions.

Chapter 2

SGD Based Method to Distributed Machine Learning over Distributed Features

2.1 Introduction

While the success of modern machine learning lays the foundation of many intelligent services, the performance of a model often depends on the availability of data. In most applications, however, a large quantity of useful data may be generated on and held by multiple parties. Collecting such data to a central site for training incurs extra management and business compliance overhead, privacy concerns, or even regulation and judicial issues. As a result, a number of distributed machine learning techniques have been proposed to collaboratively train a model by letting each party perform local model updates and exchange locally computed gradients [Shokri and Shmatikov, 2015] or model parameters[McMahan *et al.*, 2016] with the central server to iteratively improve model accuracy. Most of the existing schemes, however, fall into the range of *data parallel* computation, where the training samples are located on different parties. For example, different users hold different images to jointly train a classifier. Different organizations may contribute their individual corpora to learn a joint language model.

We study distributed machine learning based on another motivation, where different *features* of a same sample are held by different parties. The question is—can we improve the predictive power at one party by leveraging additional features from another domain or party, yet without requiring any party to share its features? This is a real problem we are solving at *Tencent MyApp*, one of the largest Android app stores in China, with a market share of 24.7% in China in 2017. *Tencent MyApp* performs app recommendation and activation rate prediction based on the user and app features logged in its own platform. However, it turns out other Tencent apps including *Tencent QQ Browser* and *Tencent Mobile Safeguard* share a large number of common users with MyApp. Since these apps may have complementary information about a user, such cross-domain knowledge from another app, if utilized, may help to train a joint model that can improve app recommendation and customer behavior preference prediction in MyApp. However, due to privacy and customer protection regulations, raw customer data are not to be shared across apps that belong to different departments.

A natural question is—how can we train a joint machine learning model if the features of each training sample are located on multiple distributed parties? To make the solution practical with the most conservative assumption on information sharing, we bear the following goals:

- To minimize information leakage, no party should share its feature set. Neither should any of its local model parameters be communicated to other parties.
- The prediction made by the joint model should outperform the prediction made by each isolated model trained only with a single party's feature set, provided that such improvement from joint features also exists in centralized training.
- The joint model produced should approach the model trained in a centralized manner if all the features were collected centrally.
- The system should be efficient in the presence of both large numbers of features and samples.

To solve the above challenges, in this chapter, we design, implement and extensively evaluate a practical Feature Distributed Machine Learning (FDML) system based on real-world datasets. For any supervised learning task, e.g., classification, our system enables each party to use an arbitrary model (e.g., logistic regression, factorization machine, SVM, and deep neural networks) to map its local feature set to a local prediction, while different local predictions are aggregated into a final prediction for classification via a "hyper-linear structure," which is similar to softmax. The entire model is trained end-to-end using a mini-batched stochastic gradient descent (SGD) algorithm performed in the sense of stale synchronous parallel (SSP)[Ho *et al.*, 2013], i.e., different parties are allowed to be at different iterations of parameter updates up to a bounded delay.

A highlight of our system is that during each training iteration, every party is solely responsible for updating its own local model parameters (local net) using its own mini-batch of local feature sets, and for each record, only needs to share its local prediction to the central server (or to other parties directly in a fully decentralized scenario). Since neither the original features nor the local model parameters of a party are transferred to any external sites, the FDML system preserves data locality and is much less vulnerable to model inversion attacks[Hitaj *et al.*, 2017] targeting other collaborative learning algorithms[Shokri and Shmatikov, 2015; Zhou *et al.*, 2016] that share model parameters between parties. Moreover, we further enhance the data privacy by adopting a differential-privacy-based method[Dwork, 2008; Dwork *et al.*, 2014; Shokri and Shmatikov, 2015]. by adding some perturbations to the shared local predictions.

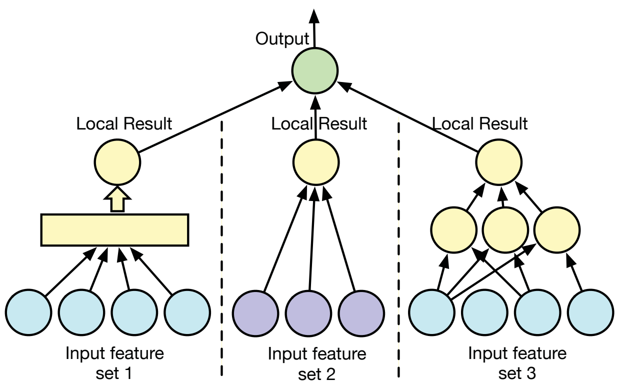


Figure 2.1: Illustrating FDML model (2.2), where the local predictions, each depending on the local model on a party, are aggregated into a final output using linear and nonlinear transformations (2.1).

We theoretically establish a convergence rate of $O(\frac{1}{\sqrt{T}})$ for the proposed asynchronous FDML algorithm under certain assumptions (including the bounded delay assumption[Ho *et al.*, 2013]), where T is the number of iterations on (the slowest) party, which matches the standard convergence rate of fully centralized synchronous SGD training with a convex loss as well as that known for asynchronously distributed data-parallel SGD in SSP[Ho *et al.*, 2013].

We developed a distributed implementation of FDML in a parameter server architecture, and conducted experiments based on both a public data set a9a[Dheeru and Karra Taniskidou, 2017], and a large dataset of $5,000,000$ samples and 8700 decentralized features collected from three popular Tencent Apps, including *Tencent MyApp*, *Tencent QQ Browser* and *Tencent Mobile Safeguard*. Extensive experimental results have demonstrated that FDML can even closely approach centralized learning in terms of testing errors, without violating data locality constraints, although centralized learning can use a more sophisticated model, since all features are collected centrally. In the meantime, FDML significantly outperforms models trained only based on the local features of each single app, demonstrating its advantage in harvesting insights from additional cross-domain features.

2.2 Problem Formulation

Consider a system of m different parties, each party holding different aspects about the same training samples. Let $\{(\xi_i^1, \xi_i^2, \ldots, \xi_i^m), y_i\}_{i=1}^n$ represent the set of n training samples, where the vector $\xi_i^j \in \mathbb{R}^{dj}$ denotes the features of the ith sample located on jth party, and y_i is the label of sample i. Let $\xi_i \in \mathbb{R}^d$ be the overall feature vector of sample i, which is a concatenation of the vectors $\xi_i^1, \xi_i^2, \ldots, \xi_i^m$, with $d = \sum_j d_j$. Suppose the parties are not allowed to transfer their respective feature vector to each other out of regulatory and privacy reasons as has been mentioned above. In our problem, the feature vectors on two parties may or may not contain overlapped features. The goal of machine learning is to find a model $p(x, \xi)$ with parameters x that given an input ξ, can predict its label y, by minimizing the loss between the model prediction $p(x, \xi_i)$ and its corresponding label y_i over all training samples i.

We propose a Feature Distributed Machine Learning (FDML) algorithm that can train a joint model by utilizing all the distributed features while keeping the raw features at each party unrevealed to other parties. To achieve this goal, we adopt a specific class of model that has the form

$$p(x, \xi) = \sigma\bigg(\sum_{j=1}^{m} a_j \alpha^j(x^j, \xi^j)\bigg), \tag{2.1}$$

where $\alpha^j : \mathbb{R}^{D^j} \times \mathbb{R}^{d^j} \to \mathbb{R}$, $j = 1, \ldots, m$, is a sub-model on party j with parameters $x^j \in \mathbb{R}^{D^j}$, which can be a general function that maps the local features ξ^j on each party j to a local prediction. In addition, $\sigma : \mathbb{R} \to \mathbb{R}$ is a continuously differentiable function to aggregate local intermediate predictions $\alpha^j(x^j, \xi^j)$ weighted by a_j. Note that $x \in \mathbb{R}^D$, with $D = \sum_j D_j$, is a concatenation of the local model parameters x^j over all parties j.

As illustrated by Fig. 2.1, the model adopted here is essentially a composite model, where each sub-model α^j on party j with parameters x^j could be an arbitrary model, e.g., logistic regression, SVM, deep neural networks, factorization machines, etc. Each sub-model x^j on party j is only concerned with the local features ξ^j. The final prediction is made by merging the local intermediate results through a linear followed by nonlinear transformations, e.g., a softmax function. Note that in (2.1), all a_j can be eliminated by scaling some corresponding parameters in $\alpha(x^j, \xi^j)$ by $1/a_j$. Without loss of generality, we simplify the model to the following:

$$p(x, \xi) = \sigma\bigg(\sum_{j=1}^{m} \alpha^j(x^j, \xi^j)\bigg). \tag{2.2}$$

Apparently, in this model, both the local features ξ^j and the sub-model parameters x^j are stored and processed locally within party j, while only the local predictions $\alpha^j(x^j, \xi^j)$ need be shared to produce the final prediction. Therefore, the raw features as well as all sub-model parameters are kept private. In Sec. 2.3, we propose an asynchronous SGD algorithm that also preserves the non-sharing properties for all the local features as well as all sub-model parameters even during the model training phase, with theoretical convergence guarantees.

In general, the model is trained by solving the following problem:

$$\text{minimize}_x \frac{1}{n} \sum_{i=1}^{n} L(x; \xi_i, y_i) + \lambda \sum_{j=1}^{m} z^j(x^j), \tag{2.3}$$

where $L\big(p(x, \xi); y\big)$ is the loss function, indicating the gap between the predicted value and the true label for each sample. $z(x^j)$ is the regularizer for sub-model x^j.

2.3 Asynchronous SGD for FDML

In this chapter, we describe our asynchronous and distributed stochastic gradient descent (SGD) algorithm specifically designed to solve the optimization problem (2.3) in FDML, with theoretical convergence guarantees.

Since we consider a stochastic algorithm, let $i(t)$ be the index of the sample $\xi_{i(t)}$ presented to the training algorithm in iteration t. We denote the regularized loss of sample $i(t)$ by

$$F_t(x) := L(x; \xi_{i(t)}, y_{i(t)}) + \lambda \sum_{j=1}^{m} z^j(x^j). \tag{2.4}$$

In stochastic optimization, minimizing the loss in (2.3) over the entire training set is equivalently to solving the following problem[Ho *et al.*, 2013]:

$$\text{minimize}_x F(x) := \frac{1}{T} \sum_{t} F_t(x), \tag{2.5}$$

where T is the total number of iterations. Let $\nabla F(x) \in \mathbb{R}^D$ be the gradient of F. Let $\nabla^j F(x) \in \mathbb{R}^{D^j}$ be the partial gradient of F with respect to the sub-model parameters $x^j \in \mathbb{R}^{D^j}$, i.e., $\nabla^j F(x) := \frac{\partial F(x)}{\partial x^j}$. Clearly, $\nabla F(x)$ is the concatenation of all the partial gradients $\nabla^1 F(x), \nabla^2 F(x), \ldots, \nabla^m F(x)$.

2.3.1 The Synchronous Algorithm

In a synchronous setting, we can simply parallelizing a SGD algorithm by updating each parameter block x^j concurrently for all $j = 1, \ldots, m$, given a coming sample $i(t)$, i.e., $x^j_{t+1} := x^j_t - \eta_t \nabla^j F_t(x^1_t, \ldots, x^m_t)$, where η_t is a predefined learning rate scheme. Specifically for model (2.2), according to (2.4), we can obtain the partial gradient $\nabla^j F_t(x)$ for $j = 1, \ldots, m$ as

$$\begin{aligned} \nabla^j F_t(x) &= \lambda \frac{\partial z^j(x^j)}{\partial x^j} + \\ & L'\Bigg(\sigma\Bigg(\sum_{k=1}^{m} \alpha^k(x^k, \xi^k_{i(t)})\Bigg)\Bigg)\sigma'\Big(\sum_{k=1}^{m} \alpha^k(x^k, \xi^k_{i(t)})\Big)\frac{\partial \alpha^j(x^j, \xi^j_{i(t)})}{\partial x^j} \qquad (2.6) \\ &:= H\Bigg(\sum_{k=1}^{m} \alpha^k(x^k, \xi^k_{i(t)})\Bigg)\frac{\partial \alpha^j(x^j, \xi^j_{i(t)})}{\partial x^j} + \lambda \frac{\partial z^j(x^j)}{\partial x^j}, \qquad (2.7) \end{aligned}$$

where we simplify the notation of the first few terms related to $\sum_{k=1}^m \alpha^k(x^k, \xi^k_{i(t)})$ by a function $H(\cdot)$. In practice, z^j could be non-smooth. This setting is usually handled by proximal methods. In this work, we are only focused on the smooth case.

This indicates that for the class of models in (2.2) adopted by FDML, each party j does not even need other parties' models x^k, where $k \neq j$, to compute its partial gradient $\nabla^j F_t$. Instead, to compute $\nabla^j F_t$ in (2.7), each party j only needs one term, $\sum_{k=1}^m \alpha^k(x^k, \xi^k_{i(t)})$, which is the aggregation of the local prediction results from all parties at iteration t, while the remaining terms in (2.7) is only concerned with party j's local model x^j and local features $\xi^j_{i(t)}$. Therefore, this specific property enables a parallel algorithm with minimum sharing among parties, where neither local features nor local model parameters need be passed among parties.

2.3.2 The Asynchronous Algorithm

The *asynchronous* implementation of this idea in a distributed setting of multiple parties, with theoretical convergence guarantees, is significantly more challenging than it seems. As our proposed algorithm is closely related to asynchronous SGD, yet extends it from the data-parallel setting[Ho *et al.*, 2013] to a block-wise model parallel setting, we would call our algorithm Asynchronous SGD for FDML.

Note that in an asynchronous setting, each party j will update its own parameters x^j_t asynchronously and two parties may be in different iterations. However, we assume different parties go through the samples $\xi_{i(t)}$ in the same order, although asynchronously, i.e., all the parties share the randomly generated sample index sequence $\{i(t)|t = 1, \ldots, T\}$, which can easily be realized by sharing the seed of a pseudo random number generator.

When each party j has its own iteration t, the local model parameters x^j_t on party j is updated by

$$x^j_{t+1} = x^j_t - \eta_t \Bigg(H\Bigg(\sum_{k=1}^{m} \alpha^k(x^k_{t-\tau^j_t(k)}, \xi^k_{i(t)}) \Bigg) \frac{\partial \alpha^j(x^j_t, \xi^j_{i(t)})}{\partial x^j} + \lambda \frac{\partial z^j(x^j_t)}{\partial x^j} \Bigg), \tag{2.8}$$

where the requested aggregation of local predictions for sample $\xi_{i(t)}$ may be computed from possibly *stale versions* of model parameters, $x^k_{t-\tau^j_t(k)}$ on other parties $k \neq j$, where $\tau^j_t(k)$ represents how many iterations of a "lag" there are from party k to party j at the tth iteration of party j. We abuse the word "lag" here since party k could be faster than party j. We overflow the notation for that case by assigning negative value to $\tau^j_t(k)$. We give a convergence speed guarantee of the proposed algorithm under certain assumptions, when the lag $\tau^j_t(k)$ is bounded.

2.4 Distributed Implementation

2.4.1 Implementation

We present a distributed implementation of the proposed asynchronous SGD algorithm for FDML. Our implementation is inspired by the Parameter Server architecture[Li *et al.*, 2014*b*; Li *et al.*, 2014*a*; Chilimbi *et al.*, 2014]. In a typical Parameter Server system, the workers compute gradients while the server updates the model parameters with the gradients computed by workers. Yet, in our implementation, as described in Algorithm 1, the only job of the server is to maintain and update a matrix $A_{i,j}$, $i = 1, \ldots, n$, $j = 1, \ldots, m$, which is introduced to hold the latest m local predictions for each sample i. We call $[A_{i,j}]_{n\times m}$ the *local prediction matrix*. On the other hand, unlike servers, the workers in our system each represent a participating party. They do not only compute gradients, but also need to update their respective local model parameters with SGD.

Furthermore, since each worker performs local updates individually, each worker can even further employ a parameter server cluster or a shared-memory system, e.g., a CPU/GPU workstation, to scale up and parallelize the computation workload related to any local model it adopts, e.g., a DNN or FM. A similar hierarchical cluster is considered in Gaia[Hsieh *et al.*, 2017], though for data-parallel machine learning among multiple data centers.

Algorithm 1 A Distributed Implementation of FDML

```
Require: each worker $j$ holds the local feature set $\{\xi_i^j, y_i\}_{i=1}^n$, $j = 1, \dots, m$; a sample
    presentation schedule $i(t)$, $t = 1, \dots, T$, is pre-generated randomly and shared among
    workers.
Ensure: model parameters $x_T = (x_T^1, \dots, x_T^m)$.
 1: Server:..................................................................
 2: Initialize the local prediction matrix $[A_{i,j}]_{n \times m}$.
 3: while True do
 4:     if Pull request (worker: $j$, iteration: $t$) received then
 5:         if $t$ is not $\tau$ iterations ahead of the slowest worker then
 6:             Send $\sum_{k=1}^m A_{i(t),k}$ to Worker $j$
 7:         else
 8:             Reject the Pull request
 9:     if Push request (worker: $j$, iteration: $t$, value: $c$) received then
10:         $A_{i(t),j} := c$.
11: Worker $j$ ($j = 1, \dots, m$) asynchronously performs:.....................
12: for $t = 1, \dots, T$ do
13:     Push $c := \alpha^j(x_t^j, \xi_{i(t)}^j)$ to Server
14:     while Pull not successful do
15:         Pull $\sum_{k=1}^m A_{i(t),k}$ from Server
16:     $\nabla^j F_t := \left( H(\sum_{k=1}^m A_{i(t),k}) \cdot \frac{\partial \alpha^j(x_t^j, \xi_{i(t)}^j)}{\partial x^j} + \lambda \frac{\partial z^j(x_t^j)}{\partial x^j} \right)$
17:     Update the local weights as
                $x_{t+1}^j := x_t^j - \eta_t \nabla^j F_t.$          (2.9)
```

First, we describe how the input data should be prepared for the FDML system. Before the training task, for consistency and efficiency, a *sample coordinator* will first randomly shuffle the sample indices and generate the sample presentation schedule $i(t)$, which dictates the order in which samples should be presented to the training algorithm. However, since features of a same sample are located on multiple parties, we need to find all the local features $\xi_i^1, \xi_i^2, \dots, \xi_i^m$ as well as the label y_i associated with sample i. This can be done by using some common identifiers that are present in all local features of a sample, like user IDs, phone numbers, data of birth plus name, item IDs, etc. Finally, the labels y_i will be sent to all workers (parties) so that they can compute error gradients locally. Therefore, before the algorithm starts, each worker j holds a local dataset $\{\xi_i^j, y_i\}_{i=1}^n$, for all $j = 1, \dots, m$.

Let us explain Algorithm 1 from a worker's perspective.

To solve for x collaboratively, each worker j goes through the iterations $t = 1, \dots, T$ individually and asynchronously in parallel, according to the (same) predefined sample presentation schedule $i(t)$ and updates its local model x^j according to (2.9). In a particular iteration t, when worker j updates x_t^j with the current local features $\xi_{i(t)}^j$, it first sends its updated local prediction about sample $i(t)$ to the server in order to update $A_{i(t),j}$, i.e., $A_{i(t),j} := \alpha^j(x_t^j, \xi_{i(t)}^j)$. And this update is done through the value c uploaded to the server in a *Push request* from worker j with iteration index t and value c. After this update, it pulls the latest $\sum_{k=1}^m A_{i(t),k}$ from the server based on the latest versions of local predictions, $A_{i(t),k}$, maintained on the server for all the workers $k = 1, \dots, m$. Then x_t^j is updated into x_{t+1}^j locally by (2.9).

Since the workers perform local model updates asynchronously, at a certain point, different workers might be in different iterations, and a faster worker may be using the stale local predictions from other workers. We adopt a *stale synchronous* protocol to strike a balance between the evaluation time for each iteration and the total number of iterations to converge—a fully synchronous algorithm takes the least number of iterations to converge yet incurs large waiting time per iteration due to straggler workers, while on the other hand, an asynchronous algorithm reduced the per iteration evaluation time, at the possible cost of more iterations to converge. In order to reduce the overall training time, we require that the iteration of the fastest party should not exceed the iteration of the slowest party by τ, i.e., the server will reject a pull request if the t from the *Pull request*(worker: j, iteration: t) is τ iterations ahead of the slowest worker in the system. A similar bounded delay condition is enforced in most Parameter-Server-like systems[Li *et al.*, 2014*b*; Chilimbi *et al.*, 2014; Li *et al.*, 2016; Abadi *et al.*, 2016*b*; Hsieh *et al.*, 2017] to ensure convergence and avoid chaotic behavior of a completely asynchronous system.

In real applications, the SGD algorithm can easily be replaced with the mini-batched SGD, by replacing the sample presentation schedule $i(t)$ with a set $I(t)$ representing the indices of a mini-batch of samples to be used iteration t, and replacing the partial gradient in (2.8) with the sum of partial gradients over the mini-batch $I(t)$.

2.4.2 Privacy

In FDML, one of the primary concerns is to preserve the privacy of the local feature data. Due to the specific model structure and the well designed algorithm, no model weights or features are uploaded from any parties. The only shared information is the intermediate local prediction results for each training or testing sample, which is some comprehensive function over both the local features and model weights. Therefore, there is little chance to leak the original features to honest servers or other parties.

To further protect the feature data at each party from malicious servers and parties, we apply differential privacy based methods by perturbing the local predictions to be uploaded[Dwork, 2008; Dwork *et al.*, 2014; Shokri and Shmatikov, 2015]. In particular, we add some noise to the local prediction result $\alpha^j(x_t^j, \xi_{i(t)}^j)$ at party j to protect the privacy of all the input features at party j.

2.5 Convergence Analysis

Inspired by a series of studies[Langford *et al.*, 2009; Ho *et al.*, 2013; Hsieh *et al.*, 2017] on the convergence behavior of convex objective functions, we analyze the convergence property of the proposed asynchronous algorithm by evaluating a *regret* function R, which is the difference between the aggregated training loss and the loss of the optimal solution, defined as

$$R = \frac{1}{T} \sum_t F_t(x_t) - F(x_*), \tag{2.10}$$

where x_* is the optimal solution for $F(x)$, such that $x_* = \text{argmin}_x F(x)$. During training, the same set of data will be looped through for several *epochs*. This is as if a very large dataset is gone through till Tth iteration. We will prove convergence by showing that R will decrease to 0 with regard to T. Before presenting the main result, we introduce several notations and assumptions. We use D_t to denote the distance measure from x_t to x_*, i.e., $D_t := \frac{1}{2}\|x_t - x_*\|_2^2$. We make the following common assumptions on the loss function, which are used in many related studies as well.

Assumption 1 *1. The function F_t is differentiable and the partial gradient $\nabla^j f$ are Lipschitz continuous with L_j, namely,*

$$\|\nabla^j F_t(x_1) - \nabla^j F_t(x_2)\| \leq L_j \|x_1 - x_2\|, \tag{2.11}$$

for $\forall x_1, x_2 \in \mathbb{R}^D$. We denote L_{max} as the maximum among the L_j for $\forall j$.

2. Convexity of the loss function $F_t(x)$.

3. Bounded solution space. There exists a $D > 0$, s.t., $D_t \leq \frac{1}{2}D^2$ for $\forall t$.

As a consequence of the assumptions, the gradients are bounded, i.e., $\exists G > 0$, s.t., $\|\nabla F(x)\|_2^2 \leq G^2$. for $\forall x \in \mathbb{R}^D$ With these assumptions, we come to our main result on the convergence rate of the proposed SGD algorithm.

Proposition 1 *Under circumstances of the assumptions in Assumption 1, with a learning rate of $\eta_t = \frac{\eta}{\sqrt{t}}$, and a bounded staleness of τ, the regret R given by the updates (2.8) for the FDML problem is $R = O(\frac{1}{\sqrt{T}})$.*

Proof. Please refer to Appendix for the proof.

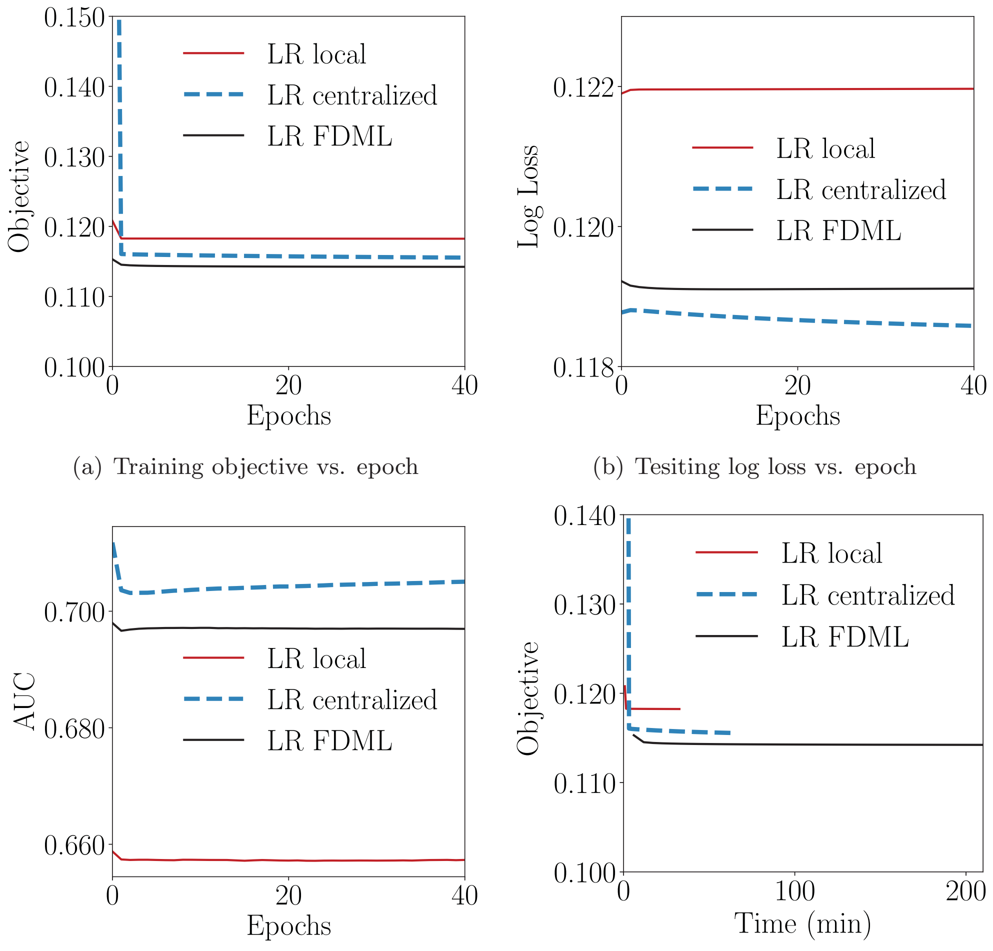


Figure 2.2: A comparison between the three model training schemes for the LR model. All curves are plotted for epochs 1–40, including the time curve in (d).

2.6 Experiments

We are testing the application of the proposed FDML system in an app recommendation task at Tencent MyApp, which is a major Android market with an extremely large body of users in China. In this task, *user features*, including the past download activities in MyApp, are recorded. In the meantime, the task can also benefit from cross-domain features about the same users logged in two other apps (run by different departments of the same company), including QQ Browser that tracks user interests based on their content viewing history, as well as Tencent Mobile Safeguard, which records the app invoking and usage history of users.

The goal here is to leverage the additional user features available from the other domains to improve the app recommendation in MyApp, yet without having to download the raw user features from other apps to avoid regulatory issues, as customer data in different

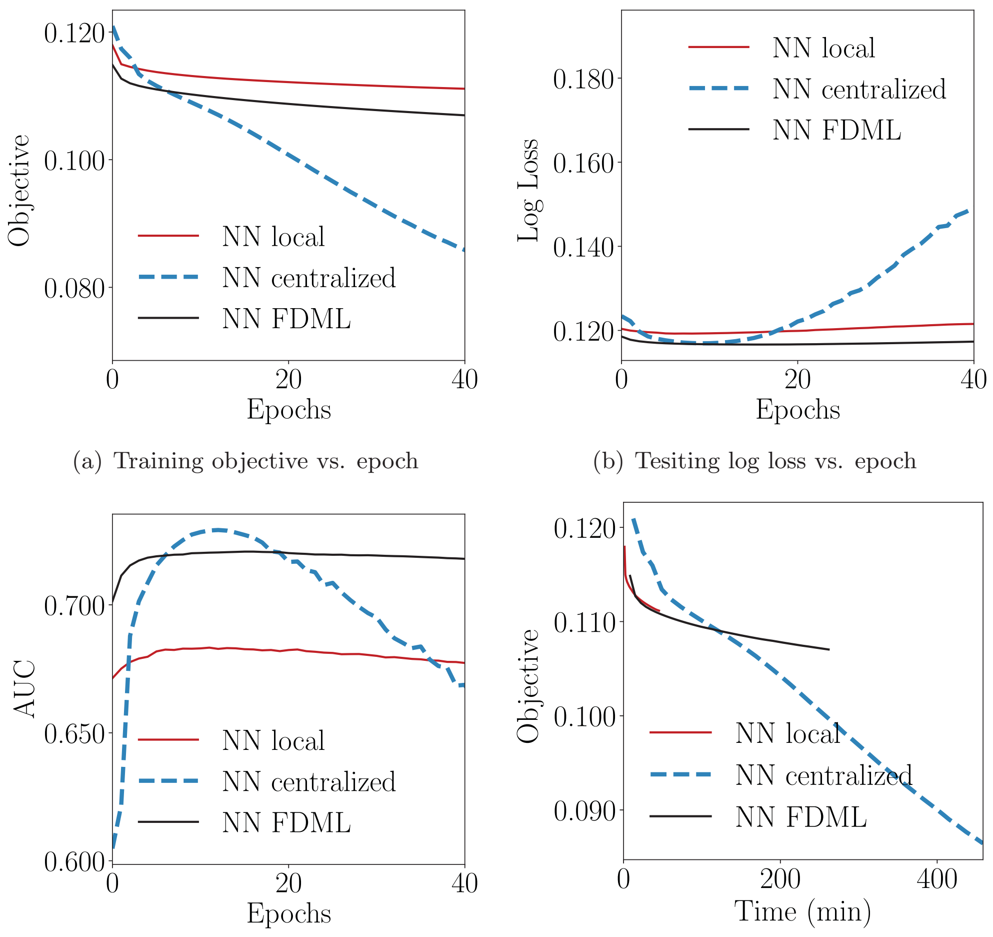
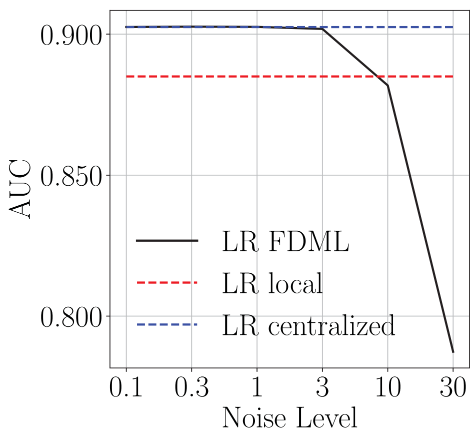


Figure 2.3: A comparison between the three model training schemes for the NN model. All curves are plotted for epochs 1–40, including the time curve in (d).

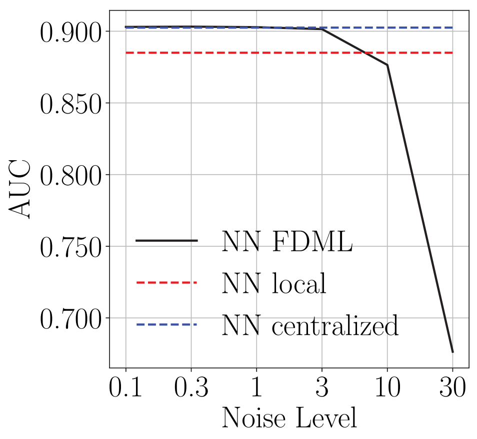
departments are protected under different security levels and even under different customer agreements. Some sensitive features under strong protection are prohibited to be moved to other parties, including other departments.

The dataset we use contains $5,000,000$ labeled samples indicating whether a user will download an app or not. Each sample is a user-app pair, which contains around $8,700$ (sparse) features in total, among which around $7,000$ features come from Tencent MyApp itself, while the remaining $1,700$ features are from the other two apps. We randomly shuffle the data and split it into a 4.5 million training set and a 0.5 million testing set.

We also evaluate FDML on another public data set `a9a`[Dheeru and Karra Taniskidou, 2017], a classical census dataset, where the prediction task is to determine whether a person makes over \$50K a year. There are $48,842$ samples, each with 124 features. $32,661$ samples are training data and $16,281$ samples are testing data. We split the 124 features into two sets of 67 and 57. We run both a logistic regression (LR) and a two layered fully connected



(a) Test AUC vs. added noise for LR



(b) Test AUC vs. added noise for NN

Figure 2.4: Testing AUC under different levels of added noise during training for `a9a` data set

Table 2.1: The performance on Tencent MyApp data.

Algorithm	Train loss	Test loss	Test AUC	Time(s)
LR local	0.1183	0.1220	0.6573	546
LR centralized	0.1159	**0.1187**	**0.7037**	1063
LR FDML	**0.1143**	0.1191	0.6971	3530
NN local	0.1130	0.1193	0.6830	784
NN centralized	**0.1083**	0.1170	**0.7284**	8051
NN FDML	0.1101	**0.1167**	0.7203	4369

neural network (NN) under three different training schemes for both data sets:

- *Local*: only use the 7,000 local features from MyApp or the 67 features of `a9a` to train a model.
- *Centralized*: collect all the 8,700 features from all three apps to a central server or using all the 124 features in `a9a` and train the model using the standard mini-batched SGD.
- *FDML*: use FDML system to train a joint model for app recommendation based on all 8,700 features distributed in three apps or train the `a9a` classification model on all 124 features from two different parties, without centrally collecting data.

For FDML, there is a single server with several workers, each of which equipped with an Intel Xeon CPU E5-2670 v3 @ 2.30GHz. Each worker handles the features from one party. The system will be asynchronous as the lengths of features handled by each worker are different. The FDML NN only considers a fully connected NN within each party while merging the three local predictions in a composite model, whereas the Centralized NN uses a fully connected neural network over all the 8,700124 features, thus leading to a more

Table 2.2: The performance on a9a data.

Algorithm	Train loss	Test loss	Test AUC	Time(s)
LR local	0.3625	0.3509	0.8850	41
LR centralized	0.3359	0.3247	0.9025	45
LR FDML	**0.3352**	**0.3246**	**0.9026**	99
NN local	**0.3652**	0.3484	0.8864	53
NN centralized	0.4008	**0.3235**	**0.9042**	57
NN FDML	0.4170	0.3272	0.9035	110

complex model (with interactions between the local features of different departments) than FDML NN.

For all training schemes, a mini-batched SGD is used with a batch size of 100. For each epoch, we keep track of the optimization objective value for training data, the log loss, the AUC for testing data and the elapsed time. Fig. 2.2 and Fig. 2.3 present the major statistics of the models during the training procedure for LR and NN for Tencent MyApp dataset, respectively. Table 2.1 presents the detailed statistics at the epoch when all the algorithms yield a stable and good performance on the testing data. Table 2.2 presents the performance for `a9a` dataset. The results show that FDML outperforms the corresponding Local scheme with only local features, and even approaches the performance of the Centralized scheme, while keeping the feature sets local to their respective workers.

For LR, as shown by Fig. 2.2, Table 2.1 and Table 2.2, we can see that Centralized LR and FDML LR both achieve a smaller training objective value as well as significantly better performance on the testing set than Local LR. As we have expected, additional features recorded by other related services could indeed help improve the app recommendation performance. Furthermore, Centralizd LR and FDML LR have very close performance, since these two methods use the essentially the same model for LR, though with different training algorithms.

For NN shown in Fig. 2.3, Table 2.1 and Table 2.2, by leveraging additional features, both FDML NN and Centralized NN substantially outperform Local NN. Meanwhile, Centralized NN is slightly better than FDML NN, since Centralized NN has essentially adopted a more complex model, enabling feature interaction between different parties directly through fully connected neural networks.

Fig. 2.2(d) and Fig. 2.3(d) compare the training time and speed among the three learning schemes for Tencent MyApp dataset. Without surprise, for both the LR and NN model, the Local scheme is the fastest since it uses the smallest amount of features and has no communication or synchronization overhead. For LR in Fig. 2.2(d), FDML LR is slower than Centralized LR since the computation load is relatively smaller in this LR model and thus the communication overhead dominates. On the contrary, for NN, as shown in Fig. 2.3(d), the Centralized NN is slower than FDML NN. This is because Centralized NN has much more inner connections and hence much more model parameters to train. Another reason is that FDML distributes the heavy computation load in this NN scenario to three different workers, which in fact speeds up training. Interestingly, for the smaller dataset `a9a`, in Table 2.2, the NN FDML is slower than the centralized one since in this case, the model is small and the communication overhead dominate the processing time in FDML.

Fig. 2.4 shows the performance when different levels of noise is added according to the differential privacy mechanism during the training procedure for `a9a` dataset. Conforming to the intuition, we can see that a higher level of noise will bring worse results. However, for a noise level no more than 3, we can still expect a performance improvement over learning only based on local data, while achieving stronger privacy guarantee due to the perturbations introduced to the shared local prediction results.

2.7 Related Work

Distributed Machine Learning. Distributed machine learning algorithms and systems have been extensively studied in recent years to scale up machine learning in the presence of big data and big models. Existing work focuses either on the theoretical convergence speed of proposed algorithms, or on the practical system aspects to reduce the overall model training time[Xing *et al.*, 2016]. Bulk synchronous parallel algorithms (BSP)[Dekel *et al.*, 2012; Zinkevich *et al.*, 2010] are among the first distributed machine learning algorithms. Due to the hash constraints on the computation and communication procedures, these schemes share a convergence speed that is similar to traditional synchronous and centralized gradient-like algorithms. Stale synchronous parallel (SSP) algorithms[Ho *et al.*, 2013] are a more practical alternative that abandons strict iteration barriers, and allows the workers to be off synchrony up to a certain bounded delay. The convergence results have been developed for both gradient descent and SGD[Recht *et al.*, 2011; Ho *et al.*, 2013; Lian *et al.*, 2015] as well as proximal gradient methods[Li *et al.*, 2014*a*] under different assumptions of the loss functions. In fact, SSP has become central to various types of current distributed Parameter Server architectures[Li *et al.*, 2014*b*; Chilimbi *et al.*, 2014; Li *et al.*, 2016; Abadi *et al.*, 2016*b*; Hsieh *et al.*, 2017].

Depending on how the computation workload is partitioned[Xing *et al.*, 2016], distributed machine learning systems can be categorized into *data parallel* and *model parallel* systems. Most of existing distributed machine learning systems[Li *et al.*, 2014*b*; Chilimbi *et al.*, 2014; Li *et al.*, 2016; Abadi *et al.*, 2016*b*; Hsieh *et al.*, 2017] fall into the range of *data parallel*, where different workers hold different training samples.

Model Parallelism. There are only a couple of studies on *model parallel* systems, i.e., DistBelief[Dean *et al.*, 2012] and STRADS[Lee *et al.*, 2014], which aims to train a big model by letting each worker be responsible for updating a subset of model parameters. However, both DistBelief and STRADS, require collaborating workers to transmit their local model parameters to each other (or to a server), which violates our non-leakage requirement for models and inevitably incurs more transmission overhead. Furthermore, nearly all recent advances on model parallel neural networks (e.g., DistBelief[Dean *et al.*, 2012] and AMPNet[Ben-Nun and Hoefler, 2018]) mainly partition the network horizontally according to neural network layers with motivation to scale up computation to big models. In contrast, we study a completely vertical partition strategy based strictly on features, which is motivated by the cooperation between multiple businesses/organizations that hold different aspects of information about the same samples. Another difference is that we do not require transmitting the model parameters; nor any raw feature data between parties.

On a theoretical perspective of model parallel algorithm analysis,[Zhou *et al.*, 2016] has proposed and analyzed the convergence of a model parallel yet non-stochastic *proximal gradient* algorithm that requires passing model parameters between workers under the SSP setting. Parallel coordinate descent algorithms have been analyzed recently in[Bradley *et al.*, 2011; Scherrer *et al.*, 2012]. Yet, these studies focus on randomized coordinate selection in a *synchronous* setting, which is different from our setting where multiple nodes can update disjoint model blocks asynchronously. Although Stochastic gradient descent (SGD) is the most popular optimization method extensively used for modern distributed data analytics and machine learning, to the best of our knowledge, there is still no convergence result of (asynchronous) SGD in a model parallel setting to date. Our convergence rate of FDML offers the first analysis of asynchronous *model parallel* SGD, which matches the standard convergence rate of the original SSP algorithm[Ho *et al.*, 2013] for data parallel SGD.

Learning Privately. A variant of distributed SGD with a filter to suppress insignificant updates has recently been applied to collaborative deep learning among multiple parties in a data parallel fashion[Shokri and Shmatikov, 2015]. Although raw data are not transferred by the distributed SGD in[Shokri and Shmatikov, 2015], a recent study[Hitaj *et al.*, 2017] points out that an algorithm that passes model parameters may be vulnerable to model inversion attacks based on generative adversarial networks (GANs). In contrast, we do not let parties transfer local model parameters to server or any other party.

Aside from the distributed optimization approach mentioned above, another approach to privacy preserving machine learning is through encryption, e.g., via homomorphic encryption[Gilad-Bachrach *et al.*, 2016; Takabi *et al.*, 2016] or secret sharing[Mohassel and Zhang, 2017; Wan *et al.*, 2007; Bonte and Vercauteren, 2018]. Models are then trained on encrypted data. However, this approach cannot be flexibly generalized to all algorithms and operations, and incurs additional computation and design cost. Relatively earlier, differential privacy has also been applied to collaborative machine learning[Pathak *et al.*, 2010; Rajkumar and Agarwal, 2012], with an inherent tradeoff between privacy and utility of the trained model. To the best of our knowledge, none of the previous work addressed the problem of collaborative learning when the features of each training sample are distributed on multiple participants.

2.8 Summary

We study a feature distributed machine learning (FDML) problem motivated by real-world recommender applications at Tencent MyApp, where the features about the same training sample can be found at three different apps. However, the features of one app should be kept confidential to other parties due to regulatory constraints. This motivation is in contrast to most existing literature on collaborative and distributed machine learning which assumes the data samples (but not the features) are distributed and works in a data-parallel fashion. We propose an asynchronous SGD algorithm to solve the new FDML scenario, with a convergence rate of $O(1/\sqrt{T})$, T being the total number of iterations, matching the existing convergence rate known for data-parallel SGD in a stale synchronous parallel setting[Ho *et al.*, 2013].

We have developed a distributed implementation of the FDML system in a parameter server architecture and performed extensive evaluation based on both a public data set and a large dataset of $5,000,000$ records and $8,700$ decentralized features from *Tencent MyApp*, *Tencent QQ Browser* and *Tencent Mobile Safeguard* for a realistic app recommendation task. Results have shown that FDML can closely approximate centralized training (the latter collecting all data centrally and using a more complex model allowing more interactions among cross-domain features) in terms of the testing AUC, while significantly outperforming the models trained only based on the local features of *MyApp*. Currently, we are deploying the FDML system at *Tencent MyApp* and improving the robustness of the system by adding momentum based techniques. We are also developing schemes that can support more sophisicated models, taking more interactions between cross-party features into account.

Chapter 3

ADMM Sharing Based Method to Distributed Machine Learning over Distriubted Features

3.1 Introduction

In Chapter 2, we study the FDML problem, a collaborative machine learning problem for distributed features, where multiple parties may possess different features about a same sample, yet do not wish to share these features with each other. An SGD based algorithm is proposed to train a special family of composition models. However, as we will show in the experiment in this chapter, the convergence rate is limited for SGD based algorithm when the number of features is relatively large. What's more, larger number of epochs may dramatically increase the risk of leaking sensitive information since more intermediate results have to be transmitted in this situation.

In this chapter, we propose an ADMM algorithm to solve the empirical risk minimization (ERM) problem, a general optimization formulation of many machine learning models visited by a number of recent studies on distributed machine learning [Ying *et al.*, 2018; Chaudhuri *et al.*, 2011], which is a special case for FDML problem. We propose an ADMM-sharing-based distributed algorithm to solve ERM, in which each participant does not need to share any raw features or local model parameters to other parties. Instead, each party only transmits a single value for each sample to other parties, thus largely preventing the local features from being disclosed. We establish theoretical convergence guarantees and iteration complexity results under the non-convex loss in a fully parallel setting, whereas previously, the convergence of ADMM sharing algorithm for non-convex losses is only known for the case of sequential (Gauss-Seidel) execution [Hong *et al.*, 2016].

To further provide privacy guarantees, we present a privacy-preserving version of the ADMM sharing algorithm, in which the transmitted value from each party is perturbed by a carefully designed Gaussian noise to achieve the notion of ϵ, δ-differential privacy [Dwork, 2008; Dwork *et al.*, 2014]. For distributed features, the perturbed algorithm ensures that the probability distribution of the values shared is relatively insensitive to any change to a single feature in a party's local dataset.

Experimental results on two realistic datasets suggest that our proposed ADMM sharing algorithm can converge efficiently. Compared to the gradient-based method, our method can scale as the number of features increases and yields robust convergence. The algorithm can also converge with moderate amounts of Gaussian perturbation added, therefore enabling the utilization of features from other parties to improve the local machine learning task.

3.2 Empirical Risk Minimization over Distributed Features

Consider N samples, each with d features distributed on M parties, which do not wish to share data with each other. The entire dataset $\mathcal{D} \in \mathbb{R}^N \times \mathbb{R}^d$ can be viewed as M vertical partitions $\mathcal{D}_1, \dots, \mathcal{D}_M$, where $\mathcal{D}_m \in \mathbb{R}^N \times \mathbb{R}^{d_m}$ denotes the data possessed by the mth party and d_m is the dimension of features on party m. Clearly, $d = \sum_{m=1}^M d_m$. Let $\mathcal{D}^i$ denote the ith row of $\mathcal{D}$, and $\mathcal{D}_m^i$ be the ith row of $\mathcal{D}_m$ $(k = 1, \cdots, N)$. Then, we have

$$\mathcal{D} = \begin{bmatrix} \mathcal{D}_1^1 & \mathcal{D}_2^1 & \cdots & \mathcal{D}_M^1 \\ \mathcal{D}_1^2 & \mathcal{D}_2^2 & \cdots & \mathcal{D}_M^2 \\ \vdots & \vdots & \ddots & \vdots \\ \mathcal{D}_1^N & \mathcal{D}_2^N & \cdots & \mathcal{D}_M^N \end{bmatrix},$$

where $\mathcal{D}_m^i \in \mathcal{A}_m \subset \mathbb{R}^{d_m}$, $(i = 1, \cdots, N, m = 1, \cdots, M)$. Let $Y_i \in \{-1, 1\}^N$ be the label of sample i.

Let $x = (x_1^\top, \cdots, x_m^\top, \cdots, x_M^\top)^\top$ represent the model parameters, where $x_m \in \mathbb{R}^{d_m}$ are the local parameters associated with the mth party. The objective is to find a model $f(\mathcal{D}^i; x)$ with parameters x to minimize the regularized empirical risk, i.e.,

$$\underset{x \in X}{\text{minimize}} \quad \frac{1}{N} \sum_{i=1}^N l_i(f(\mathcal{D}^i; x), Y_i) + \lambda R(x),$$

where $X \subset \mathbb{R}^d$ is a closed convex set and the regularizer $R(\cdot)$ prevents overfitting.

Similar to recent literature on distributed machine learning[Ying *et al.*, 2018; Zhou *et al.*, 2016], ADMM[Zhang and Zhu, 2016; Zhang *et al.*, 2018], and privacy-preserving machine learning[Chaudhuri *et al.*, 2011; Hamm *et al.*, 2016], we assume the loss has a form

$$\sum_{i=1}^N l_i(f(\mathcal{D}^i; x), Y_i) = \sum_{i=1}^N l_i(\mathcal{D}^i x, Y_i) = l\left(\sum_{m=1}^M \mathcal{D}_m^i x_m\right),$$

where we have abused the notation of l and in the second equality absorbed the label Y_i into the loss l, which is possibly a non-convex function. This framework incorporates a wide range of commonly used models including support vector machines, Lasso, logistic regression, boosting, etc.

Therefore, the risk minimization over distributed features, or vertically partitioned datasets $\mathcal{D}_1, \ldots, \mathcal{D}_M$, can be written in the following compact form:

$$\underset{x}{\text{minimize}} \quad l\left(\sum_{m=1}^{M} \mathcal{D}_m x_m\right) + \lambda \sum_{m=1}^{M} R_m(x_m), \tag{3.1}$$

$$\text{subject to} \quad x_m \in X_m, m = 1, \ldots, M, \tag{3.2}$$

where $X_m \subset \mathbb{R}^{d_m}$ is a closed convex set for all m.

We have further assumed the regularizer is separable such that $R(x) = \sum_{m=1}^{M} R_m(x_m)$. This assumption is consistent with our algorithm design philosophy—under vertically partitioned data, we require each party focus on training and regularizing its local model x_m, without sharing any local model parameters or raw features to other parties at all.

3.3 The ADMM Sharing Algorithm

We present an ADMM sharing algorithm[Boyd *et al.*, 2011; Hong *et al.*, 2016] to solve Problem (3.1) and establish a convergence guarantee for the algorithm. Our algorithm requires each party only share a single value to other parties in each iteration, thus requiring the minimum message passing. In particular, Problem (3.1) is equivalent to

$$\underset{x}{\text{minimize}} \quad l\,(z) + \lambda \sum_{m=1}^{M} R_m(x_m), \tag{3.3}$$

$$\text{s.t.} \quad \sum_{m=1}^{M} \mathcal{D}_m x_m - z = 0, \quad x_m \in X_M, m = 1, \ldots, M, \tag{3.4}$$

where z is an auxiliary variable. The corresponding augmented Lagrangian is given by

$$\mathcal{L}(\{x\}, z; y) = l(z) + \lambda \sum_{m=1}^{M} R_m(x_m) + \langle y, \sum_{m=1}^{M} \mathcal{D}_m x_m - z\rangle + \frac{\rho}{2}\| \sum_{m=1}^{M} \mathcal{D}_m x_m - z\|^2, \tag{3.5}$$

where y is the dual variable and ρ is the penalty factor. In the t^{th} iteration of the algorithm, variables are updated according to

$$x_m^{t+1} := \underset{x_m \in X_m}{\text{argmin}} \quad \lambda R_m(x_m) + \langle y^t, \mathcal{D}_m x_m\rangle + \frac{\rho}{2}\Big\| \sum_{k=1,\ k\neq m}^{M} \mathcal{D}_k x_k^t + \mathcal{D}_m x_m - z^t\Big\|^2,$$

$$m = 1, \cdots, M \tag{3.6}$$

$$z^{t+1} := \underset{z}{\text{argmin}} \quad l(z) - \langle y^t, z\rangle + \frac{\rho}{2}\Big\| \sum_{m=1}^{M} \mathcal{D}_m x_m^{t+1} - z\Big\|^2 \tag{3.7}$$

$$y^{t+1} := y^t + \rho\big(\sum_{m=1}^{M} \mathcal{D}_m x_m^{t+1} - z^{t+1}\big). \tag{3.8}$$

Algorithm 2 The ADMM Sharing Algorithm

```
1: ——Each party $m$ performs in parallel:...........................................
2: for $t$ in $1, \dots, T$ do
3:     Pull $\sum_k \mathcal{D}_k x_k^t - z^t$ and $y^t$ from central node
4:     Obtain $\sum_{k \neq m} \mathcal{D}_k x_k^t - z^t$ by subtracting the locally cached $\mathcal{D}_m x_m^t$ from the pulled
   value $\sum_k \mathcal{D}_k x_k^t - z^t$
5:     Compute $x_m^{t+1}$ according to (3.6)
6:     Push $\mathcal{D}_m x_m^{t+1}$ to the central node
7: ——Central node:...........................................................
8: for $t$ in $1, \dots, T$ do
9:     Collect $\mathcal{D}_m x_m^{t+1}$ for all $m = 1, \dots, M$
10:    Compute $z^{t+1}$ according to (3.7)
11:    Compute $y^{t+1}$ according to (3.8)
12:    Distribute $\sum_k \mathcal{D}_k x_k^{t+1} - z^{t+1}$ and $y^{t+1}$ to all the parties.
```

Formally, in a distributed and fully parallel manner, the algorithm is described in Algorithm 2. Note that each party m needs the value $\sum_{k \neq m} \mathcal{D}_k x_k^t - z^t$ to complete the update, and Lines 3, 4 and 12 in Algorithm 2 present a trick to reduce communication overhead. On each local party , (3.6) is computed where a proper x_m is derived to simultaneously minimize the regularizer and bring the global prediction close to z^t, given the local predictions from other parties. When $R_m(\cdot)$ is l_2 norm, (3.6) becomes a trivial quadratic program which can be efficiently solved. On the central node, the global prediction z is found in (3.7) by minimizing the loss $l(\cdot)$ while bringing z close to the aggregated local predictions from all local parties. Therefore, the computational complexity of (3.7) is independent of the number of features, thus making the proposed algorithm scalable to a large number of features, as compared to SGD or Frank-Wolfe algorithms.

3.3.1 Convergence Analysis

We follow Hong et al.[Hong *et al.*, 2016] to establish the convergence guarantee of the proposed algorithm under mild assumptions. Note that[Hong *et al.*, 2016] provides convergence analysis for the Gauss-Seidel version of the ADMM sharing, where $x_1, \dots, x_M$ are updated sequentially, which is not naturally suitable to parallel implementation. In (3.6) of our algorithm, x_m's can be updated by different parties in parallel in each iteration. We establish convergence as well as iteration complexity results for this parallel scenario, which is more realistic in distributed learning. We need the following set of common assumptions.

Assumption 2 *1. There exists a positive constant $L > 0$ such that*

$$\|\nabla l(x) - \nabla l(z)\| \leq L\|x - z\| \quad \forall x, z.$$

Moreover, for all $m \in \{1, 2, \cdots, M\}$, X_m's are closed convex sets; each $\mathcal{D}_m$ is of full column rank so that the minimum eigenvalue $\sigma_{min}(\mathcal{D}_m^\top \mathcal{D}_m)$ of matrix $\mathcal{D}_m^\top \mathcal{D}_m$ is positive.

2. *The penalty parameter ρ is chosen large enough such that*

   (a) *each x_m subproblem* (3.6) *as well as the z subproblem* (3.7) *is strongly convex, with modulus $\{\gamma_m(\rho)\}_{m=1}^M$ and $\gamma(\rho)$, respectively.*

   (b) *$\gamma_m(\rho) \geq 2\sigma_{max}(\mathcal{D}_m^\top \mathcal{D}_m), \forall m$, where $\sigma_{max}(\mathcal{D}_m^\top \mathcal{D}_m)$ is the maximum eigenvalue for matrix $\mathcal{D}_m^\top \mathcal{D}_m$.*

   (c) *$\rho\gamma(\rho) > 2L^2$ and $\rho \geq L$.*

3. *The objective function $l\left(\sum_{m=1}^M \mathcal{D}_m x_m\right) + \lambda \sum_{m=1}^M R_m(x_m)$ in Problem 3.1 is lower bounded over $\Pi_{m=1}^M X_m$ and we denote the lower bound as $\underline{f}$.*

4. *R_m is either smooth nonconvex or convex (possibly nonsmooth). For the former case, there exists $L_m > 0$ such that $\|\nabla R_m(x_m) - \nabla R_m(z_m)\| \leq L_m \|x_m - z_m\|$ for all $x_m, z_m \in X_m$.*

Specifically, 1, 3 and 4 in Assumptions 2 are common settings in the literature. Assumptions 2.2 is achievable if the ρ is chosen large enough.

Denote $\mathcal{M} \subset \{1, 2, \ldots, M\}$ as the index set, such that when $m \in \mathcal{M}$, R_m is convex, otherwise, R_m is nonconvex but smooth. Our convergence results show that under mild assumptions, the iteratively updated variables eventually converge to the set of primal-dual stationary solutions. Theorem 2 formally states this result.

Theorem 2 *Suppose Assumption 2 holds true, we have the following results:*

1. $\lim_{t\to\infty} \| \sum_{m=1}^M \mathcal{D}_m x_m^{t+1} - z^{t+1} \| = 0$.

2. *Any limit point $\{\{x^*\}, z^*; y^*\}$ of the sequence $\{\{x^{t+1}\}, z^{t+1}; y^{t+1}\}$ is a stationary solution of problem* (3.1) *in the sense that*

$$x_m^* \in \underset{x_m \in X_m}{argmin} \quad \lambda R_m(x_m) + \langle y^*, \mathcal{D}_m x_m \rangle, m \in \mathcal{M}, \tag{3.9}$$

$$\langle x_m - x_m^*, \lambda \nabla l(x_m^*) - \mathcal{D}_m^T y^* \rangle \leq 0 \quad \forall x_m \in X_m, m \notin \mathcal{M}, \tag{3.10}$$

$$\nabla l(z^*) - y^* = 0, \tag{3.11}$$

$$\sum_{m=1}^M \mathcal{D}_m x_m^* = z^*. \tag{3.12}$$

3. *If $\mathcal{D}_m$ is a compact set for all m, then $\{\{x_m^t\}, z^t; y^t\}$ converges to the set of stationary solutions of problem* (3.1), *i.e.,*

$$\lim_{t\to\infty} \quad dist\big((\{x^t\}, z^t; y^t); Z^*\big) = 0,$$

where Z^* is the set of primal-dual stationary solutions for problem (3.1).

3.3.2 Iteration Complexity Analysis

We evaluate the iteration complexity over a *Lyapunov function*. More specifically, we define V^t as

$$V^t := \sum_{m=1}^{M} \|\tilde{\nabla}_{x_m}\mathcal{L}(\{x_m^t\}, z^t; y^t)\|^2 + \|\nabla_z \mathcal{L}(\{x_m^t\}, z^t; y^t)\|^2 + \|\sum_{m=1}^{M} \mathcal{D}_m x_m^t - z^t\|^2, \quad (3.13)$$

where

$$\tilde{\nabla}_{x_m}\mathcal{L}(\{x_m^t\}, z^t; y^t) = \nabla_{x_m}\mathcal{L}(\{x_m^t\}, z^t; y^t) \quad \text{when } m \notin \mathcal{M},$$

$$\tilde{\nabla}_{x_m}\mathcal{L}(\{x_m^t\}, z^t; y^t) = x_m^t - \mathrm{prox}_{\lambda R_m}\left[x_m^t - \nabla_{x_m}\big(\mathcal{L}(\{x_m^t\}, z^t; y^t) - \lambda \sum_{m=1}^{M} R_m(x_m^t)\big)\right]$$
$$\text{when } m \in \mathcal{M},$$

with $\mathrm{prox}_h[z] := \mathrm{argmin}_x h(x) + \frac{1}{2}\|x-z\|^2$. It is easy to verify that when $V^t \to 0$, a stationary solution is achieved due to the properties. The result for the iteration complexity is stated in the following theorem, which provides a quantification of how fast our algorithm converges. Theorem 3 shows that the algorithm converges in the sense that the *Lyapunov function* V^t will be less than any $\epsilon > 0$ within $O(1/\epsilon)$ iterations.

Theorem 3 *Suppose Assumption 2 holds. Let $T(\epsilon)$ denote the iteration index in which:*

$$T(\epsilon) := min\{t | V^t \leq \epsilon, t \geq 0\},$$

for any $\epsilon > 0$. Then there exists a constant $C > 0$, such that

$$T(\epsilon)\epsilon \leq C(\mathcal{L}(\{x^1\}, z^1; y^1 - \underline{f}), \quad (3.14)$$

where $\underline{f}$ is the lower bound defined in Assumption 2.

3.4 Differentially Private ADMM Sharing

Differential privacy[Dwork *et al.*, 2014; Zhou *et al.*, 2010] is a notion that ensures a strong guarantee for data privacy. The intuition is to keep the query results from a dataset relatively close if one of the entries in the dataset changes, by adding some well designed random noise into the query, so that little information on the raw data can be inferred from the query. Formally, the definition of differential privacy is given in Definition 4.

Definition 4 *A randomized algorithm $\mathcal{M}$ is $(\varepsilon, \delta)-$differentially private if for all $S \subseteq$ range$(\mathcal{M})$, and for all x and y, such that $|x - y|_1 \leq 1$, we have*

$$Pr(\mathcal{M}(x) \in S) \leq \exp(\varepsilon) Pr(\mathcal{M}(y) \in S) + \delta. \quad (3.15)$$

Definition 4 provides a strong guarantee for privacy, where even if most entries of a dataset are leaked, little information about the remaining data can be inferred from the randomized output. Specifically, when ε is small, $\exp(\varepsilon)$ is approximately $1+\varepsilon$. Here x and y denote two possible instances of some dataset. $|x-y|_1 \leq 1$ means that even if most of the data entries but one are leaked, the difference between the randomized outputs of x and y is at most ε no matter what value the remaining single entry takes, preventing any adversary from inferring the value of that remaining entry. Moreover, δ allows the possibility that the above ε-guarantee may fail with probability δ.

In our ADMM algorithm, the shared messages $\{\mathcal{D}_m x_m^{t+1}\}_{t=0,1,\cdots,T-1}$ may reveal sensitive information from the data entry in D_m of Party m. We perturb the shared value $\mathcal{D}_m x_m^{t+1}$ in Algorithm 2 with a carefully designed random noise to provide differential privacy. The resulted perturbed ADMM sharing algorithm is the following updates:

$$x_m^{t+1} := \underset{x_m \in X_m}{\operatorname{argmin}} \quad \lambda R_m(x_m) + \langle y^t, \mathcal{D}_m x_m \rangle + \frac{\rho}{2} \Big\| \sum_{k=1,\ k \neq m}^{M} \mathcal{D}_k \tilde{x}_k^t + \mathcal{D}_m x_m - z^t \Big\|^2,$$

$$\begin{aligned} \xi_m^{t+1} &:= \mathcal{N}(0, \sigma_{m,t+1}^2 (\mathcal{D}_m^\top \mathcal{D}_m)^{-1}) \\ \tilde{x}_m^{t+1} &:= x_m^{t+1} + \xi_m^{t+1} \\ z^{t+1} &:= \underset{z}{\operatorname{argmin}} \quad l(z) - \langle y^t, z \rangle + \frac{\rho}{2} \Big\| \sum_{m=1}^{M} \mathcal{D}_m \tilde{x}_m^{t+1} - z \Big\|^2 \\ y^{t+1} &:= y^t + \rho \Big(\sum_{m=1}^{M} \mathcal{D}_m \tilde{x}_m^{t+1} - z^{t+1} \Big). \end{aligned} \tag{3.16}$$

In the remaining part of this section, we demonstrate that (3.16) guarantees (ε, δ) differential privacy with outputs $\{\mathcal{D}_m \tilde{x}_m^{t+1}\}_{t=0,1,\cdots,T-1}$ for some carefully selected $\sigma_{m,t+1}$. Beside Assumption 2, we introduce another set of assumptions widely used by the literature.

Assumption 3 *1. The feasible set $\{x, y\}$ and the dual variable z are bounded; their l_2 norms have an upper bound b_1.*

2. The regularizer $R_m(\cdot)$ is doubly differentiable with $|R_m''(\cdot)| \leq c_1$, where c_1 is a finite constant.

3. Each row of $\mathcal{D}_m$ is normalized and has an l_2 norm of 1.

Note that Assumption 3.1 is adopted in[Sarwate and Chaudhuri, 2013] and[Wang *et al.*, 2019]. Assumption 3.2 comes from[Zhang and Zhu, 2017] and Assumption 3.3 comes from[Zhang and Zhu, 2017] and[Sarwate and Chaudhuri, 2013]. As a typical method in differential privacy analysis, we first study the l_2 sensitivity of $\mathcal{D}_m x_m^{t+1}$, which is defined by:

Definition 5 *The l_2-norm sensitivity of $\mathcal{D}_m x_m^{t+1}$ is defined by:*

$$\Delta_{m,2} = \max_{\substack{\mathcal{D}_m, D_m' \\ \|\mathcal{D}_m - D_m'\| \leq 1}} \|\mathcal{D}_m x_{m,\mathcal{D}_m}^{t+1} - \mathcal{D}_m' x_{m,\mathcal{D}_m'}^{t+1}\|.$$

where $\mathcal{D}_m$ and $\mathcal{D}'_m$ are two neighbouring datasets differing in only one feature column, and $x^{t+1}_{m,\mathcal{D}_m}$ is the x^{t+1}_m derived from the first line of equation (3.16) under dataset $\mathcal{D}_m$.

We have Lemma 6 to state the upper bound of the l_2-norm sensitivity of $\mathcal{D}_m x^{t+1}_m$.

Lemma 6 *Assume that Assumption 2 and Assumption 3 hold. Then the l_2-norm sensitivity of $\mathcal{D}_m x^{t+1}_{m,\mathcal{D}_m}$ is upper bounded by $\mathbb{C} = \frac{3}{d_m \rho}[\lambda c_1 + (1 + M\rho)b_1]$.*

We have Theorem 7 for differential privacy guarantee in each iteration.

Theorem 7 *Assume assumptions 3.1-3.3 hold and $\mathbb{C}$ is the upper bound of $\Delta_{m,2}$. Let $\varepsilon \in (0, 1]$ be an arbitrary constant and let $\mathcal{D}_m \xi^{t+1}_m$ be sampled from zero-mean Gaussian distribution with variance $\sigma^2_{m,t+1}$, where*

$$\sigma_{m,t+1} = \frac{\sqrt{2ln(1.25/\delta)}\mathbb{C}}{\varepsilon}.$$

Then each iteration guarantees (ε, δ)-differential privacy. Specifically, for any neighboring datasets $\mathcal{D}_m$ and $\mathcal{D}'_m$, for any output $\mathcal{D}_m \tilde{x}^{t+1}_{m,\mathcal{D}_m}$ and $\mathcal{D}'_m \tilde{x}^{t+1}_{m,\mathcal{D}'_m}$, the following inequality always holds:

$$P(\mathcal{D}_m \tilde{x}^{t+1}_{m,\mathcal{D}_m} | \mathcal{D}_m) \le e^{\varepsilon} P(\mathcal{D}'_m \tilde{x}^{t+1}_{m,\mathcal{D}'_m} | \mathcal{D}'_m) + \delta.$$

With an application of the composition theory in[Dwork *et al.*, 2014], we come to a result stating the overall privacy guarantee for the whole training procedure.

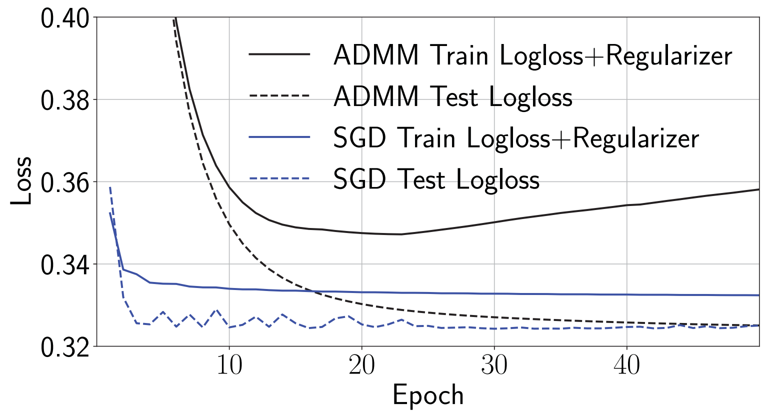
Corollary 8 *For any $\delta' > 0$, the algorithm described in (3.16) satisfies $(\varepsilon', T\delta + \delta')-$differential privacy within T epochs of updates, where*

$$\varepsilon' = \sqrt{2T ln(1/\delta')}\varepsilon + T\varepsilon(e\varepsilon - 1). \tag{3.17}$$

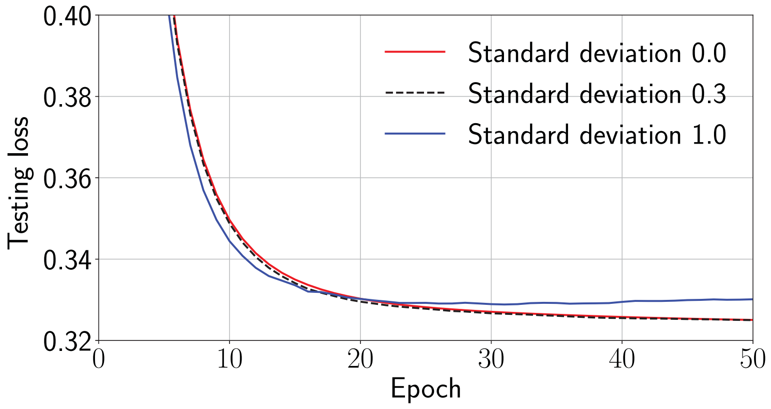
Without surprise, the overall differential privacy guarantee may drop dramatically if the number of epochs T grows to a large value, since the number of exposed results grows linearly in T. However, as we will show in the experiments, the ADMM-sharing algorithm converges fast, taking much fewer epochs to converge than SGD when the number of features is relatively large. Therefore, it is of great advantage to use ADMM sharing for wide features as compared to SGD or Frank-Wolfe algorithms. When T is confined to less than 20, the risk of privacy loss is also confined.

3.5 Experiments

We test our algorithm by training l_2-norm regularized logistic regression on two popular public datasets, namely, *a9a* from UCI[Dua and Graff, 2017] and *giette*[Guyon *et al.*, 2005]. We get the datasets from[*LIBSVM Data: Classification (Binary Class), n.d.*] so that we follow the same preprocessing procedure listed there. *a9a* dataset is 4 MB and contains



(a) Loss vs. epoch



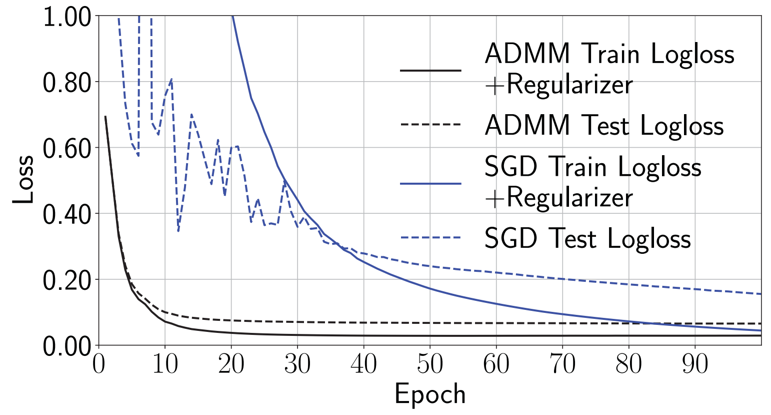
(b) Test log loss under different noise levels

Figure 3.1: Performance over the *a9a* data set with 32561 training samples, 16281 testing samples and 123 features.

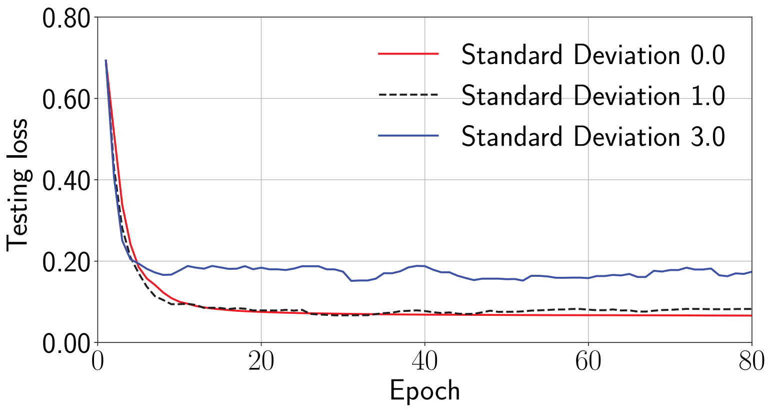
32561 training samples, 16281 testing samples and 123 features. We divide the dataset into two parts, with the first part containing the first 66 features and the second part remaining 57 features. The first part is regarded as the local party who wishes to improve its prediction model with the help of data from the other party. On the other hand, *gisette* dataset is 297 MB and contains 6000 training samples, 1000 testing samples and 5000 features. Similarly, we divide the features into 3 parts, the first 2000 features being the first part regarded as the local data, the next 2000 features being the second part, and the remaining 1000 as the third part. Note that *a9a* is small in terms of the number of features and *gisette* has a relatively higher dimensional feature space.

A prototype system is implemented in *Python* to verify our proposed algorithm. Specifically, we use optimization library from *scipy* to handle the optimization subproblems. We apply L-BFGS-B algorithm to do the x update in (3.6) and entry-wise optimization for z in (3.7). We run the experiment on a machine equipped with Intel(R) Core(TM) i9-9900X CPU @ 3.50GHz and 128 GB of memory.

We compare our algorithm against an SGD based algorithm proposed in[Hu *et al.*, 2019].



(a) Loss vs. epoch

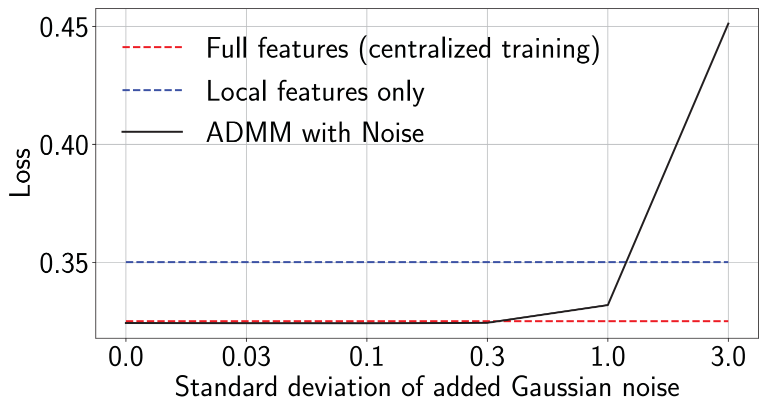


(b) Test log loss under different noise levels

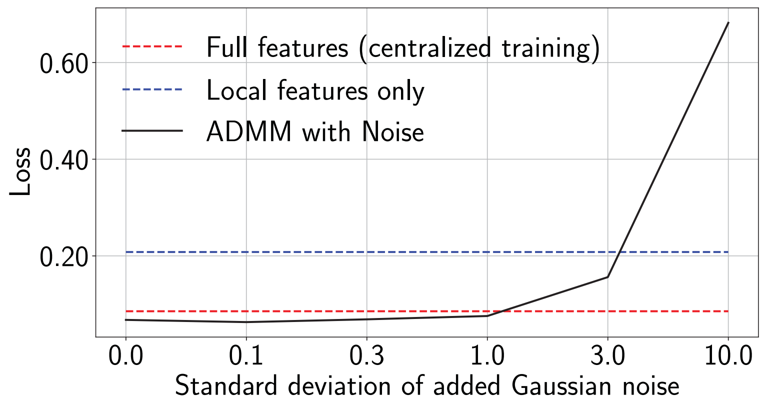
Figure 3.2: Performance over the *gisette* data set with 6000 training samples, 1000 testing samples and 5000 features.

We keep track of the training objective value (log loss plus the l_2 regularizer), the testing log loss for each epoch for different datasets and parameter settings. We also test our algorithm with different levels of Gaussian noise added. In the training procedure, we initialize the elements in x, y and z with 0 while we initialize the parameter for the SGD-based algorithm with random numbers.

Fig. 3.1 and Fig. 3.2 show a typical trace of the training objective and testing log loss against epochs for *a9a* and *gisette*, respectively. On *a9a*, the ADMM algorithm is slightly slower than the SGD based algorithm, while they reach the same testing log loss in the end. On *gisette*, the SGD based algorithm converges slowly while the ADMM algorithm is efficient and robust. The testing log loss from the ADMM algorithm quickly converges to 0.08 after a few epochs, but the SGD based algorithm converges to only 0.1 with much more epochs. This shows that the ADMM algorithm is superior when the number of features is large. In fact, for each epoch, the x update is a trivial quadratic program and can be efficiently solved numerically. The z update contains optimization over computationally expensive functions, but for each sample, it is always an optimization over a single scalar



(a) *a9a* data set



(b) *gisette* data set

Figure 3.3: Test performance for ADMM under different levels of added noise.

so that it can be solved efficiently via scalar optimization and scales with the number of features.

Moreover, Corollary 8 implies that the total differential privacy guarantee will be stronger if the number of epochs required for convergence is less. The fast convergence rate of the ADMM sharing algorithm also makes it more appealing to achieve differential privacy guarantees than SGD, especially in the case of wide features (*gisette*).

Fig. 3.3 shows the testing loss for ADMM with different levels of Gaussian noise added. The other two baselines are the logistic regression model trained over all the features (in a centralized way) and that trained over only the local features in the first party. The baselines are trained with the built-in logistic regression function from *sklearn* library. We can see that there is a significant performance boost if we employ more features to help training the model on Party 1. Interestingly, in Fig. 3.3(b), the ADMM sharing has even better performance than the baseline trained with all features with *sklearn*. It further shows that the ADMM sharing is better at datasets with a large number of features.

Moreover, after applying moderate random perturbations, the proposed algorithm can still converge in a relatively small number of epochs, as Fig. 3.1(b) and Fig. 3.2(b) suggest, although too much noise may ruin the model. Therefore, ADMM sharing algorithm under moderate perturbation can improve the local model and the privacy cost is well contained as the algorithm converges in a few epochs.

3.6 Related Work

Machine Learning Algorithms and Privacy. [Chaudhuri and Monteleoni, 2009] is one of the first studies combing machine learning and differential privacy (DP), focusing on logistic regression. [Shokri and Shmatikov, 2015] applies a variant of SGD to collaborative deep learning in a data-parallel fashion and introduces its variant with DP. [Abadi *et al.*, 2016*a*] provides a stronger differential privacy guarantee for training deep neural networks using a momentum accountant method. [Pathak *et al.*, 2010; Rajkumar and Agarwal, 2012] apply DP to collaborative machine learning, with an inherent tradeoff between the privacy cost and utility achieved by the trained model. Recently, DP has been applied to ADMM algorithms to solve multi-party machine learning problems [Zhang *et al.*, 2018; Zhang and Zhu, 2016; Zhang *et al.*, 2019; Zhang and Zhu, 2017].

However, all the work above is targeting the data-parallel scenario, where samples are distributed among nodes. The uniqueness of our work is to enable privacy-preserving machine learning among nodes with vertically partitioned features, or in other words, the feature-parallel setting, which is equally important and is yet to be explored.

Another approach to privacy-preserving machine learning is through encryption [Gilad-Bachrach *et al.*, 2016; Takabi *et al.*, 2016; Kikuchi *et al.*, 2018] or secret sharing [Mohassel and Zhang, 2017; Wan *et al.*, 2007; Bonte and Vercauteren, 2018], so that models are trained on encrypted data. However, encryption cannot be generalized to all algorithms or operations, and incurs additional computational cost.

Learning over Distributed Features. [Gratton *et al.*, 2018] applies ADMM to solve ridge regression. [Ying *et al.*, 2018] proposes a stochastic learning method via variance reduction. [Zhou *et al.*, 2016] proposes a proximal gradient method and mainly focuses on speeding up training in a model-parallel scenario. These studies do not consider the privacy issue. [Hu *et al.*, 2019] proposes a composite model structure that can jointly learn from distributed features via a SGD-based algorithm and its DP-enabled version, yet without offering theoretical privacy guarantees. Our work establishes (ϵ, δ)-differential privacy guarantee result for learning over distributed features. Experimental results further suggest that our ADMM sharing method converges in fewer epochs than gradient methods in the case of high dimensional features. This is critical to preserving privacy in machine learning since the privacy loss increases as the number of epochs increases [Dwork *et al.*, 2014]. Another closely related work is based on the Frank-Wolfe algorithm [Bellet *et al.*, 2015; Lou and Cheung, 2018], which is shown to be efficient for sparse features. In contrast, our ADMM sharing approach is more efficient for dense features and scales much better as the number of features grows, as will be explained in Sec. 3.3.

Querying Vertically Partitioned Data Privately. [Vaidya and Clifton, 2002; Dwork and Nissim, 2004] are among the early studies that investigate the privacy issue of querying vertically partitioned data. [Kenthapadi *et al.*, 2013] adopts a random-kernel-based method to mine vertically partitioned data privately. These studies provide privacy guarantees for simpler static queries, while we focus on machine learning jobs, where the risk comes from the shared values in the optimization algorithm. Our design simultaneously achieves minimum message passing, fast convergence, and a theoretically bounded privacy cost under the DP framework.

3.7 Summary

We study learning over distributed features (vertically partitioned data) where none of the parties shall share the local data. We propose the parallel ADMM sharing algorithm to solve this challenging problem where only intermediate values are shared, without even sharing model parameters. We have shown the convergence for convex and non-convex loss functions. To further protect the data privacy, we apply the differential privacy technique in the training procedure to derive a privacy guarantee within T epochs. We implement a prototype system and evaluate the proposed algorithm on two representative datasets in risk minimization. The result shows that the ADMM sharing algorithm converges efficiently, especially on dataset with large number of features. Furthermore, the differentially private ADMM algorithm yields better prediction accuracy than model trained from only local features while ensuring a certain level of differential privacy guarantee.

Chapter 4

Efficient Load Direction Method to Erasure Coded Storage System

4.1 Introduction

Cloud storage systems, such as Hadoop Distributed File System (HDFS)[Borthakur, 2008], Google File System (GFS)[Ghemawat *et al.*, 2003], Windows Azure Storage (WAS)[Calder *et al.*, 2011] store huge amounts of data that are regularly accessed by personal users and enterprises. Built upon commodity hardware in datacenters, these systems may suffer from frequent data unavailability due to hardware failure, software glitches, I/O hotspots or local congestions. While the first generation of cloud storage systems rely on replication, e.g., 3-way replication in HDFS, for fault tolerance, many current production systems, e.g., Windows Azure Storage (WAS), Google's ColossusFS, Facebook 's HDFS, have adopted erasure coding, e.g., a (k, r) Reed-Solomon (RS) code, to offer much higher reliability than replication at a lower storage cost[Weatherspoon and Kubiatowicz, 2002; Khan *et al.*, 2012]. Local Reconstruction Codes (LRC)[Huang *et al.*, 2012; Sathiamoorthy *et al.*, 2013; Tamo and Barg, 2014] can further reduce the recovery cost, while still maintaining a low storage overhead.

However, the superior durability of erasure coded storage does not come without any tradeoff; it has been widely reported that coded storage suffers from long access latencies[Huang *et al.*, 2012; Rashmi *et al.*, 2014; Chen *et al.*, 2012; Ren *et al.*, 2013; Xia *et al.*, 2015]. Major cloud providers including Amazon, Microsoft and Google have made a common observation that a slight increase in overall data access latency (e.g., by only 400 ms) may lead to observable fewer accesses from users and thus significant potential revenue loss[Schurman and Brutlag, 2009]. Due to the latency issue, coded storage is only limited to storing data that are seldom accessed, and may suffer from long *tail* latencies. Most prior studies have attributed the cause of latency tails to *degraded reads*, defined as the passive action to read multiple (coded) objects from other storage nodes to reconstruct the unavailable object when the original object is temporarily unavailable. Therefore, most studies have focused on inventing new coding schemes to reduce the reconstruction cost, i.e., the number of reads required during degraded reads[Huang *et al.*, 2012; Xia *et al.*, 2015; Rashmi *et al.*, 2014; Sathiamoorthy *et al.*, 2013; Tamo and Barg, 2014; Khan *et al.*, 2012].

In spite of the efforts to reduce recovery cost following data unavailability events, an equally important yet largely overlooked question is—what is the most significant triggering factor of degraded reads in the first place? To answer this question, it is worth noting that most storage systems based on erasure codes today, including Google's ColossusFS and WAS, adopt systematic codes, which place each original uncoded object on a *single* node[Khan *et al.*, 2012]. A request for an object is first served by the *normal read* from its single original copy, while a timeout can trigger the degraded read. Although such a design choice attempts to avoid degraded reads as much as possible, yet it may not fulfill its intention; by presumably serving every request with the single original object, this design may greatly increase the risk of a storage node becoming congested, forcing degraded reads to be triggered involuntarily[Khan *et al.*, 2012; Huang *et al.*, 2012]. The local congestion issue is further exacerbated by the fact that most real-world demands are highly skewed[Chen *et al.*, 2012; Ren *et al.*, 2013; Xia *et al.*, 2015]. In this case, some node hosting hot data may become a hotspot and there is little chance of load balancing in the current system design, since the original copy of each object is stored only on a single node. In contrast, 3-way replication is unlikely to suffer from the same congestion issue, since it can always direct a request to a least loaded node out of the three nodes, each storing a copy of the requested object.

In this chapter, we take a radically different approach to latency reduction in coded storage systems. Instead of triggering degraded reads passively following normal read timeouts, we propose to *proactively and intelligently* launch degraded reads in order to shift loads away from hotspots and prevent potential congestion early. Note that we do not consider object write/update, since many big data stores today are append-only, in which each object is immutable and any changes are recorded as separate timestamped objects that get stored on new nodes. Intuitively speaking, if a hot object is attracting a large number of requests which may potentially congest its original storage node, we may serve some of these requests through degraded reads in the first place, without waiting for normal read timeouts. Although proactive degraded reads may reduce the longest queues, they may flood the system with more reading tasks and affect the service latencies for other requests in general. Therefore, we must carefully decide: 1) for which request a degraded read should be performed, and 2) should a degraded read be used, from which storage nodes the degraded read should be served.

Toward these objectives, we propose a variety of load balancing approaches to reduce latencies in erasure coded storage systems, including statistical optimization that can globally coordinate different requests and per-request optimal decisions. A first approach is an efficient optimization framework that intelligently maintains a load direction table between all requests and storage nodes, based on periodically sampled demand and queue statistics.

This approach is sub-optimal since it only updates direction decisions periodically, failing to utilize instantaneous load information. We then naturally turn to per-request optimal decisions, one of which is *least latency first*, that is to serve each request with the normal read or a degraded read, whichever minimizes the current request latency. However, this may lead to an excessive number of degraded reads and increase overall system burden, affecting future requests. To solve this issue, we introduce the key notion of *marginal load* and propose a novel *least-marginal-load-first* policy which judiciously and lazily launches degraded reads for load balancing based on server queue length statistics, without flooding the system. To reduce the server queue length probing overhead, we further adapt the power-of-two sampling idea to our per-request optimal load balancing in coded storage systems. We show that the per-request optimal decision is essentially the optimal solution to the statistical optimization problem for each single request with a specific objective function. Note that per-request optimal decisions have an inherent probing overhead that scales with the demand, such that a large number of concurrent controllers must be used for heavy workloads. In contrast, the statistical optimization, though being a sub-optimal approach, is scalable since it only needs to make a small fixed amount of probes in each period.

We deployed a coded storage testbed on 98 machines to evaluate the performance of the proposed schemes by replaying a large amount of real request traces collected from Windows Azure Storage. Results suggest that the proposed schemes based on proactive degraded reads can reduce the median latency by more than 40% and the 95-percentile tail latency by more than 75% in RS-coded systems and LRC-based systems, as compared to the current approach of normal read with timeouts. We show that *least marginal load first* can achieve supreme latency reduction when there is an enough number of controllers and the network I/O is not a bottleneck, whereas the statistical optimization can yield a latency close to *least marginal load first* with inertia probing, yet achieving a higher request processing throughput when the number of controllers is limited.

4.2 Load Balancing in Coded Storage

In the traditional design of erasure coded storage systems, degraded reads are triggered passively to serve a request when the *storage node* (server) storing the original requested object is temporarily unavailable or to restore a failed server. We take a radically different approach, by letting the system intentionally and intelligently perform degraded reads based on demand information and load statistics in the system to direct requests away from hot servers. Bearing request latencies, server load balancing and network I/O overhead in mind, we present several approaches to decide for which request a degraded read should be launched and from which servers the degraded read should be served.

4.2.1 Terminology and System Model

A cloud storage cluster makes redundant copies of each single logic unit of data in order to maintain the availability of highly demanded data. In such systems, a large number of small *objects* are grouped to form relatively larger *partitions* (or blocks), typically each of size 64 MB to several GBs. In a *replication-based system*, these partitions are replicated and placed on several different storage nodes. Each incoming request is served by one of the replicas[Mitzenmacher, 2001] chosen either randomly or according to more sophisticated load balancing schemes. Such systems suffer from a high storage overhead. For the typical 3-replica, there is a $3\times$ storage overhead.

Erasure coded systems are now widely adopted to reduce storage overhead while achieving high reliability and availability. Partitions form *coding groups* (or stripes). In each group, *parity partitions* are generated from the original partitions. The original and parity partitions are spread across different storage nodes. For instance, with a typical $(6, 3)$ RS code, in each coding group, 3 parity partitions are generated from 6 original partitions. Each of the 9 partitions can be recovered from any other 6 partitions in the same coding group. A $(6, 3)$ RS code can reduce the storage overhead down to $1.5\times$ with a higher reliability than 3-replica.

Each request is usually directed to the node that stores the original partition containing the requested object. We call this storage node the *original node* for the object and a read from the original node a *normal read*. When the normal read has a large delay due to temporal unavailability of the corresponding storage node, the request will be served by a *degraded read*, that is to read any 6 other partitions in the same coding group. In both a normal read and a degraded read, we do not need to read the entire partition(s); only the *offset* corresponding to the requested object needs to be read from each partition. A common problem of RS coded storage is that the system will suffer from high *recovery cost* defined as the number reads that must be performed to recover an unavailable object. Other codes have been proposed to further reduce the recovery cost, e.g., the Local Reconstruction Code (LRC)[Huang *et al.*, 2012] optimizes the recovery cost for the failure or unavailability of a single node, which is a common case in practice. Specifically, for a $(6, 2, 2)$ LRC, every 6 original partitions form a coding group, divided into two subgroups. One local parity partition is generated for each subgroup and there are two global parity partitions. Every single node failure can be recovered from 3 partitions in the local subgroup. The failure of 3 partitions in the same subgroup and some failures of 4 partitions can be recovered with the help of global parity partitions.

For a typical cloud storage system, such as WAS, client requests first arrive at certain *frontend servers* or gateways. The frontend servers direct the incoming requests to different storage nodes subject to content placement constraints and certain load balancing policies. Requests are then directly served by the selected storage node(s) to the clients. We use *request latency* to describe the time gap from the arrival of a request at the frontend server

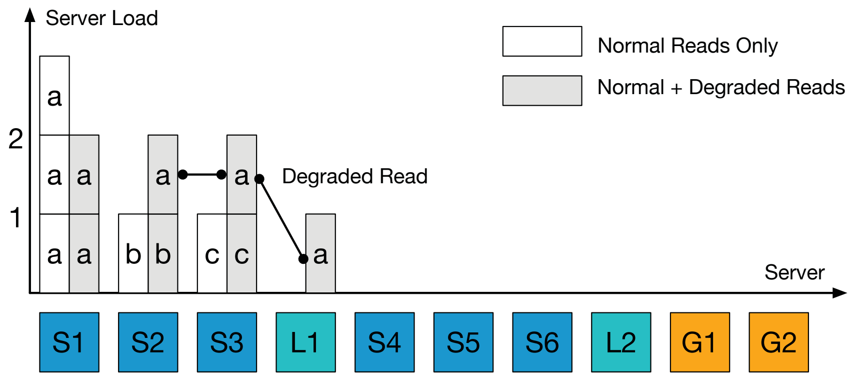


Figure 4.1: Illustration on why carefully triggered degraded reads may help balance the loads.

until the request is fully served, and use *task latency* to describe the time that it takes to perform a read task for a particular single object (coded or uncoded) being assigned to some storage node. For example, in a $(6, 3)$ RS coded system, the request latency for a normal read is just the queuing time of the request at the frontend server plus the task latency of a single read. In contrast, a degraded read will launch 6 read tasks on 6 different storage nodes. In this case, the request latency will be the longest read task latency plus the queuing time of the request at the frontend server (gateway).

We can assume that the storage nodes are homogeneous in terms of network configuration and task processing speeds, which is common in pratice. However, our ideas can easily be generalized to heterogeneous storage nodes by considering the processing speeds of servers.

4.2.2 Proactive Degraded Reads

In a replication-based system, each request can be directed to the least loaded storage node storing the requested object. However, for an erasure coded system, the original object is stored on only one storage node with little opportunities for load balancing. Therefore, traditionally, degraded reads are launched only when the normal read has timed out. However, in this chapter, we show that launching degraded reads proactively for carefully selected requests can in fact reduce access latencies and improve the overall system efficiency.

From the request traces of WAS, we found that the requests are highly skewed: most requests are for a small portion of partitions. Consider a toy example of (6,2,2) LRC code in Fig. 4.1, where 3 original partitions placed on servers $S1$, $S2$ and $S3$ form a subgroup, with a local parity partition stored on server $L1$. Suppose that server $S1$ is already heavily loaded with requests, while $S2$, $S3$ and $L1$ are relatively less loaded. When a request for an object on sever $S1$ comes, the traditional scheme still directs the request to $S1$, since $S1$ is still available although heavily loaded. In contrast, if we have proactively launched a degraded read (on servers $S2$, $S3$ and $L1$ together) to reconstruct the requested object in the first place, both the request latency and the load of server $S1$ can be reduced.

4.2.3 Statistical Optimization

To answer the question whether to serve a request with a normal read or a degraded read and which servers to serve the degraded read, we first inspect the problem in an optimization point of view. Other approaches are inspired and justified by the optimization framework.

We adopt a statistical optimization approach based on the queue status refreshed periodically as well as the request history in near past. Specifically, we keep a *load direction table* computed periodically by solving an optimization problem to be described soon, based on the latest storage node queue status and request statistics measured within a certain window. The table specifies optimized proportions at which the requests for an object in each partition should be served by the normal read or each degraded read combination. For each non-splittable request that arrives at a frontend server, load direction is made at random according to the probabilities derived from the proportions given in the load direction table.

Although we direct the load via the optimal solution, the result is still sub-optimal since the table is only updated periodically, failing to utilize the instantaneous load information. However, this approach only needs a fixed amount of probes in each period and thus is scalable to the number of requests. We present the details of the statistical optimization model in the remaining subsection.

Suppose that n partitions are placed on m storage nodes. To take into account the pending tasks on each node, let $\vec{Q} = (Q_1, \ldots, Q_m)$ represent the existing queue sizes (in bytes) on m storage nodes. Denote by $s(i)$ the original node of partition i. Let the k-tuple $c = (s_{j_1}, ...s_{j_k})$ denote a combination of k storage nodes, and $\mathcal{C}_i$ be the set of all k-node combinations c which can serve degraded reads for objects in partition i. Note that for an RS code, k is fixed. For an LRC, k could take several values depending on whether a local or global recovery is triggered[Calder *et al.*, 2011].

Suppose D_i is the instantaneous total request size per second for objects in partition i, $i = 1, \ldots, n$. For each request for an object in partition i, we use $x_i \in [0, 1]$ to denote the probability of serving it with the normal read and $y_{ic} \in [0, 1]$ to denote the probability of serving it with a degraded read through the combination $c \in \mathcal{C}_i$. Apparently, we must have $x_i + \sum_{c \in \mathcal{C}_i} y_{ic} = 1$. Let $\vec{L} = (L_1, \ldots, L_m)$, where L_j is the *expected load* of storage node j as a result of load direction. Then, due to the linearity of expectations, the expected load L_j is a weighted sum of all the demands $D_1, \ldots, D_n$ weighted by the load direction probabilities x_i and $\{y_{ic} | c \in \mathcal{C}_i\}$, $i = 1, \ldots, n$.

Let $F(\cdot)$ be a *load balancing metric* defined as a function of $\vec{L} + \vec{Q}$, i.e., the expected loads $\vec{L}$ on all storage nodes plus the existing queues pending on them. Therefore, the optimal load direction probabilities are the solution to the following problem:

$$\begin{aligned}
\underset{\{x_i\},\{y_{ic}\}}{\text{minimize}} \quad & F(\vec{L}+\vec{Q}) && (4.1)\\
\text{subject to} \quad & L_j = \sum_{\{i:s(i)=j\}} D_i x_i + \sum_{\{(i,c):j\in c, c\in\mathcal{C}_i\}} D_i y_{ic}, \\
& \quad j = 1,\ldots,m, && (4.2)\\
& x_i + \sum_{c\in\mathcal{C}_i} y_{ic} = 1, \quad i = 1,\ldots,n. && (4.3)\\
& y_{ic} \geq 0, \quad \forall c \in \mathcal{C}_i,\ i = 1,\ldots,n, && (4.4)\\
& x_i \geq 0,\ i = 1,\ldots,n, && (4.5)
\end{aligned}$$

where load L_j directed to storage node j is given by (4.2). The first term in (4.2) is the sum of all normal reads $D_i x_i$ with the storage node j being the original node, while the second term is the sum of all degraded reads $D_i y_{ic}$ that may incur loads on node j, i.e., $j \in c, c \in \mathcal{C}_i$.

Several load balancing metrics can be used, for example, the ℓ_∞ norm: $F(\vec{L}+\vec{Q}) = \|\vec{L}+\vec{Q}\|_\infty = \max_{j=1,\ldots,m}(L_j+Q_j)$, which models the longest expected queue length after load direction, and the ℓ_2 norm: $F(\vec{L}+\vec{Q}) = \frac{1}{2}\|\vec{L}+\vec{Q}\|_2^2 = \frac{1}{2}\sum_{j=1}^{m}(L_j+Q_j)^2$, which models the aggregated per-byte processing latency in the system, assuming a similar processing rate across storage nodes in a homogeneous system. The metric can easily accommodate the case of heterogeneous systems where storage node j has a data processing rate of S_j; in this case, $F(\vec{L}+\vec{Q}) = \frac{1}{2}\sum_{j=1}^{m}(L_j+Q_j)^2/S_j$. If the ℓ_∞ norm is used, problem (4.1) is a linear program, and if the ℓ_2 norm is used, problem (4.1) is a quadratic program. In our experiment, we adopt the ℓ_2 norm. Standard tools like MOSEK[ApS, 2017] can solve it efficiently with worst-case time complexity of $O(n^3)$.

4.2.4 Per-Request Optimal Decisions

With the statistical optimization approach, the load direction table is updated in a synchronized manner and is not changed within each cycle, leading to a sub-optimal result. A more direct approach is to instantaneously probe the queue status of related data nodes and to make an optimal decision for each request. In this case, we need a criterion to measure how good a load direction choice is. Optimal policies can be derived by adapting problem (4.1) to per-request decisions under different load balancing metrics.

Least Latency First. We first introduce the *least latency first* policy, which corresponds to the per-request minimization of the ℓ_∞ norm of storage node queue sizes (or the maximum queue size). Consider a single request for an object of size D in partition i_0. In this case, the optimal policy comes from a solution to a special case of problem (4.1), with $D_{i_0} = D > 0$ and $D_i = 0, \forall i \neq i_0$, that is to solve problem (4.1) for this single request. Every single non-splittable request will be served by either the normal read or one of the degraded combinations. The optimal choice will be the one that leads to the smallest objective value in problem (4.1).

To find out the optimal decision, we can choose either the normal read or a degraded read that results in the lowest estimated request latency for the current request. For the normal read, the request latency almost equals to the corresponding task latency, which can be estimated *by the total size of all queued requests at the storage node plus the requested object size, divided by the average node processing rate.* For a degraded read served by k other storage nodes, the request latency can be estimated *by the longest task latency among the k nodes.* Note that in problem (4.1), different servers are assumed to have the same processing rate. Thus, the processing rate is omitted in the objective function.

For example, the least-latency-first policy can be performed in a $(6, 3)$ RS coded system in the following way: Upon the arrival of a request, the queue sizes at the 9 storage nodes that can serve this request are probed, including the original node and 8 other nodes, any 6 of which can serve the request via a degraded read. Then, we will compare the task latency of the normal read with the longest task latency among the 6 *least loaded nodes* out of the other 8 nodes, and pick whichever is smaller to serve the request.

However, the least-latency-first policy may not perform the best in the long term since it only optimizes the latency of *the current request in question* regardless of future requests. In fact, for a request for a hot object, the least-latency-first policy tends to shift load away from the original node, whenever there are at least 6 other nodes in the same coding group with a queue size smaller than that of the original node. Although such a “water-filling” approach will help to balance server loads, the drawback is that it encourages degraded reads too much and increases the overall number of read tasks launched in the system, which may prolong the service latencies of future requests, as their original nodes have been used to perform degraded reads for earlier requests. Therefore, we need a load direction policy to reduce the burden of heavily loaded storage nodes, while still penalizing degraded reads.

Least Marginal Load First. To strike a balance between reducing the current request latency and minimizing the overall system load, we propose a different metric to measure different load direction choices. We introduce the *least-marginal-load-first* policy. Similar to the case of LLF, this policy is essentially an optimal solution to problem (4.1), but with an ℓ_2 objective function. Let us consider the special case of problem (4.1) again, with $D_{i_0} = D > 0$ and $D_i = 0, \forall i \neq i_0$, which is to solve problem (4.1) for this single request. Comparing the ℓ_2 objective function values before and after the request is directed, each direction decision will increase the objective by a certain amount. Specifically, for the normal read, the increase is

$$\Delta F_{s(i_0)} = \frac{1}{2}\sum_{j \neq s(i_0)} Q_j^2 + \frac{1}{2}(Q_{s(i_0)} + D_{i_0})^2 - \frac{1}{2}\sum_j Q_j^2$$
$$= D_{i_0}(Q_{s(i_0)} + \frac{1}{2}D_{i_0}) = D(Q_{s(i_0)} + \frac{1}{2}D), \tag{4.6}$$

and for a degraded read with $c \in \mathcal{C}(i_0)$, the increase is

$$\Delta F_c = \frac{1}{2}\sum_{j \notin c} Q_j^2 + \frac{1}{2}\sum_{j \in c}(Q_j + D_{i_0})^2 - \frac{1}{2}\sum_j Q_j^2$$
$$= \sum_{j \in c} D_{i_0}(Q_j + \frac{1}{2}D_{i_0}) = \sum_{j \in c} D(Q_j + \frac{1}{2}D). \tag{4.7}$$

The optimal choice would be the one that leads to the minimum increase of the objective function. We can pick it out by computing the value of (4.6) for the normal read and the values of (4.7) for all degraded reads and selecting the one with the minimum value.

Intuitively speaking, consider assigning a read task of size D on a storage node with existing queue size Q. With a similar processing rate across servers, $Q+D/2$ can be regarded as the per-byte *average* processing time. $D \cdot (Q+D/2)$ is the summation of processing times per byte in this read task. We call this value the *marginal load* of the read task of size D. Let us now define the marginal load of a normal read and that of a degraded read. The marginal load of a normal read is just the marginal load of the corresponding single read task, whereas the marginal load of a degraded read is *the summation of all the marginal loads of the associated read tasks.* We define the *least-marginal-load-first* (LMLF) policy as choosing the normal read or a degraded read that achieves the least marginal load, which is naturally an optimal solution to problem (4.1) for a single request with an ℓ_2 objective function.

For example, in a $(6, 3)$ RS coded system, for an object request, *least marginal load first* will compare the marginal load $DQ_0 + D^2/2$, where Q_0 is the original node queue size, with $\sum_{i=1}^{6}(DQ_i + D^2/2)$, where $Q_1, \ldots, Q_6$ are the queue sizes of the 6 *least loaded nodes* out of the other 8 nodes, and pick whichever is smaller to serve the request.

LMLF strikes a balance between reducing the current request latency and minimizing the overall system load. On one hand, storage nodes with a large Q incur a larger marginal load and are less likely to be selected. On the other hand, the marginal load of a degraded read is the summation of the marginal loads on all associated nodes, penalizing degraded reads from flooding the system. Moreover, objects with a larger size D are less likely to be served by degraded reads due to their higher marginal loads, attributed to the additive term $D^2/2$ in the marginal load of each associated task. In other words, LMLF launches degraded reads lazily and saves the system resources for future requests.

4.2.5 Distributed Power-of-Two Sampling

The per-request optimal decisions above rely on instantaneously probed queue sizes for each request, incurring much probing overhead at the frontend (gateway) servers. For a (k, r) RS code, $k + r$ storage nodes need to be probed for each request.

To save the probing overhead, we exploit an idea similar to the Power of Two in job scheduling for load direction in coded storage systems. Specifically, we can just pick a random degraded read and only probe the k storage nodes related to this degraded read and compare its marginal load with that of the normal read. Taking the $(6, 3)$ RS code as an example, instead of having 9 probes for each request, in the sampled solution, we only need to probe the original node and a set of randomly chosen 6 storage nodes in the same coding group as the requested object, and use whichever achieves a lower marginal load, which saves the probing overhead by 22.2%. In particular, codes that are designed to optimize the recovery cost for single node failures will benefit the most from sampling. In a typical $(6, 2, 2)$ LRC, if only the local recovery combination and the normal read are considered, we only need to probe 4 storage nodes for each request, which saves the probing overhead by 60%, compared to the full probing of 10 storage nodes.

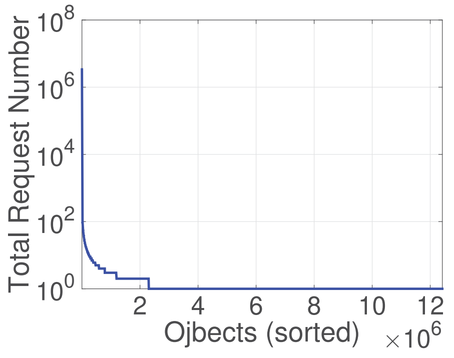
Theoretical results[Mitzenmacher, 2001] on Power-of-Two load balancing in traditional queuing systems have shown that the expected performance will not drop too much as compared to full probing. We will verify the effectiveness of our version of distributed power-of-two sampling with experiments.

4.2.6 Summary

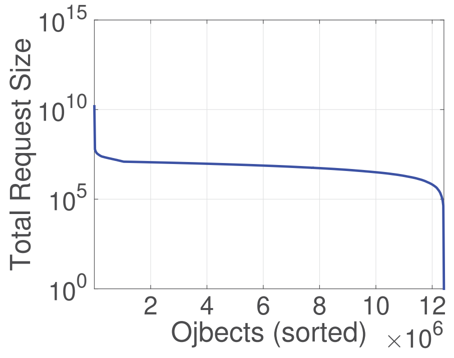
We propose different methods to help making the optimal decision for the proactive degraded reads. There is a tradeoff in each method. Least latency first probes queue status and optimizes for each request instantaneously. But it does not coordinate different requests and incurs larger probing overhead. Least marginal load first not only optimizes for each request with instantaneous probing, but also saves system resources for future requests by penalizing degraded reads. The distributed power-of-two sampling can alleviate the probing burden at the cost of a slight deviation from the optimal solution. Finally, in the case that probing overhead could form a bottleneck, statistical optimization can be used to jointly direct the loads for all requests taking advantage of joint statistics of different requests, although the solution is only an approximation to the optimality due to the lazily updated demands and queue table.

4.3 Implementation and Experiment Setup

We implemented and deployed a prototype coded storage testbed to evaluate the performance the proposed load balancing mechanisms on a cluster of 98 Amazon EC2 virtual machines (which do not use SSD) by replaying the request traces we have collected from



(a) Distribution of aggregatedrequest number



(b) Distribution of aggregated request size (bytes)

Figure 4.2: The properties of the trace data collected from Windows Azure Storage (WAS).

Windows Azure Storage (WAS), containing the request logs for 12 million objects over a one-day period. The objects are randomly grouped into 2004 partitions of an equal size. Fig. 5.1(a) plots the distribution of the aggregated demands in terms of request number and request file size for different objects in the trace. We can see that they are highly skewed: a small number of hot objects attract a large amount of requests, while most objects are cold. We use a randomized partition placement strategy, typically adopted in WAS: when a (k, r) RS code is applied, each coding group has k original and r parity partitions. These $k+r$ partitions are placed on $k+r$ random storage nodes such that the storage cost at each node is balanced; when a (k, l, r) LRC is used, a coding group of $k+l+r$ original/parity partitions are placed on $k+l+r$ random nodes in a similar way.

Our testbed has an architecture shown in Fig. 4.3 and Fig. 4.4, consisting of two major parts: *frontend servers* and *storage nodes*. Each frontend server works in a decentralized way, and can receive requests and direct them to storage nodes according to a load balancing policy. Each frontend has access to the content placement map that indicates which partition is stored on which storage nodes, as well as the offsets of each object in its partition. For each request, we just need to read from the specified offset in the corresponding partition for the object size. In our experiments, the requests in the traces (each in a form of <time>, <object id>, <size requested>) are fed to one frontend server at random to mimic the way that requests arrive at frontend servers from the clients in a balanced way. We do not consider more sophisticated frontend load balancing, since the focus is on the load balancing of storage nodes.

The read tasks on each storage node are executed sequentially in a first-come-first-service (FCFS) queue. Each storage node keeps track its queue size. Since the read tasks are different in sizes (due to different object sizes), the number of tasks on a storage node is not a good estimate of its real load[Ousterhout *et al.*, 2013]. Our system uses the aggregated size of read tasks on each storage node to estimate its queue size, which is also easy to keep track of. We do not consider object write/update, since many big data stores today are

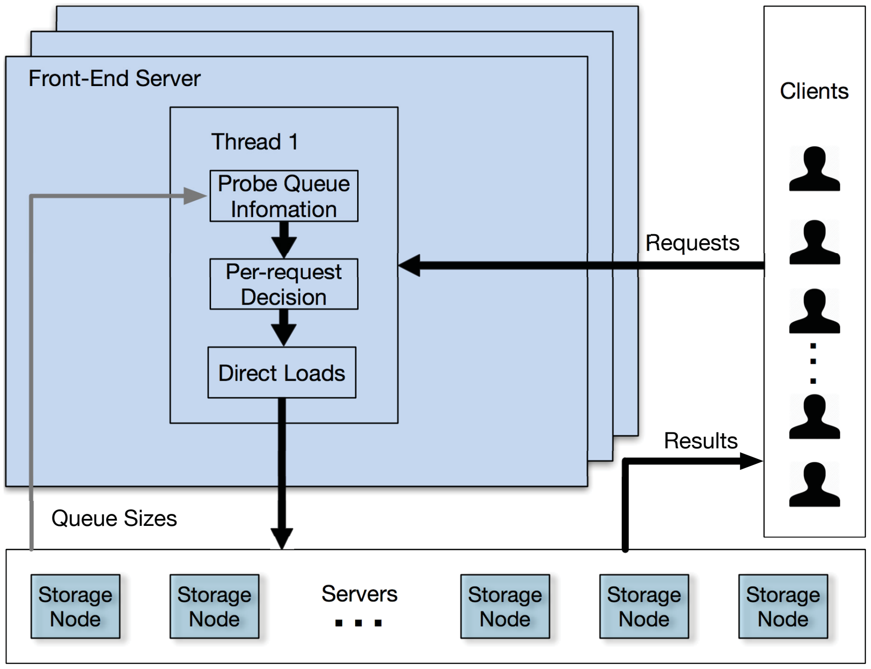


Figure 4.3: System architecture with per-request optimal decisions based on instantaneous probing.

append-only, in which each object is immutable and any changes are recorded as separate timestamped objects that get stored on new nodes. Hence, write requests are unlikely to create a hotspot on a single node and our focus here is on enabling faster reads.

4.3.1 The Controller and Logging

Each frontend server has a controller to execute load direction policies. In this work, we evaluate the performance of the following polices:

1) **Normal**: normal read with degraded read triggered upon timeout (a common existing solution);

2) **LLF**: least latency first with per-request full probing of all related storage nodes;

3) **LMLF**: least marginal load first with per-request full probing of all related storage nodes;

4) **LLF PW2**: least latency first with per-request power-of-two probing on sampled nodes;

5) **LMLF PW2**: least marginal load first with per-request power-of-two probing on sampled nodes;

6) **LMLF Lazy**: least marginal load first with periodic updates of storage node queue status;

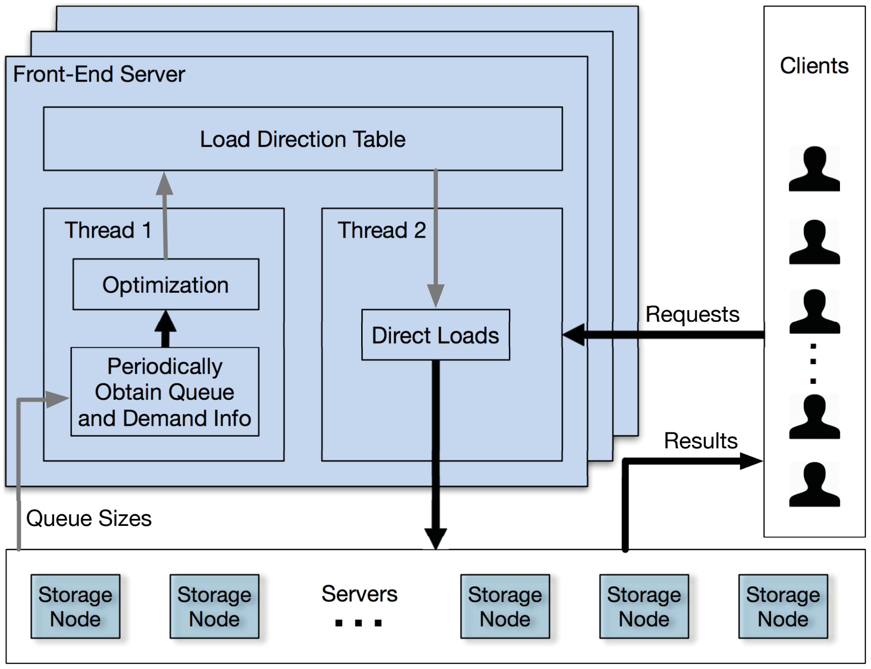
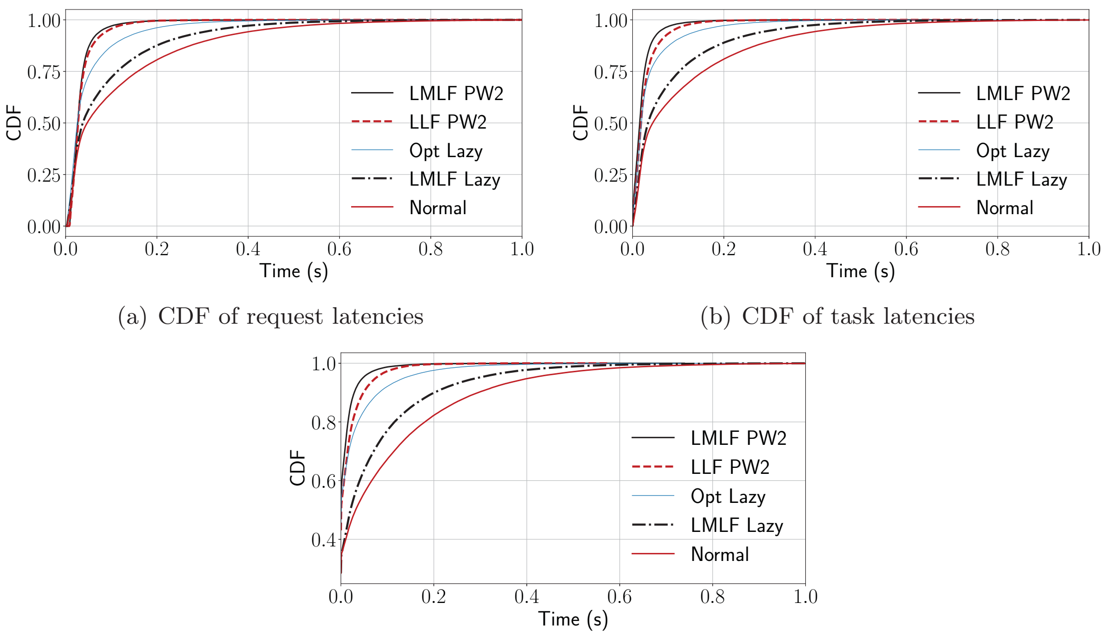


Figure 4.4: System architecture with statistical optimization based on lazy queue status updates.

7) **Opt Lazy**: statistical optimization of ℓ_2 norm with periodic updates of storage node queue status.

For LLF and LMLF as shown in Fig. 4.3, the controller probes all the related nodes for each request, e.g., 9 nodes for a $(6, 3)$, for their current queue sizes and make per-request decisions based on LLF or LMLF. For LLF PW2 and LMLF PW2, only the original node and the nodes for one degraded read combination are probed: for RS, it is a random degraded read, and for LRC, it is the local recovery combination. Thus, LLF PW2 and LMLF PW2 have lower probing overhead. For LMLF Lazy, one separate thread probes all queue status periodically and updates the queue table on all controllers every T seconds, while each controller makes an LMLF decision for each incoming request based on the lazily updated queue table.

For Opt Lazy as shown in Fig. 4.4, besides the queue table updated every T seconds, the aggregate request rate for each partition is also recorded from the previous T seconds. Then, a separate thread, implemented with MOSEK[ApS, 2017] in Python, computes a *load direction table* with the statistical optimization method in Sec. 4.2.3, and obtains the optimal *probabilities* that a request for an object in each partition will be served by the normal read *or* certain degraded reads. Note that the same load direction probabilities are applied to all the objects in a same partition, thus limiting the load direction table to a practical size. We do not differentiate between the objects in a partition, since overly fine-grained load direction increases overhead and partition-level load balancing is sufficient



(a) CDF of request latencies

(b) CDF of task latencies

(c) CDF of task waiting times

Figure 4.5: The CDFs of different latency metrics for the storage system based on a (6,2,2) LRC.

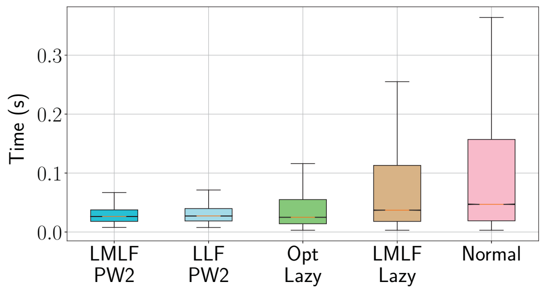
in our experiments. Each controller then probabilistically directs each incoming request based on the current load direction table.

Note that Opt Lazy only solves problem (4.1) between the normal read and a single degraded read: for RS, it is a random degraded read, and for LRC, it is the local recovery combination. This simplification significantly reduces the optimization computation time to around 0.1 second for our problem sizes yet achieving similar objective values to full optimization according to simulations. To simplify the synchronization issue among frontend servers, we let each controller compute a separate (yet nearly identical) load direction table every T seconds based on the current queue status and collected demand information. And there are two separate concurrent threads for computing the load direction table and for directing each incoming request based on the current table.

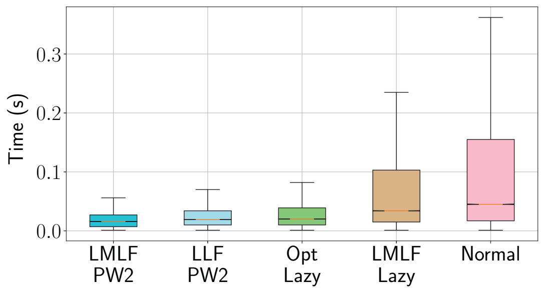
We record the following real timestamps for each request: A) the time that the request arrived at the frontend server, B) the time that the frontend server sent the read task(s) of this request to storage nodes, C) the time that each task entered the queue at the designated storage node, D) the time that the task processing started, E) the time that the request processing finished.

4.4 Experimental Results

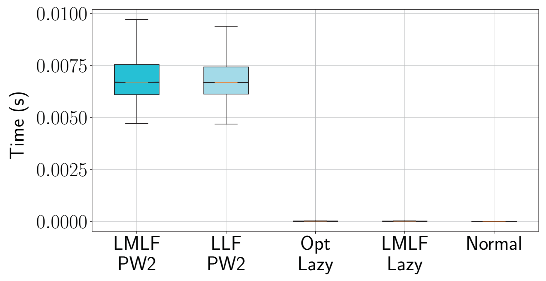
Our evaluation is based on replaying a large amount of traces collected from the Windows Azure Storage (WAS), containing the request logs for 12 million objects over a one-day



(a) Box plot of request latencies



(b) Box plot of task latencies



(c) Box plot of controller processing times

Figure 4.6: Box plots for different latency metrics for the storage system based on a (6,2,2) LRC.

period. We randomly group them into 2004 partitions of an equal size and import them into the coded storage testbed deployed on a 98-machine cluster, including 90 Amazon EC2 *t*2.*nano* instances (which are not based on SSD to mimic commodity hardware used in practice), which serve as the storage nodes, and 3 or 8 quad-core Amazon EC2 *m*4.*xlarge* instances, which serve as the front-end servers. Fig. 5.1(a) shows the aggregated request numbers and sizes for different objects. The requests are highly skewed in that a few objects have contributed a considerable portion of the total workload to the system.

We replayed the traces in a typical peak hour under different load direction polices listed in Sec.4.3, respectively, in the testbed for both a $(6, 3)$ RS coded storage system and a $(6, 2, 2)$ LRC-based storage system, and record the timestamps we mentioned in Sec. 4.3.1. For the LRC system, we evaluated the following 5 polices: Normal, LLF PW2, LMLF PW2, LMLF Lazy and Opt Lazy, since in LRC probing all the degraded read combinations will incur an overly high cost. For the $(6, 3)$ RS coded system, we evaluated all the polices listed in Sec. 4.3.1. In our experiment, we set the queue status probing frequency to once per 4 seconds for Opt Lazy and once per second for LMLF Lazy. We use 8 front-end servers for the LLF, LMLF, LLF PW2, LMLF PW2 policies to support a high capability for instantaneous probing and 3 front-end servers for the Normal, Opt Lazy and LMLF Lazy policies, which have much lower probing overhead.

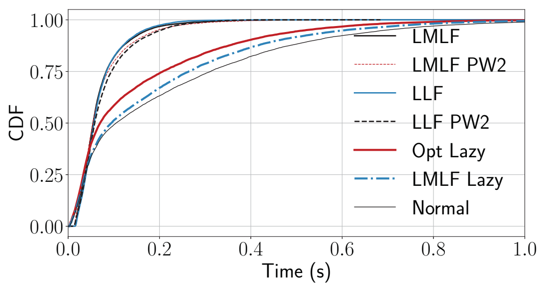
4.4.1 Performance in the LRC-based System

For an LRC-based system, Fig. 4.5(a) and Fig. 4.6(a) show the request latencies under different policies. The request latency is the actual latency each request encounters including the controller processing time (the delay at the controller before read tasks are launched) and the maximum of related task latencies. Fig. 4.5(b) and Fig. 4.6(b) show the task latencies under different policies. The task latency is the time gap between the time a task is inserted into the queue of a storage node and the completion of the task. Fig. 4.5(c) shows the task waiting time which is the time between a task entering a storage node and the beginning of its processing. Fig. 4.6(c) shows the controller processing time, which is the delay each request experiences at the controller of the front-end server, including queuing and decision-making. Table 4.1 further shows the overall statistics of the performance of different polices.

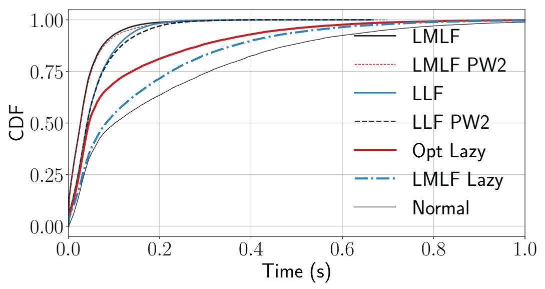
First, we can see that all the polices with judiciously launched proactive degraded reads can improve the access latency compared to the traditional policy Normal that triggers degraded reads following the timeouts of normal reads. Specifically, LMLF PW2, using the marginal load and power-of-two sampled probes for optimal per-request decisions, reduces the mean latency by 44.7% and the 95th-percentile by 77.8% with respect to Normal.

Least Marginal Load First vs Least Latency First. Fig. 4.5(b) and Fig. 4.6(b) show the task latencies for different policies. LMLF PW2 is considerably better than the LLF PW2 by using the marginal load as a load balancing criterion. As we have analyzed, LMLF PW2 tries to strike a balance between the reduced individual request latency and the overall system load while LLF PW2 only makes decisions to optimize the current request latency, thus launching degraded reads too aggressively. Although LMLF PW2 does not optimize the current request latency directly, it is essentially a per-request optimization for the aggregated task latencies in the long run as we have shown in Sec. 4.2. In Fig. 4.5(a) and Fig. 4.6(a), the advantage of LMLF PW2 over LLF PW2 in terms of the request latency is not as obvious as it is in terms of the task latency. This is due to a higher decision-making complexity on the front-end servers for LMLF PW2 as shown in Fig. 4.6(c) and the fact that the request latency for the degraded read is dominated by the maximum of all the related task latencies. In general, LMLF PW2 outperforms LLF PW2 with better request latency performance and much better task latency performance.

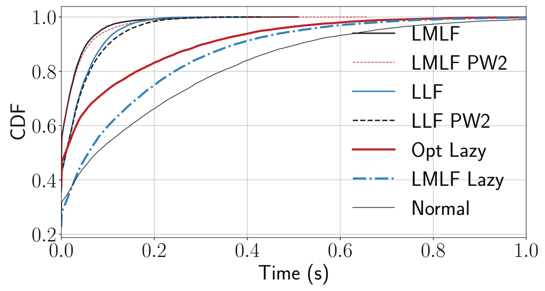
Opt Lazy vs. LMLF PW2. In terms of the request latency and task latency as shown in Fig. 4.5(a) and Fig. 4.5(b), LMLF PW2 outperforms the optimization scheme Opt Lazy. However, Opt Lazy needs much fewer probes than the LMLF as is shown in Table 4.1. In fact, the probing overhead of LMLF PW2 scales linearly as the number of requests increases while for Opt Lazy, the probing overhead is linearly related to the number of storage nodes, thanks to the use of periodic lazy updates. Therefore, Opt Lazy may potentially be able to handle a much larger amount of requests than the LMLF PW2 especially when the number controllers is limited. Moreover, the controller processing time of LMLF PW2



(a) CDF of request latencies



(b) CDF of task latencies



(c) CDF of task waiting times

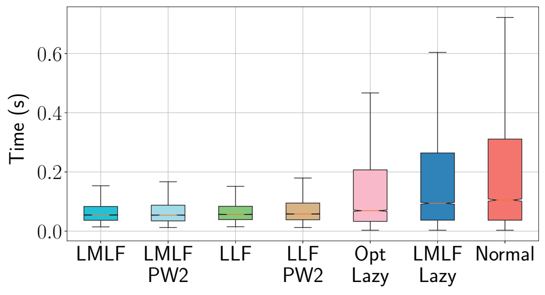
Figure 4.7: CDFs of different latency metrics for the storage system based on a (6,3) RS code.

shown in Fig. 4.6(c) is much larger than that of Opt Lazy. Opt Lazy can efficiently assign the requests to the storage nodes based on the statistical optimization results calculated by another thread while LMLF PW2 might suffer from a possibly long probing delay or even timeout when the demand is too high as compared to the available network bandwidth. This phenomenon is not obvious in our system since the network environment was very stable when the experiments were conducted.

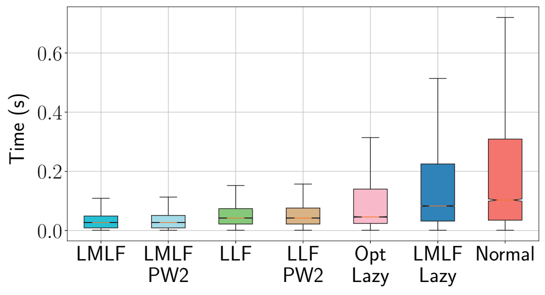
Opt Lazy vs. LMLF Lazy. Both of the two approaches only need to acquire the queue status of the storage node periodically, thus with low probing overhead. However, Opt Lazy outperforms LMLF lazy significantly since LMLF Lazy usually suffers from race conditions since queue status is not up-to-date. The optimization method makes decisions based on both the periodically updated queue status and request statistics, which help to make a better decisions by jointly consider and coordinate the requests for objects in different partitions.

4.4.2 Performance in the RS-coded System

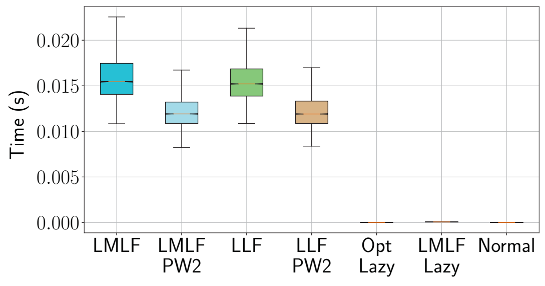
Fig. 4.7(a) and Fig. 4.8(a) plot the request latencies under different load direction policies in a (6,3) RS coded system. Fig. 4.7(b) and Fig. 4.8(b) plot the task latencies, while Fig. 4.5(c) shows the task waiting times. Fig. 4.8(c) shows the box plot for task waiting times and controller processing times. Table 4.2 further shows the overall statistics of the performance of different polices. For the RS code, the performance demonstrates a similar overall trend



(a) Box plot of request latencies



(b) Box plot of task latencies



(c) Box plot of controller processing times

Figure 4.8: Box plots of different latency metrics for the storage system based on a (6,3) RS code.

as that for the LRC system: we can reduce the request latency significantly by judiciously launching proactive degraded reads.

LMLF PW2 vs. LMLF. As shown in Fig. 4.7(a) and Fig. 4.7(b), LMLF is better than LMLF PW2 since it makes per-request optimal choices based on full probing. However, the improvement is limited, since the requests are highly skewed and most of the time, the decision between whether to trigger a degraded read or not has a more significant impact to the performance while selecting which recovery combination for a degraded read is relatively less important. There is a similar trend between LLF PW2 and LLF. Therefore, there is a high motivation to use the LMLF PW2 and LLF PW2 instead of LMLF and LLF with full probing to save the probing cost. Note that in a (6,3) RS coded system, the saving of probes is 1/3. This amount varies among different erasure coded schemes depending on the total number of partitions in a coded group and how many partitions are needed to recover a single partition.

LMLF vs. LLF. As shown in Fig. 4.7(a), the performance of LMLF seems to be slightly worse than LLF. Similar to the case in LRC system, LMLF has higher controller processing time. Also, LMLF tries to keep the overall workload of the storage nodes to a lower level and only reduce the request latency tail. On the other hand, LLF searches for a decision with minimum per-request latency directly and thus gets a better request latency. But as is shown in Fig. 4.7(b), LMLF has much better task latencies so that it incurs a much lower overall task workload on the storage system and has the potential to serve more

Table 4.1: Overall request latencies (seconds) and probing overhead (# probes/s) for the LRC-based system.

Policy	mean	var	median	95th	min	max	# probs per second
Normal	0.114	0.022	0.047	0.424	0.003	1.244	0
LMLF Lazy	0.087	0.013	0.037	0.324	0.003	1.197	270
Opt Lazy	0.048	0.005	0.025	0.177	0.003	1.038	67.5
LMLF PW2	0.033	0.001	0.026	0.080	0.008	1.038	888.9
LLF PW2	0.036	0.001	0.027	0.094	0.008	1.040	888.9

Table 4.2: Overall request latencies (seconds) and probing overhead (# probes/s) for the RS coded system.

Policy	mean	var	median	95th	min	max	# probs per second
Normal	0.207	0.056	0.105	0.695	0.003	1.951	0
LMLF Lazy	0.180	0.041	0.094	0.607	0.003	1.331	270
Opt Lazy	0.148	0.031	0.069	0.520	0.003	1.420	67.5
LMLF PW2	0.073	0.005	0.054	0.196	0.012	3.043	784.5
LLF PW2	0.076	0.004	0.058	0.203	0.012	0.684	784.5
LMLF	0.070	0.003	0.055	0.174	0.014	1.016	1008.7
LLF	0.069	0.002	0.057	0.166	0.015	1.053	1008.7

demands.

4.5 Related Work

Most prior work has attributed the long latency tails in coded storage to degraded reads. A large amount of research has focused on reducing the recovery cost during degraded reads. Local Reconstruction Code (LRC)[Huang *et al.*, 2012] is proposed to reduce the IOs required for reconstruction over Reed-Solomon (RS) codes, while still achieving significant reduction in storage overhead as compared to 3-way replication. Similar locally recoverable codes have been presented in[Sathiamoorthy *et al.*, 2013; Tamo and Barg, 2014]. HitchHiker[Rashmi *et al.*, 2014] is another erasure-coded storage system that reduces both network traffic and disk I/O during reconstruction, residing on top of RS codes based on new encoding/decoding techniques. HACFS[Xia *et al.*, 2015] uses two different erasure codes, i.e., a fast code for frequently accessed data to lower the recovery cost, and a compact code for the majority of data to maintain a low overall storage overhead.[Zhu *et al.*, n.d.] presents an algorithm that finds the optimal number of codeword symbols needed for recovery with any XOR-based erasure code and produces recovery schedules to use a minimum amount of data. [Khan *et al.*, 2012] proposes FastDR, a system that addresses node heterogeneity and exploits I/O parallelism to enhance degraded read performance.

However, another important question is: what is the cause of degraded reads in coded storage in the first place? In fact, aside from node failures, the majority of degraded reads are passively triggered during *temporary unavailability* of the original node[Huang *et al.*, 2012; Xia *et al.*, 2015; Rashmi *et al.*, 2014]. For example, Over 98% of all failure modes in Facebook's data-warehouse and other production HDFS clusters require recovery of a single temporary block failure[Rashmi *et al.*, 2013] instead of node failures. And only less than 0.05% of all failures involve three or more blocks simultaneously. Furthermore, a major reason underlying such temporary node unavailability is that under skewed real-world demands[Abad *et al.*, 2012; Chen *et al.*, 2012; Ren *et al.*, 2013; Xia *et al.*, 2015], there is a high risk that a few nodes storing hot data may become hotspots while other nodes are relatively idle. In this chapter, we argue that rigid load balancing schemes, i.e., passive recovery after timeout, is a major cause for long latency tails in coded storage, especially in the presence of skewed demands. In this case, we can actually reduce latency by proactively launching degraded reads for some requests to shift loads away from hotspots early.

Recently, there have been other studies to reduce download latency from coded storage systems, mainly leveraging redundant downloads[Chen *et al.*, 2014; Liang and Kozat, 2014; Sun *et al.*, 2015; Shah *et al.*, 2016; Joshi *et al.*, 2015; Shah *et al.*, 2014]. The idea is to download more than k coded blocks in a (k, r) RS-coded system to exploit a queueing-theoretical gain: as soon as the first k blocks are obtained, the remaining downloads can be stopped. However, such a scheme mainly benefits *non-systematic* codes, where there is no original copies of objects in each coding group. A latency optimization model has been proposed in[Xiang *et al.*, 2014] to jointly perform erasure code selection, content placement, and scheduling policy optimization, also for *non-systematic* codes. In contrast, we focus on systematic codes (where for each object, there is a single node storing its original copy) that are commonplace in production environments to allow normal reads. With systematic codes, always downloading k or more blocks is not efficient. Besides, we mainly focus on disk I/O bottlenecks due to queued read tasks instead of variable network latencies of downloads.

The power-of-two-choices algorithm[Mitzenmacher, 2001; Mitzenmacher, 1996] is a classical randomized load balancing scheme with theoretical guarantees and wide applications[Richa *et al.*, 2001]. Recently, there have been renewed interests to generate power-of-two load balancing to low-latency scheduling of batched tasks. Sparrow[Ousterhout *et al.*, 2013] proposes to schedule a batch of m tasks in a Spark job to multiple workers by selecting the lead loaded m out of dm probed workers. Later, it is theoretically shown [Ying *et al.*, 2015] that a similar batch sampling technique maintains the same asymptotic performance as the power-of-two-choices algorithm while reducing the number of probes. Our per-request optimal decisions generalize the power-of-two idea to the load balancing between the normal read and different degraded reads in a erasure coded storage system, where the objective for comparison is not so obvious as in prior job scheduling literature. We propose the least-marginal-load-first policy can judiciously trades between current request latency and overall system efficiency. Moreover, we unify the proposed schemes in an optimization framework that can be executed lazily to further save probing overhead.

Many prior efforts have been devoted to reducing tail latencies in replication-based systems. Specifically, the $C3$ system in[Ganjam *et al.*, 2015] presents a distributed approach to reducing the tail latency, stabilizing the behavior via a server ranking function that considers concurrent requests on the fly and penalizes those servers with long queue sizes. Tiny-Tail Flash[Yan *et al.*, 2017] eliminates tail latencies induced by garbage collection by circumventing GC-blocked I/Os with four novel strategies proposed. One of such strategies is to proactively generate content of read I/Os that are blocked by ongoing GCs. In this work, we focus on reducing tail latencies in storage systems that are based on systematic erasure codes, by leveraging proactively launched degraded reads. Furthermore, we address the request concurrency issue in a much more complex situation with theoretically inspired methods.

4.6 Summary

Erasure-coding-based storage systems often suffer from long access latency tails. Prior studies have attributed this to the presence of degraded reads when the original data is unavailable and mainly aimed at improving coding structures to reduce degraded read costs. We take a radically different approach to tail latency reduction in coded storage systems and launch degraded reads intentionally and judiciously to balance the loads. Specifically, we propose a variety of schemes to direct loads based on either per-request decisions made from instantaneously probed storage node queue status or an optimized load direction table computed by statistical optimization with lazy queue status probes. We implemented a prototype system and deployed it on a cluster of 98 machines to evaluate the performance based on a large amount of real-world traces. Results suggest that the proposed least-marginal-load-first policy based on instantaneous sampled queue status can reduce the median request latency by more than 40% and the 95-percentile tail latency by more than 75% in both RS-coded systems and LRC-based systems, as compared to the existing approach of normal reads followed by passive degraded reads upon timeouts. The statistical optimization approach with lazy queue probing can also significantly reduce request and task latencies with a much lower system probing overhead.

Chapter 5

Efficient Block Placement Method to Erasure Coded Storage System

5.1 Introduction

In Chapter 4, we study the latency reduction problem in systematic erasure coded storage systems. We propose to proactively and intelligently launch degraded reads to shift loads away from hotspots and prevent congestion. In this chapter, we propose to reduce the access latency in coded storage systems through a new approach of block placement adjustment and controlled block migration. Although there is little chance to choose servers during normal reads, we may place blocks with *anti-correlated* demands on a same server to benefit from statistical multiplexing and prevent certain hot blocks from congesting a specific server. We formulate the content placement optimization to minimize the expected average waiting time of all incoming requests, based on the mean and covariance of demands for different blocks, which can be readily measured according to recent request history. To avoid globally shuffling blocks across the system, we require all block migration to be *local*, and move as few blocks as possible with respect to the current placement to reduce the migration overhead.

Our statistical content placement problem is similar to the Min-k-Partition problem, a well-known NP-complete problem[Kann *et al.*, 1997; Karp, 1972], which aims to divide a graph into k partitions to minimize the sum of all intra-partition edge weights. Yet, our problem turns out to be even more challenging, since we also need to handle an additional constraint that no blocks from the same coded group can be placed on the same server, which is needed to maintain the promised reliability of an (k, r) RS code. We introduce a novel technique to convert this constraint into carefully designed weights in the objective function and propose a time-efficient local search algorithm which only moves the block that reduces the latency objective the most at a time. We prove that our algorithm always produces a feasible solution that satisfies the special constraint and theoretically derive the worst-case approximation ratio of our algorithm with respect to the global optimality. We characterize such a ratio as a function of demand statistics; the larger the demand variation among blocks, the better the approximation.

Through simulations based on real request traces collected from the Windows Azure Storage system, we show that our local block migration scheme can greatly reduce the overall access latency by only moving a small portion of all the blocks, and outperforms a best randomized content placement that requires global shuffling. In the meantime, our scheme does not affect storage overhead, reliability or repair cost. It turns out that the real request pattern exhibits high skewness and variation, which can significantly benefit from our local block migration with only a few necessary moves.

5.2 System Model and Problem Formulation

The content in a typical cloud storage system is stored in *data blocks*. When an erasure code is used, e.g., a systematic (k, r) RS code, every k original uncoded data blocks are grouped in a *coded group* and another r parity blocks are generated. In order to maintain a high availability, all these $k + r$ blocks are placed on *different* server nodes. We will call them "coded blocks" in general with respect to the k original blocks. In a *normal read*, any access request will be directed to the server containing the original block. If the server is unavailable, a *degraded read* is performed by reading any other k blocks in the same coded group, requiring k server accesses. Suppose the system has a total number of n coded blocks placed on m servers.

In a small unit of time, which we call *time slot* (e.g., a second), we denote the number of requests for each *coded block* i by a random variable D_i. Let $\vec{D} := \{D_1, D_2, \ldots, D_n\}$. With request rates represented by random variables, we can model demand fluctuation in different time slots. We use $\vec{\mu} := \mathbb{E}(\vec{D})$ to denote the mean of $\vec{D}$, and $\mathbf{\Sigma} := \mathbb{COV}(\vec{D})$ the covariance matrix of $\vec{D}$. We can assume that within a certain *measurement period*, $\vec{\mu}$ and $\mathbf{\Sigma}$ remain unchanged.

Note that the mean and covariance of $\vec{D}$ can be readily measured or estimated from system traces. For example, the system can keep track of the number of requests per second or per minute for each *original* content block at a frontend gateway[Calder *et al.*, 2011] to calculate the empirical mean and covariances of request rates for original content in the measurement period. It can also easily record the rate of degraded reads (due to node failures or temporary unavailability) and convert the request rate statistics for original content blocks to those for all the coded blocks, assuming degraded reads are randomly directed to k other coded blocks. Alternatively, the access statistics for all the coded blocks can even be measured directly in the backend storage system. This way, $\vec{\mu}$ and $\mathbf{\Sigma}$ for all coded blocks are directly computed.

We use an integer variable y_i, $i = 1, \ldots, n$ to represent the index of the server on which the i^{th} coded block is placed. Denote $\{L_1, \ldots, L_m\}$ the server loads, where $L_i = \sum_{j:y_j=i} D_j$ represents the amount of requests directed to server i in the time slot of interest. Let $\alpha := k + r$ denote the total number of coded blocks in each coded group. Furthermore, we use G_i, $i = 1, \ldots, n$, to denote the index of the coded group to which the i^{th} coded block belongs; two blocks are in the same coded group if and only if they have the same group index.

Considering a specific time slot, we formulate the optimal content placement (CP) problem as

$$
\begin{aligned}
\text{(CP)} \underset{y_1,y_2,\ldots,y_n}{\text{minimize}} \quad & \mathbb{E}\left(\sum_{i=1}^{m} \frac{1}{2} L_i^2\right) && (5.1)\\
\text{subject to} \quad & L_i = \sum_{j:y_j=i} D_j, \quad \forall i, && (5.2)\\
& y_i \neq y_j, \text{ if } G_i = G_j, \quad \forall i \neq j, && (5.3)\\
& y_i = \{1, 2, \ldots, m\}, \quad \forall i, && (5.4)
\end{aligned}
$$

Problem (CP) minimizes the expected squared l^2-norm of server loads, which represents the expected sum of waiting times of all the requests in this time slot. We assume that the request processing speed of servers are homogeneous[Li *et al.*, 2014*c*], which is common for storage servers in the same rack in a datacenter. The purpose of (5.1) is to distribute random loads $\vec{D}$ across different servers in a statistically balanced manner. Constraint (5.2) is the mapping from request rates to server loads according to content placement $y_1, \ldots, y_n$. Constraint (5.3) requires that the coded blocks from the same coded group must be placed on different servers to guarantee the promised reliability of an RS (k, r) code. We do not consider queue accumulation along multiple time slots in our model and focus on solving the single period problem (CP). Note that server processing capacity is usually over-provisioned in production systems, and once server loads are balanced, queues will vanish fast in a stable system.

We now convert Problem (CP) to an equivalent form similar to the well-known Min-k-Partition Problem in graph theory yet with one additional constraint. We define a weight matrix $\boldsymbol{W}$ by

$$\boldsymbol{W} := \mathbb{E}(\vec{D} \cdot \vec{D}^T) = \vec{\mu} \cdot \vec{\mu}^T + \boldsymbol{\Sigma}, \tag{5.5}$$

Clearly, all the elements in $\boldsymbol{W}$ are **nonnegative**. Consider the following problem, which we call Constrained Min-k-Partition Problem (CMKP+):

$$
\begin{aligned}
\text{(CMKP+)} \underset{y_1,y_2,\ldots,y_n}{\text{minimize}} \quad & \sum_{i<j} \boldsymbol{W}_{ij} \delta(y_i - y_j) + \frac{1}{2} \sum_i \boldsymbol{W}_{ii}, && (5.6)\\
\text{subject to} \quad & y_i \neq y_j, \text{ if } G_i = G_j, \forall i \neq j, && (5.7)\\
& y_i = \{1, 2, \ldots, m\}, \forall i, && (5.8)
\end{aligned}
$$

where $\delta(\cdot)$ is an indicator function, i.e.,

$$\delta(x) := \begin{cases} 1, & \text{if } x = 0, \\ 0, & \text{otherwise.} \end{cases}$$

Note that we use CMKP to represent the problem with the constant term $\frac{1}{2} \sum_i \boldsymbol{W}_{ii}$ removed from (5.6).

Proposition 9 *Problem (CP) is equivalent to Problem (CMKP+).*

Proof. Please refer to the Appendix for the proof. ■

Therefore, we can consider the (CMKP+) problem instead of the original (CP) problem. In fact, (CMKP+) is a partition problem in graph, where all the coded blocks can be deemed as nodes, with $\boldsymbol{W}$ representing edge weights between every pair of nodes. The objective is to divide nodes into k partitions to minimize the sum of intra-partition edge weights, subject to constraint (5.7), that is, no coded blocks from the same coded group appear in the same partition. Without constraint (5.7), the (CMKP+) problem can be converted to Min-k-Partition (MKP) and Max-k-Cut (MKC), which are well-known NP-complete problems[Kann *et al.*, 1997; Karp, 1972]. However, our problem (CMKP+) is even more challenging due to the additional constraint (5.7) to maintain the promised reliability offered by erasure coding.

5.3 Local Block Migration Algorithm

We present the local block migration (LBM) algorithm to solve (CMKP+) with theoretical worst-case approximation guarantees, which equivalently solves the optimal content placement problem (CP). We first present our technique to handle the special constraint (5.7) before presenting the algorithm.

5.3.1 Problem Reduction

First, we reduce (CMKP+) to a form without constraint (5.7). Our idea is to solve the problem with constraint (5.7) removed, while setting a *sufficiently large* weight for each pair of coded blocks in the same coded group to prevent them from being placed on the same server. Define $\boldsymbol{W'}$ as

$$\boldsymbol{W'}_{ij} = \begin{cases} f_{ij}(\boldsymbol{W}) & , \text{if } G_i = G_j, i \neq j, \\ \boldsymbol{W}_{ij}, & \text{otherwise}, \end{cases} \tag{5.9}$$

where $f_{ij}(\boldsymbol{W})$ is a penalty function to be defined later. Replacing $\boldsymbol{W}$ by $\boldsymbol{W'}$ and removing constraint (5.7) in (CMKP+), we obtain

$$\text{(MKP+)} \underset{y_1,y_2,\ldots,y_n}{\text{minimize}} \quad \sum_{i<j} \boldsymbol{W'}_{ij}\delta(y_i - y_j) + \frac{1}{2}\sum_i \boldsymbol{W'}_{ii}, \tag{5.10}$$

$$\text{subject to} \quad y_i = \{1, 2, \ldots, m\}, \text{ for } \forall i. \tag{5.11}$$

Note that the term $\frac{1}{2}\sum_i \boldsymbol{W'}_{ii}$ in (5.10) is a constant. Hence, (MKP+) is equivalent to Min-k-Partition Problem (MKP):

$$\text{(MKP)} \underset{y_1,y_2,\ldots,y_n}{\text{minimize}} \quad \sum_{i<j} \boldsymbol{W'}_{ij}\delta(y_i - y_j), \tag{5.12}$$

$$\text{subject to} \quad y_i = \{1, 2, \ldots, m\}, \text{ for } \forall i, \tag{5.13}$$

whose dual problem is the famous Max-k-Cut (MKC) problem:

$$\text{(MKC)} \underset{y_1,y_2,\ldots,y_n}{\text{maximize}} \quad \sum_{i<j} \boldsymbol{W'}_{ij}(1-\delta(y_i-y_j)), \tag{5.14}$$

$$\text{subject to} \quad y_i = \{1,2,\ldots,m\}, \text{ for } \forall i. \tag{5.15}$$

Furthermore, (MKC) and (MKP) also have the same optimal solution(s). The reason is that the sum of the objective values of the two problems is

$$\sum_{i<j} \boldsymbol{W'}_{ij}\delta(y_i-y_j) + \sum_{i<j} \boldsymbol{W'}_{ij}(1-\delta(y_i-y_j)) = \sum_{i<j} \boldsymbol{W'}_{ij}, \tag{5.16}$$

which is a constant. Since they are minimization and maximization problems, respectively, they will have the same optimal solution(s).

In the following, to solve (CMKP+), we carefully design a penalty function f, such that we always get a feasible solution to the original problem (CMKP+) by solving (MKC) with a new $\boldsymbol{W'}$ yet without constraint (5.7). We propose an algorithm to solve (CMKP+) and thus the original (CP) problem, by approximately solving (MKC) using a classical *local search heuristic*[Kernighan and Lin, 1970; Fiduccia and Mattheyses, 1982; Zhu *et al.*, 2013]. We are able to theoretically derive a worst-case approximation ratio of the proposed solution to our problem (CMKP+), which did not appear in prior literature[Kernighan and Lin, 1970; Fiduccia and Mattheyses, 1982; Zhu *et al.*, 2013], by leveraging a unique problem structure in our objective function.

5.3.2 Local Block Migration Algorithm

Considering the influence of every single move on the objective, we define the *gain* of moving block i to server j as

$$g_j(i) := \sum_{k:y_k=y_i;k\neq i} \boldsymbol{W'}_{ik} - \sum_{k:y_k=j;k\neq i} \boldsymbol{W'}_{ik}, \forall i,j : j \neq y_i, \tag{5.17}$$

which is the reduction of the objective of (MKP) if this move is carried out. For consistency, let $g_j(i) = -\infty$ for $j = y_i$. Define $f^l_{ij}(\boldsymbol{W})$ as

$$f^l_{ij}(\boldsymbol{W}) := \epsilon + \frac{1}{m-\alpha+1}\min\Big\{\sum_{k:k\neq i;G_k\neq G_i} \boldsymbol{W}_{ik}, \sum_{k:k\neq j;G_k\neq G_j} \boldsymbol{W}_{kj}\Big\}, \tag{5.18}$$

where ϵ is an *arbitrary* positive constant.

Algorithm 3 describes our Local Block Migration (LBM) algorithm to solve (CMKP+). In every iteration, we execute the move of block i to server j who achieves the largest *gain* $g_j(i)$, until no move can reduce the objective any more, as shown in Lines 6-9.

To pick the best move $\text{argmax}_{\{i,j\}} g_j(i)$, in Line 8 of Algorithm 3, we do not need to recalculate all the $m \times n$ $g_j(i)$ by (5.17) in every iteration. Instead, since there is only one move in each iteration, we only need to update the gains $g_j(i)$ affected by the move.

Algorithm 3 Local Block Migration

```
1: **Input** initial placement $\{y_1, y_2, \ldots, y_n\}$, $\boldsymbol{W}$.
2: **Output** migrated placement $\{y_1, y_2, \ldots, y_n\}$.
3: Calculate $\boldsymbol{W}'$ by (5.9) and (5.18)
4: Calculate $g_j(i)$ for $\forall i, j$ by (5.17)
5: procedure LOCAL BLOCK MIGRATION
6:     while $\max_{\{i,j\}} g_j(i) > 0$ do
7:         Find the best move: $i_m, j_m \leftarrow \text{argmax}_{\{i,j\}} g_j(i)$
8:         Update all the affected $g_j(i)$ entries by Algorithm 4
9:         Execute the move: $y_{i_m} \leftarrow j_m$
```

Moreover, even the affected gains $g_j(i)$ do not need to be recalculated by (5.17), and can be updated by incrementally. The details of our efficient procedure to update the gains $g_j(i)$ is described in Algorithm 4.

Note that LBM is a "local" algorithm that performs one best move at a time to improve the latency performance. In real systems, since it is impractical to globally shuffle all the block placement to optimize load balancing, we can use the proposed LBM to locally improve an existing arbitrary placement at some frequency, e.g., every hour or every day. Moreover, we do not actually carry out all the moves $y_{i_m} \leftarrow j_m$ computed by Algorithm 3. Instead, we only make the moves to change the initial placement to the LBM outcome. In Sec. 5.4, we show that only a few moves will achieve the most latency reduction.

5.3.3 Feasibility and Worst-Case Approximation Ratio

We first show Algorithm 3 yields a feasible solution to our problem with the special constraint (5.7).

Theorem 10 *If $\boldsymbol{W}'$ in (5.9) is defined with $f_{ij}(\boldsymbol{W})$ given by $f_{ij}(\boldsymbol{W}) = f^l_{ij}(\boldsymbol{W})$ in (5.18), any solution given by Algorithm 3 will satisfy (5.7), and thus will be a feasible solution to (CMKP+) and (CP).*

Proof. Please refer to the Appendix for the proof. ■

Note that we do not necessarily have $f_{ij}(\boldsymbol{W}) > \boldsymbol{W}_{ij}$. However, Theorem 10 guarantees that if $f_{ij}(\boldsymbol{W}) = f^l_{ij}(\boldsymbol{W})$, LBM always produces a feasible solution that no two blocks from the same coded group are placed on the same server. Theorem 11 provides a worst case approximation for our Local Block Migration with respect to the globally optimal solution to (CP).

Theorem 11 *Suppose $f_{ij}(\boldsymbol{W})$ in (5.9) is given by $f^l_{ij}(\boldsymbol{W})$ in (5.18). Then, the worst-case approximation ratio of Algorithm 3 with respect to the optimal solution of (CMKP+) and (CP) is given by*

$$1 + \frac{1}{m - \alpha + 1}\left(\frac{\mathbb{E}\big(\big(\sum_i D_i\big)^2\big)}{\sum_i \mathbb{E}(D_i^2)} - 1\right). \tag{5.19}$$

Proof. Please refer to the Appendix for the proof. ■

Furthermore, since we have

$$\frac{\mathbb{E}\big(\big(\sum_i D_i\big)^2\big)}{\sum_i \mathbb{E}(D_i^2)} \leq \max_{\vec{D}}\big\{\big(\sum_i D_i\big)^2 / \sum_i D_i^2\big\} \leq n,$$

where the equality holds if and only if $D_1 = D_2 = \ldots = D_n$, we can derive the worst-case ratio among all the distributions of $\vec{D}$.

Corollary 12 *For any $\vec{D}$, the approximation ratio given by Theorem 11 is at most $1 + \frac{n-1}{m-\alpha+1}$.*

Remarks: the approximation ratio provided by Theorem 11 is dependent on the distribution of the requests $\vec{D}$. In the extreme case when requests for different coded blocks are identical, i.e., $D_1 = D_2 = \ldots = D_n$, the offered approximation ratio (5.19) is large as shown in Corollary 12. In fact, the ratio of $\mathbb{E}(||\vec{D}||_1^2)/\sum_i \mathbb{E}(D_i^2)$ characterizes the demand variation among different blocks. When this variation is large, the approximation ratio (5.19) is small and our algorithm is guaranteed to yield a good result even in the worst case.

On the other hand, when the demand variation is small, although the offered theoretical *worst-case* performance bound (5.19) is large, LBM can actually still generate a load balanced solution. In fact, in this case, D_i behaves uniformly cross different blocks and simple randomized or round robin placement can already achieve load balancing, so can LBM. In a nutshell, LBM provides good solutions for most situations and is especially beneficial when the requests for different blocks have a large variation and are highly skewed. In Sec. 5.4, we show that our request traces in the real world usually have a small $\frac{\mathbb{E}(||\vec{D}||_1^2)}{\sum_i \mathbb{E}(D_i^2)}$, in which case LBM will have a large benefit.

5.3.4 Further Reducing Migration Overhead

Although Theorem 10 guarantees the feasibility of the final converged solution from Algorithm 3, in reality, we may want to stop looping after a fixed number of iterations to limit the number of moves produced by LBM. In this case, the solution may not be feasible to (CP) in theory with the $f_{ij}(\boldsymbol{W})$ definition in (5.18). In order to propose an alternative scheme, we let $f_{ij}(\boldsymbol{W})$ in (5.9) be given by

$$f_{ij}^r(\boldsymbol{W}) := \epsilon + \frac{1}{m-\alpha}\max\Big\{\sum_{k:k\neq i;G_k\neq G_i} \boldsymbol{W}_{ik}, \sum_{k:k\neq j;G_k\neq G_j} \boldsymbol{W}_{kj}\Big\}, \tag{5.20}$$

where ϵ is an *arbitrary* positive constant.

Theorem 13 *If the $\boldsymbol{W}'$ in (5.9) is defined with $f_{ij}(\boldsymbol{W})$ given by $f_{ij}(\boldsymbol{W}) = f_{ij}^r(\boldsymbol{W})$ in (5.20), and the initial content placement satisfies (5.7), the solution after any iteration in Algorithm 3 will always satisfy (5.7).*

Algorithm 4 Gain Update Algorithm

```
Require: the current $g_j(i)$, $\forall i, j$, the current moving block index $i_m$ and its destination server $j_m$.
Ensure: the updated $g_j(i)$, $\forall i, j$.
1: procedure $g$ UPDATE
2:     for $\forall i \neq i_m$, such that $y_i = y_{i_m}$ do
3:         for $\forall j \neq y_{i_m}$ do
4:             $g_j(i) \leftarrow g_j(i) - \boldsymbol{W}'_{ii_m}$
5:     for $\forall i \neq i_m$, such that $y_i = j_m$ do
6:         for $\forall j \neq j_m$ do
7:             $g_j(i) \leftarrow g_j(i) + \boldsymbol{W}'_{ii_m}$
8:     for $\forall i \neq i_m$, $y_i \neq y_{i_m}$ do
9:         $g_{y_{i_m}}(i) \leftarrow g_{y_{i_m}}(i) + \boldsymbol{W}'_{ii_m}$
10:    for $\forall i \neq i_m$, $y_i \neq j_m$ do
11:        $g_{j_m}(i) \leftarrow g_{j_m}(i) - \boldsymbol{W}'_{ii_m}$
12:    for $\forall j, j \neq y_{i_m}$ and $j \neq j_m$ do
13:        $g_j(i_m) \leftarrow g_j(i_m) + \sum_{i:i\neq i_m;y_i=j_m} \boldsymbol{W}'_{i_m i} - \sum_{i:i\neq i_m;y_i=y_{i_m}} \boldsymbol{W}'_{i_m i}$
14:    Calculate $g_{y_{i_m}}(i_m)$ by (5.17)
15:    $g_{j_m}(i_m) \leftarrow -\infty$.
```

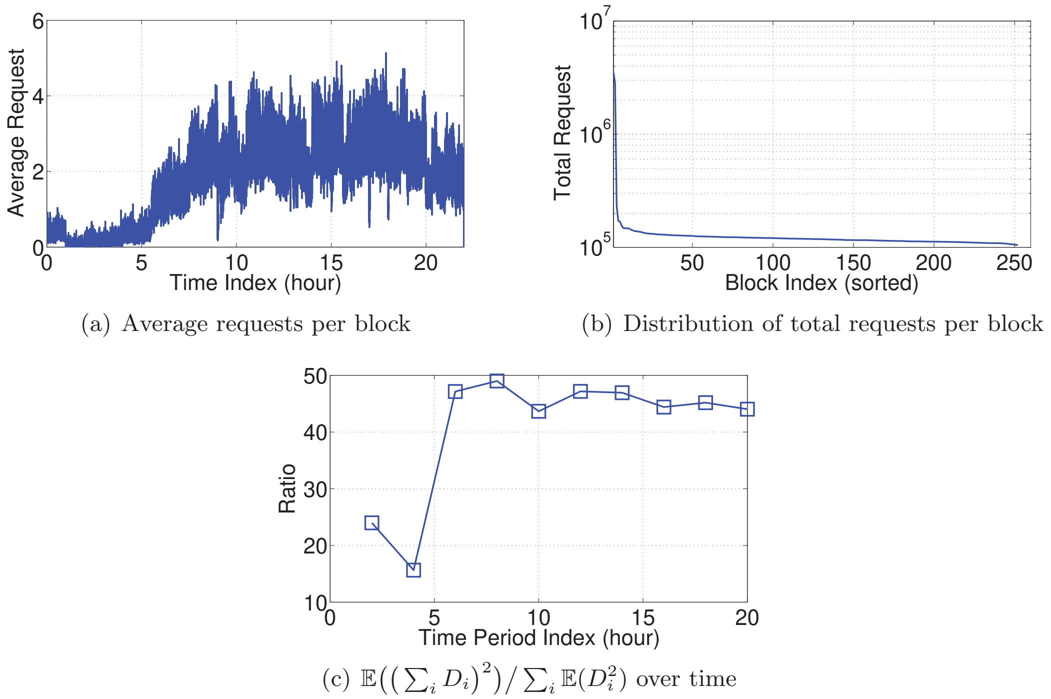
Proof. Please refer to the Appendix for the proof. ■

Remarks: Theorem 13 implies that as long as we start from a valid placement, we can put a maximum iteration number in LBM and can always get feasible solutions in any iteration. This way, we can stop the loop in Algorithm 3 when the maximum iteration number is reached and still get a feasible solution that satisfies constraint (5.7). In other words, $f^r_{ij}(\boldsymbol{W})$ allows us to trade the latency reduction off for fewer block moves, according to a budget on migration overhead.

5.3.5 Time Complexity and the Breakout Method

Algorithm 3 runs in linear time with respect to the number of coded blocks in each iteration and is very efficient. In the main loop from Line 6 to Line 9 in Algorithm 3, it only contains a finding max operation and an updating operation. The finding max runs in linear time with respect to the searching space and it is $O(mn)$ since we have mn gain entries. For the gain updating procedure described in Algorithm 4, Line 2 to Line 7 has only $O(\frac{2n}{m} \cdot m) = O(2n)$ additions or subtractions. Line 8 to Line 11 also has $O(2n)$ additions or subtractions. Line 12 to Line 15 has two updating entries with $O(\frac{2n}{m})$ basic calculations. Therefore, our Local Block Migration can finish each iteration with $O(mn)$ basic calculations.

The LBM is a local heuristic search and may get trapped into some local optimum. In order to reach the global optimum, some breakout method[Morris, 1993] can be engaged. Similar techniques in[Kernighan and Lin, 1970; Fiduccia and Mattheyses, 1982] can be used to even improve over the local optimum. The idea is that when a local optimum is reached,



(a) Average requests per block

(b) Distribution of total requests per block

(c) $\mathbb{E}\big(\big(\sum_i D_i\big)^2\big)/\sum_i \mathbb{E}(D_i^2)$ over time

Figure 5.1: The properties of the trace data collected from Windows Azure Storage (WAS).

we may keep looping in Algorithm 3 even if the max value of $g_j(i)$ is negative. To avoid infinite loops, the blocks that have been moved are locked. When all the blocks are moved for once, the history of all the moves are inspected and the placement in the history with best performance is picked. If it is better than the former converged local solution, a better solution is produced and a new round of the local search in Algorithm 3 is started from the new solution.

The time complexity of the escaping method is $O(mn^2)$. Although it will usually come to a better solution, it needs lots of block moves, which results in high system overhead. Moreover, as we will show in Sec. 5.4, the improvement of the breakout extension is limited. Therefore, our proposed LBM is enough to produce good solutions without the breakout method.

5.4 Simulation

We conduct simulations driven by real workload traces collected from a production cluster in the Windows Azure Storage (WAS) system. It contains the request traces of 252 equal-sized original data blocks *in every second* for a 22-hour period. We adopt a systematic $(6, 3)$ RS code, and the blocks will be placed on 20 server nodes. We assume that 5% of all requests may encounter block unavailability events, which happen randomly. During degraded read when the original block is unavailable, the requests are randomly directed to the combinations of nodes that can reconstruct the original block.

The properties of the data are demonstrated in Fig. 5.1. Fig. 5.1(a) is the average request per block at different times. Fig. 5.1(b) shows the distribution of the total number of requests for each block. Fig. 5.1(c) is the statistical value of $\mathbb{E}((\sum_i D_i)^2)/\sum_i \mathbb{E}(D_i^2)$ for each 2-hour measurement period, which will influence the worst-case performance of our algorithm according to Theorem 11. We can see that it is no greater than 50 and leads to an approximation ratio of 5.08.

We first evaluate our Local Block Migration algorithm in terms of the reduction on the objective function in different measurement periods. To mimic the behavior in real systems, we have also conducted a round-based simulation to replay the real request traces, considering queue accumulation over different time periods.

5.4.1 Performance of the LBM Optimization Algorithm

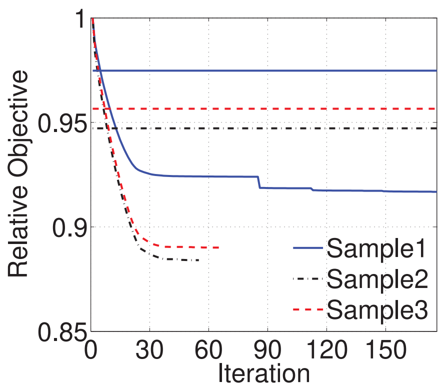
We test our algorithm in each 2-hour measurement period and present results from 3 typical sample periods in Fig. 5.2. Assuming a 5% block unavailability rate and random load direction for degraded reads, we can get the empirical mean and covariance matrix of all the 378 *coded blocks* in each 2-hour period. We perform Local Block Migration (LBM) to find the optimal block placement in each sample period, and also adopt the breakout method after the LBM algorithm converges. We compare our algorithm to choosing the best out of 1000 random placements (which requires shuffling and thus lots of block moves).

Fig. 5.2 shows how the LBM reduces the objective function in (5.1) from a random initial placement as well as the number of block moves incurred. In Fig. 5.2(a), the decreasing curves indicate the performance of the LBM while the horizontal lines are the reference performance of the best out of 1000 random placement. For each 2-hour sample period, the breakout takes place at the iteration where there is a sudden increase of moves in Fig. 5.2(b).

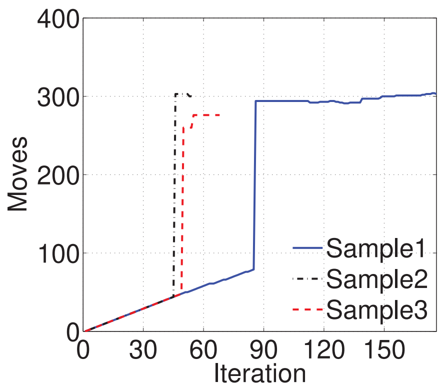
We can see that LBM can effectively reduce the optimization objective and clearly outperforms the global optimal approximation provided by the best out of 1000 random placements. More importantly, in LBM, most of the reduction on the objective is achieved within 30 iterations (thus 30 block moves) while choosing the best out of 1000 random placements will typically incur a large overhead of more than 350 moves. Furthermore, although the breakout can further reduce the objective, such further reduction is limited. Given that the breakout will incur a large number of additional moves, we can gain most of the benefit from LBM by running up to a certain number of iterations without the breakout extension.

5.4.2 Queuing Simulation with Trace Replay

Now we replay the request traces in a round-based simulator (each round being one second) and simulate the queuing effect of the system under different placement strategies. Again, we divide the 22-hour trace into 11 2-hour measurement periods. For each measurement period, we adjust the block placement by LBM with the maximum iteration set to 20, using



(a) Objective v.s. Iteration



(b) Necessary Moves v.s. Iteration

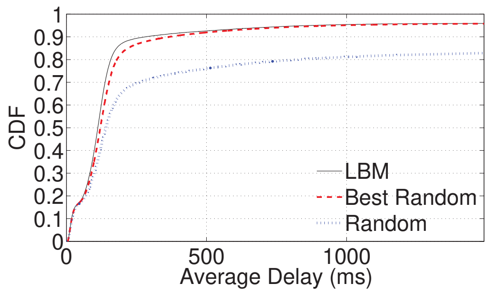
Figure 5.2: The performance of the Local Block Migration Algorithm with breakout extension. The horizontal lines in Fig. 5.2(a) indicate the reference performance of the best of 1000 random placement and the decreasing curves indicate the performance from the LBM. For each sample, the performance without breakout extension can be spot on iterations before the "sudden" rise of the moves in Fig. 5.2(b).

the request statistics from the previous 2-hour measurement period as the prediction of the request rates in the current measurement period. We feed the requests per second into the simulator assuming block unavailability happens randomly with a ratio of 5%. The request processing rate of each server is set such that the peak demand will utilize 70% of server capacity.

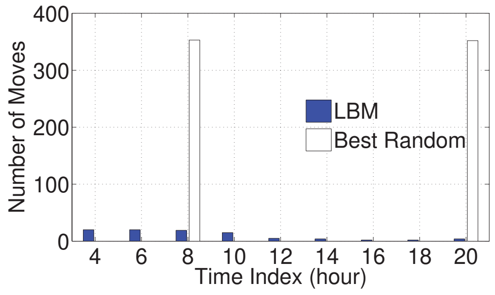
We initialize the placement by applying LBM to the statistics of the first 2-hour measurement period and evaluate the queuing performance in the remaining 10 measurement periods, where LBM is applied at the beginning of each 2-hour measurement period. We compare LBM to two schemes. One is the pure fixed random placement, which is a typical method adopted in industry today[Borthakur, 2008]. The other is dynamically adjusting the placement in each 2-hour measurement period by picking the best out of 1000 random placements, which we call the Best Random.

Fig. 5.3(a) plots the distribution of the average request delay, while Fig. 5.3(b) shows the number of block moves during each 2-hour measurement period. We can see that LBM can greatly reduce the access delay over the typical random placement. LBM can even beat the Best Random with only a small number of block moves in each measurement period, while the Best Random always needs lots of global shuffling to keep the placement optimal, bringing about high migration overhead, and is thus impractical. Fig. 5.3(c) shows the performance of LBM under different server processing rates, characterized by different levels of peak demand utilization.

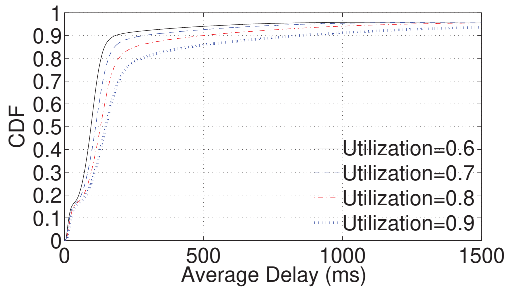
Fig. 5.4 shows the average queue length of all the servers as time goes. We can see that with LBM, the queues will generally be stable, while the random placement will suffer under peak demands due to unbalanced server loads and poor utilization of server capacities.



(a) Average request delay distribution



(b) Number of block moves over time



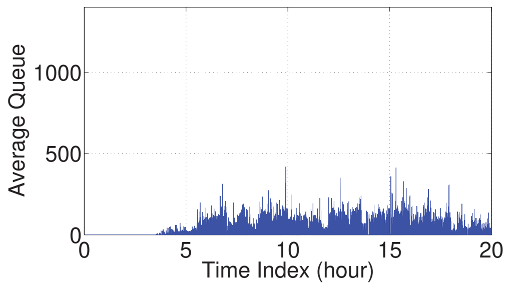
(c) Average delay (ms) under different utilization

Figure 5.3: Performance of LBM, Best Random (the best out of 1000 random placements), and Random placement in the round-based trace replay.

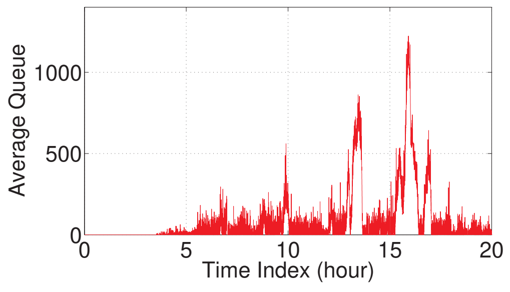
5.5 Related Work

Abundant works are on enhancing the storage overhead and reducing the recover cost for the degraded reads. In[Huang *et al.*, 2012], Local Reconstruction Code (LRC) is proposed to reduce the storage overhead. The works in[Zhu *et al.*, n.d.; Khan *et al.*, 2012] focus the optimization of the degraded reads and better load direction scheme to boost the performance of the degraded reads was presented. M. Xia et al. in[Xia *et al.*, 2015] use two different erasure codes that dynamically adapt to system load to achieve both the overall low storage overhead and low recovery cost. HitchHiker[Rashmi *et al.*, 2014] propose a new encoding technique to improve recovery performance.

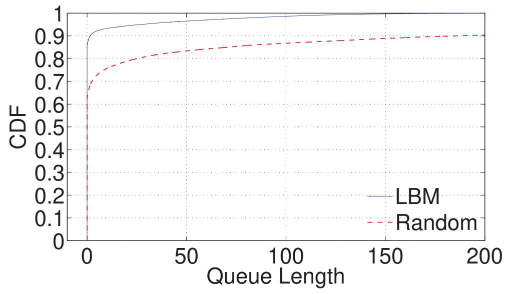
There are extensive works around the content placement problem in replication based systems with different desired QoS. In[Rochman *et al.*, 2013], Rochman et al. propose the strategies of placing the resources to distributed regions to serve more requests locally. In[Xu and Li, 2013], Xu et al. propose a reasonable request mapping and response routing scheme to maximize the total utility of serving requests minus the cost. Bonvin et al.[Bonvin *et al.*, 2010] propose a distributed scheme to dynamically allocate the resources of a data cloud based on net benefit maximization regarding the utility offered by the partition and its storage and maintenance cost. In[Agarwal *et al.*, 2010], the automatic data placement across geo-distributed datacenters is presented, which iteratively moves a data item closer to both clients and the other data items that it communicates with. B. Yang et al.[Yu



(a) Local Block Migration



(b) Random



(c) Distribution of queue lengths

Figure 5.4: The simulated queue lengths in each time slot (second).

and Pan, 2015] study the content placement problem for systems when multiple items are needed in each request and the item size is small. They try to maximize the correlation of the contents stored on the same server to reduce the IO and the CPU overhead to fulfill a request at a time. On the contrary, our work focus on the applications in which the size of content block is large and each request only relates to one block. And we study the content placement for erasure coded systems.

In[Liang and Kozat, 2014; Sun *et al.*, 2015] new parallel download scheme to optimize the delay performance of coded storage are proposed. Their work rely on parallel downloads to leverage a queueing-theoretical gain, where each request must access k or more servers, and abort the remaining download threads when k blocks are obtained.

In the line of the mathematical technique, the local search idea to solve the Max-2-Cut Problem is first proposed in[Kernighan and Lin, 1970] . It is improved with better efficiency in[Fiduccia and Mattheyses, 1982]. W. Zhu et al.[Zhu *et al.*, 2013] extend it to solve the mathematical Max-k-Cut Problem. In our work, we take the local search idea to solve the problem with challenging special constraints related to the real application and we provide a linear time searching scheme.

5.6 Summary

In this chapter, we study the problem of reducing access latency in erasure coded storage systems through block migration and content placement optimization. Based on request rate statistics, we have built a model to minimize the expected request waiting times, which is similar to the NP-Complete Min-k-Partition problem with a special additional constraint. We propose Local Block Migration which moves the block that reduces the latency objective the most at a time. We theoretically characterize the algorithm's worst-case approximation ratio, which depends on a demand variation measure across blocks. Through trace-driven simulations based on request traces from Windows Azure Storage, we show that in the presence of skewed demands, Local Block Migration can significantly reduce the access latency in erasure coded storage by only moving a few blocks once in a while, without global shuffling. Furthermore, the computation of such desired moves can be done within 1 second for 252 original blocks stored with a $(6, 3)$ RS code on tens of servers.

Chapter 6

Conclusion Remarks

6.1 Conclusion

In this dissertation, we demonstrate our strategy of applying optimization methods with privacy and efficiency concerns in two different cases.

Firstly, we study machine learning over distributed features (vertically partitioned data) where none of the parties shall share the local data. We propose the feature distributed machine learning (FDML) to handle this problem. We propose an asynchronous SGD algorithm to solve the new FDML scenario, with a convergence rate of $O(1/\sqrt{T})$, T being the total number of iterations, matching the existing convergence rate known for data-parallel SGD in a stale synchronous parallel setting[Ho *et al.*, 2013]. We also propose the parallel ADMM sharing algorithm, which is more robust during training and scalable to the number of features. We have shown the convergence for convex and non-convex loss functions. To further protect the data privacy, we apply the differential privacy technique in the training procedure. Extensive experiments conducted over real data trace from *Tencent* and public source on prototype systems show that our proposed method yields better prediction accuracy than model trained from only local features while ensuring a certain level of differential privacy guarantee.

Secondly, we study the tail latency reduction and load balancing problem for erasure coded cloud storage systems, which often suffers from long access latency tails. We propose several schemes to direct loads based on either per-request decisions made from instantaneously probed storage node queue status or an optimized load direction table computed by statistical optimization with lazy queue status probes. We implemented a prototype system and deployed it on a cluster of 98 machines to evaluate the performance based on a large amount of real-world traces. Results suggest that the proposed least-marginal-load-first policy based on instantaneous sampled queue status can reduce the median request latency by more than 40% and the 95-percentile tail latency by more than 75% in both RS-coded systems and LRC-based systems, as compared to the existing approach of normal reads followed by passive degraded reads upon timeouts. The statistical optimization approach with lazy queue probing can also significantly reduce request and task latencies with a much lower system probing overhead. We also study the problem of reducing access latency in erasure coded storage systems through block migration and content placement optimization. Based on request rate statistics, we have built a model to minimize the expected request waiting times, which is similar to the NP-Complete Min-k-Partition problem with a special additional constraint. We propose Local Block Migration which moves the block that reduces the latency objective the most at a time. We theoretically characterize the algorithm's worst-case approximation ratio, which depends on a demand variation measure across blocks. Through trace-driven simulations based on request traces from Windows Azure Storage, we show that in the presence of skewed demands, Local Block Migration can significantly reduce the access latency in erasure coded storage by only moving a few blocks once in a while, without global shuffling. Furthermore, the computation of such desired moves can be done within 1 second for 252 original blocks stored with a $(6, 3)$ RS code on tens of servers.

6.2 Future Directions

6.2.1 Privacy and Distributed Machine Learning

Insuring privacy in a distributed machine learning system is an important direction to fully utilize the power of data and still needs development in the current stage, especially for the case with distributed features we study in this dissertation.

The first concern is how to incorporate more interaction between features from different parties into the prediction model. In this dissertation, our proposed FDML only combined the local intermediate predictions in the final stage, but little interaction between the features from different parties is extracted, which limits the potential power of the rich features from other parties. However, to incorporate more interaction, we have to transmit more intermediate results, which makes it more challenging to ensure the privacy. Some naive solutions could be to generate some intermediate local results with more dimensions rather than only a single local prediction. For example, each party could generate a vector of local predictions and on the central server, we can apply several layers of networks to combine those results. In this way, the SGD based algorithm still work. Moreover, the widely used factorization machine model might also be trained here by maintaining and transmitting a vector of local predictions and utilizing some algebraic tricks.

The second concern is how to reduce the communication overhead. Nowadays, the bandwidth within a cluster of data server are more than enough to carry heavy machine learning tasks. However, bandwidth between data centers across the WAN could be limited. Moreover, bandwidth will always be an issue when collaborating with increasing amount of personal devices with unstable network connections. One direction is to follow the existing common approaches and adapt them to the feature distributed machine learning setting. These include thresholding (only transmit the value exceeding given value and transmit the vectors in sparse key value pair form), clipping and quantization (use less precision floating point data or apply some coding technique to code the more frequent values into shorter length). The other direction is to explore the special model structure in this problem. Specifically, we can explicitly model and monitor the influence of different intermediate values to the final loss, make a local cache to the past values and only update the important values more frequently while lazily transmitting the remaining ones. There could be abundant of approaches to explore the appropriate strategy to monitor the influence and optimize the updating frequency for different values.

The third concern is theoretical research, especially on the tight relation among the privacy guarantee, the model accuracy and convergence rate. In this dissertation, we have not pulled out a complete analysis combining all those factors. Instead, we have only derived partial results where some factors are ignored. Specifically, in SGD based method, we only derived the convergence rate but we missed the privacy guarantee. In ADMM sharing based method, we derived the convergence result and the pure privacy guarantee separately, without a complete analysis combining both factors. Although the proposed methods seem to work in the experiments, it is still worth studying the tight theoretical bound in order to convince the people about the privacy guarantee and motivate them to share data.

6.2.2 Load Balancing in Erasure Coded Storage Systems

Our work is only the first step of applying the optimization technique in tail reduction and load balancing problem in the erasure coded storage systems. We can envision a few important directions that further improve our work.

One of the possible important directions is to incorporate data driven methods and combine the load prediction with optimization for the load direction problem and the content placement problem. For load direction, we only explored a varieties of methods to strike a balance among the optimization accuracy, computation complexity and system overhead due to probing. We were directing the load solely based on the current queuing and request status, without considering the future load. And due to the probing overhead, we were doing a caching approach or sampling approach to make it scalable to request number, which adds more inaccuracy to the results. For future work, instead of the probing and optimization, we can collect the request statistics and other context information like time, request information, server cpu and memory status, etc. to predict both the request and queuing status in the near future. With those prediction model, we could expect a more intelligent load direction strategy and such prediction might also release the burden of frequent probing overhead. For the content placing problem, our method only considered the past statistics and used it as the estimation for the future. However, we can collect more context information and keep a load prediction model to predict the future requests. With such information, we can expect the optimization solution to yield better results. Besides, since the migration itself incurs extra reading and writing overhead, we can wisely schedule the execution of migration in the predicted request valley time.

Chapter 7

Proofs

7.1 Supplementary Materials for Chapter 2

7.1.1 Proof of Proposition 1

By the proposed algorithm and from (2.8), we have$x_{t+1}^j = x_t^j - \eta_t \nabla^j F(\tilde{x}_t(j))$, where $\tilde{x}_t(j)$ is the concatenated model parameters with staleness in which $\tilde{x}_t^i(j) = x_{t-\tau^j(i)}^i$. Note that we always have $\tau^j(i) \le \tau, \forall i, j$. To help proving the proposition, we first prove a lemma.

Lemma 14

$$< x_t - x_*, \nabla F_t(x_t) >= \frac{1}{2}\eta_t \sum_{j=1}^m \|\nabla^j F(\tilde{x}_t(j))\|^2 - \frac{D_{t+1} - D_t}{\eta_t}$$
$$+ \sum_{j=1}^m < x_t^j - x_*^j, \nabla^j F_t(x_t) - \nabla^j F_t(\tilde{x}_t(j)) > . \tag{7.1}$$

Proof. $D_{t+1} - D_t$

$$=\frac{1}{2}\sum_{j=1}^m \big(\|x_t^j - \eta_t \nabla^j F(\tilde{x}_t(j)) - x_*^j\|^2 - \|x_t^j - x_*^j\|^2\big)$$
$$=\sum_{j=1}^m \big(\frac{1}{2}\|\eta_t \nabla^j F(\tilde{x}_t(j))\|^2 - \eta_t < x_t^j - x_*^j, \nabla^j F(\tilde{x}_t(j)) > \big)$$
$$=\frac{1}{2}\eta_t^2 \sum_{j=1}^m \|\nabla^j F(\tilde{x}_t(j))\|^2 - \eta_t < x_t - x_*, \nabla F(x_t) >$$
$$+ \eta_t \sum_{j=1}^m \langle x_t^j - x_*^j, \nabla^j F(x_t) - \nabla^j F(\tilde{x}_t(j)) > . \tag{7.2}$$

Dividing the above equation by η_t, we can get the lemma. □

Another important fact for our analysis is

$$\sum_{t=a}^b \frac{1}{\sqrt{t}} \le \int_{a-1}^b \frac{1}{\sqrt{t}} dt = 2(\sqrt{b} - \sqrt{a-1}). \tag{7.3}$$

We now come to evaluate the regret R up to iteration T. By the definition in (2.10) and, we have

$$R = \frac{1}{T}\sum_t F_t(x_t) - F(x_*) = \frac{1}{T}\sum_{t=1}^T F_t(x_t) - \frac{1}{T}\sum_{t=1}^T F_t(x_*) \tag{7.4}$$

$$= \frac{1}{T}\sum_{t=1}^T \big(F_t(x_t) - F_t(x_*)\big) \leq \frac{1}{T}\sum_{t=1}^T < x_t - x_*, \nabla F_t(x_t) > \tag{7.5}$$

where (7.5) follows from the convexity of the loss functions. Inserting the result from lemma 14, we can get

$$\begin{aligned} T\cdot R \leq & \sum_{t=1}^T \big(\frac{1}{2}\eta_t \sum_{j=1}^m \|\nabla^j F(\tilde{x}_t(j))\|^2 - \frac{D_{t+1}-D_t}{\eta_t} \\ & + \sum_{j=1}^m < x_t^j - x_*^j, \nabla^j F_t(x_t) - \nabla^j F_t(\tilde{x}_t(j)) > \big) \\ = & \sum_{t=1}^T \frac{1}{2}\eta_t \sum_{j=1}^m \|\nabla^j F(\tilde{x}_t(j))\|^2 - \sum_{t=1}^T \frac{D_{t+1}-D_t}{\eta_t} \\ & + \sum_{t=1}^T\sum_{j=1}^m < x_t^j - x_*^j, \nabla^j F_t(x_t) - \nabla^j F_t(\tilde{x}_t(j)) > . \end{aligned} \tag{7.6}$$

We look into the three terms of (7.6) and bound them.

For the first term, we have

$$\sum_{t=1}^T \frac{1}{2}\eta_t \sum_{j=1}^m \|\nabla^j F(\tilde{x}_t(j))\|^2 \leq \sum_{t=1}^T \frac{1}{2}\eta_t m G^2 = \sum_{t=1}^T \frac{1}{2}\frac{\eta}{\sqrt{t}} m G^2 \leq \eta m G^2 \sqrt{T}. \tag{7.7}$$

For the second term, we have

$$-\sum_{t=1}^T \frac{D_{t+1}-D_t}{\eta_t} \ = \frac{D_1}{\eta_1} - \frac{D_{t+1}}{\eta_t} + \sum_{t=2}^T D_t\left(\frac{1}{\eta_t} - \frac{1}{\eta_{t-1}}\right) \tag{7.8}$$

$$\leq \frac{D^2}{\eta} - 0 + \sum_{t=2}^T \frac{D^2}{\eta}\left(\sqrt{t} - \sqrt{t-1}\right) = \frac{D^2\sqrt{T}}{\eta}. \tag{7.9}$$

Finally we come to the third term. We have

$$\begin{aligned}
&\sum_{t=1}^{T}\sum_{j=1}^{m} < x_t^j - x_*^j, \nabla^j F_t(x_t) - \nabla^j F_t(\tilde{x}_t(j)) > \\
\leq &\sum_{t=1}^{T}\sum_{j=1}^{m} \|x_t^j - x_*^j\| \cdot \|\nabla^j F_t(x_t) - \nabla^j F_t(\tilde{x}_t(j))\| && (7.10)\\
\leq &\sum_{t=1}^{T}\sum_{j=1}^{m} \|x_t^j - x_*^j\| \cdot L_j \|x_t - \tilde{x}_t(j)\| && (7.11)\\
\leq &\sum_{t=1}^{T}\sum_{j=1}^{m} L_j \|x_t^j - x_*^j\| \cdot \sum_{i=1}^{m} \|x_t^i - x_{t-\tau^j(i)}^i\|. && (7.12)
\end{aligned}$$

If $\tau^j(i) \geq 0$, we have

$$\begin{aligned}
\|x_t^i - x_{t-\tau^j(i)}^i\| = \|\sum_{q=t-\tau^j(i)}^{t-1} (x_{q+1}^i - x_q^i)\| &\leq \sum_{q=t-\tau^j(i)}^{t-1} \|\eta_q \nabla^i F_q(\tilde{x}_q(i))\| \\
&\leq \sum_{q=t-\tau^j(i)}^{t-1} \eta_q G. && (7.13)
\end{aligned}$$

If $\tau^j(i) < 0$, by similar technique, we can also get (7.13). Inserting (7.13) into (7.12), we have

$$\begin{aligned}
&\sum_{t=1}^{T}\sum_{j=1}^{m} < x_t^j - x_*^j, \nabla^j F_t(x_t) - \nabla^j F_t(\tilde{x}_t(j)) > \\
\leq &Gm \sum_{t=1}^{T}\sum_{j=1}^{m} L_j \|x_t^j - x_*^j\| \cdot \sum_{q=t-\tau}^{t-1} \eta_q && (7.14)\\
\leq &GmL_{\max} \sum_{t=1}^{T}\sum_{q=t-\tau}^{t-1} \eta_q \sum_{j=1}^{m} \|x_t^j - x_*^j\| && (7.15)\\
\leq &Gm^{\frac{3}{2}} L_{\max} \sum_{t=1}^{T}\sum_{q=t-\tau}^{t-1} \eta_q \|x_t - x_*\| && (7.16)\\
\leq &GDm^{\frac{3}{2}} L_{\max} \sum_{t=1}^{T}\sum_{q=t-\tau}^{t-1} \eta_q. && (7.17)
\end{aligned}$$

(7.10) is from triangle inequality. (7.11) comes from the Assumption 1's blockwise Lipschitz continuity. (7.16) comes from the fact

$$\frac{1}{m}\sum_{j=1}^{m} \|x_t^j - x_*^j\| \leq \frac{1}{\sqrt{m}} \sqrt{\sum_{j=1}^{m} \|x_t^j - x_*^j\|^2} = \frac{1}{\sqrt{m}} \|x_t - x_*\|. \tag{7.18}$$

For the last parts of (7.17), we have

$$\sum_{t=1}^{T}\sum_{q=t-\tau}^{t-1}\eta_q \leq \sum_{t=1}\tau\eta_1 t + \sum_{t=\tau+1}^{T}\sum_{q=t-\tau}^{t-1}\eta_q \leq \frac{\eta\tau(\tau+1)}{2} + \sum_{t=\tau+1}^{T}\frac{\tau\eta}{\sqrt{t-\tau}} \tag{7.19}$$

$$\leq \frac{\eta\tau(\tau+1)}{2} + 2\tau\eta\sqrt{T-\tau} \tag{7.20}$$

$$= \frac{\eta\tau}{2}(\tau+1+4\sqrt{T}), \tag{7.21}$$

where (7.20) is from the fact (7.3). Combining (7.17) and (7.21), we get

$$\sum_{t=1}^{T}\sum_{j=1}^{m} < x_t^j - x_*^j, \nabla^j F_t(x_t) - \nabla^j F_t(\tilde{x}_t(j)) >$$

$$\leq \frac{1}{2} G D m^{\frac{3}{2}} L_{\max}\eta\tau(\tau+1+4\sqrt{T}) \tag{7.22}$$

Combining (7.6), (7.7), (7.9) and (7.22), and dividing by T, we have

$$R \leq \frac{\eta m G^2}{\sqrt{T}} + \frac{D^2}{\eta\sqrt{T}} + \frac{1}{2} G D m^{\frac{3}{2}} L_{\max}\eta\tau\frac{1}{\sqrt{T}}(\frac{\tau+1}{\sqrt{T}}+4) = O(\frac{1}{\sqrt{T}}).$$

■

7.2 Supplementary Materials for Chapter 3

7.2.1 Proof of Theorem 2

To help theoretical analysis, we denote the objective functions in (3.6) and (3.7) as

$$g_m(x_m) = \lambda R_m(x_m) + \langle y^t, \mathcal{D}_m x_m\rangle + \frac{\rho}{2}\Big\|\sum_{\substack{k=1\\k\neq m}}^{M}\mathcal{D}_k x_k^t + \mathcal{D}_m x_m - z^t\Big\|^2,$$

$$h(z) = l(z) - \langle y^t, z\rangle + \frac{\rho}{2}\Big\|\sum_{m=1}^{M}\mathcal{D}_m x_m^{t+1} - z\Big\|^2, \tag{7.23}$$

correspondingly. We prove the following four lemmas to help prove the theorem.

Lemma 15 *Under Assumption 2, we have*

$$\nabla l(z^{t+1}) = y^{t+1},$$

and

$$\|y^{t+1} - y^t\|^2 \leq L^2\|z^{t+1} - z^t\|^2.$$

Proof. By the optimality in (3.7), we have

$$\nabla l(z^{t+1}) - y^t + \rho\big(z^{t+1} - \sum_{m=1}^{M}\mathcal{D}_m x_m^{t+1}\big) = 0.$$

Combined with (3.8), we can get

$$\nabla l(z^{t+1}) = y^{t+1}. \tag{7.24}$$

Combined with Assumption 2.1, we have

$$\|y^{t+1} - y^t\|^2 = \|\nabla l(z^{t+1}) - \nabla l(z^t)\|^2 \leq L^2 \|z^{t+1} - z^t\|^2. \tag{7.25}$$

□

Lemma 16 *We have*

$$\begin{aligned}&\big(\|\sum_{m=1}^{M} x_m^{t+1} - z\|^2 - \|\sum_{m=1}^{M} x_m^t - z\|^2\big) - \sum_{m=1}^{M}\big(\|\sum_{\substack{k=1\\k\neq m}}^{M} x_k^t + x_m^{t+1} - z\|^2 - \|\sum_{m=1}^{M} x_m^t - z\|^2\big)\\ \leq&\sum_{m=1}^{M}\|x_m^{t+1} - x_m^t\|^2.\end{aligned} \tag{7.26}$$

Proof.

$$\begin{aligned}\text{LHS} =&\big(\sum_{m=1}^{M}(x_m^{t+1} + x_m^t) - 2z\big)^\top\big(\sum_{m=1}^{M} x_m^{t+1} - \sum_{m=1}^{M} x_m^t\big) - \sum_{m=1}^{M}\big(\sum_{\substack{k=1\\k\neq m}}^{M} 2x_k^t + x_m^t + x_m^{t+1} - 2z\big)^\top(x_m^{t+1} - x_m^t)\\ =&-\sum_{m=1}^{M}\sum_{\substack{k=1\\k\neq m}}^{M}(x_k^{t+1} - x_k^t)^\top(x_m^{t+1} - x_m^t)\\ =&-\|\sum_{m=1}^{M}(x_m^{t+1} - x_m^t)\|^2 + \sum_{m=1}^{M}\|x_m^{t+1} - x_m^t\|^2\\ \leq&\sum_{m=1}^{M}\|x_m^{t+1} - x_m^t\|^2.\end{aligned}$$

□

Lemma 17 *Suppose Assumption 2 holds. We have*

$$\begin{aligned}&\mathcal{L}(\{x_m^{t+1}\}, z^{t+1}; y^{t+1}) - \mathcal{L}(\{x_m^t\}, z^t; y^t)\\ \leq&\sum_{m=1}^{M} -\left(\frac{\gamma_m(\rho)}{2} - \sigma_{max}(\mathcal{D}_m^\top \mathcal{D}_m)\right)\|x_m^{t+1} - x_m^t\|^2 - \left(\frac{\gamma(\rho)}{2} - \frac{L^2}{\rho}\right)\|z^{t+1} - z^t\|^2.\end{aligned}$$

Proof. The LFH can be decomposed into two parts as

$$\begin{aligned}&\mathcal{L}(\{x_m^{t+1}\}, z^{t+1}; y^{t+1}) - \mathcal{L}(\{x_m^t\}, z^t; y^t)\\ =&\big(\mathcal{L}(\{x_m^{t+1}\}, z^{t+1}; y^{t+1}) - \mathcal{L}(\{x_m^{t+1}\}, z^{t+1}; y^t)\big)\\ &+\big(\mathcal{L}(\{x_m^{t+1}\}, z^{t+1}; y^t) - \mathcal{L}(\{x_m^t\}, z^t; y^t)\big).\end{aligned} \tag{7.27}$$

For the first term, we have

$$\begin{aligned}
&\mathcal{L}(\{x_j^{t+1}\}, z^{t+1}; y^{t+1}) - \mathcal{L}(\{x_j^{t+1}\}, z^{t+1}; y^t) \\
=&\langle y^{t+1} - y^t, \sum_j D_j x_j^{t+1} - z^{t+1}\rangle \\
=&\frac{1}{\rho}\|y^{t+1} - y^t\|^2 \quad \text{(by (3.8))} \\
=&\frac{L^2}{\rho}\|z^{t+1} - z^t\|^2 \quad \text{(by Lemma 15).}
\end{aligned} \tag{7.28}$$

For the second term, we have

$$\begin{aligned}
&\mathcal{L}(\{x_m^{t+1}\}, z^{t+1}; y^t) - \mathcal{L}(\{x_m^t\}, z^t; y^t) \\
=&\mathcal{L}(\{x_m^{t+1}\}, z^{t+1}; y^t) - \mathcal{L}(\{x_m^t\}, z^{t+1}; y^t) + \mathcal{L}(\{x_m^t\}, z^{t+1}; y^t) - \mathcal{L}(\{x_m^t\}, z^t; y^t) \\
\le&\Bigg(\big(\lambda \sum_{m=1}^M R_m(x_m^{t+1}) + \langle y^t, \sum_{m=1}^M \mathcal{D}_m x_m^{t+1}\rangle + \frac{\rho}{2}\|\sum_{k=1}^M \mathcal{D}_k x_k^{t+1} - z^{t+1}\|^2\big) \\
&- \big(\lambda \sum_{m=1}^M R_m(x_m^t) + \langle y^t, \sum_{m=1}^M \mathcal{D}_m x_m^t\rangle + \frac{\rho}{2}\|\sum_{k=1}^M \mathcal{D}_k x_k^t - z^{t+1}\|^2\big)\Bigg) \\
&+ \Bigg(\big(l(z^{t+1}) - \langle y^t, z^{t+1}\rangle + \frac{\rho}{2}\|\sum_{m=1}^M \mathcal{D}_m x_m^{t+1} - z^{t+1}\|^2\big) - \big(l(z^t) - \langle y^t, z^t\rangle + \frac{\rho}{2}\|\sum_{m=1}^M \mathcal{D}_m x_m^{t+1} - z^t\|^2\big)\Bigg) \\
\le&\sum_{m=1}^M \Bigg(\big(\lambda R_m(x_m^{t+1}) + \langle y^t, \mathcal{D}_m x_m^{t+1}\rangle + \frac{\rho}{2}\|\sum_{\substack{k=1\\k\neq m}}^M \mathcal{D}_k x_k^t + \mathcal{D}_m x_m^{t+1} - z^{t+1}\|^2\big) \\
&- \big(\lambda R_m(x_m^t) + \langle y^t, \mathcal{D}_m x_m^t\rangle + \frac{\rho}{2}\|\sum_{k=1}^M \mathcal{D}_k x_k^t - z^{t+1}\|^2\big)\Bigg) + \sum_{m=1}^M \|\mathcal{D}_m(x_m^{t+1} - x_m^t)\|^2 \\
&+ \Bigg(\big(l(z^{t+1}) - \langle y^t, z^{t+1}\rangle + \frac{\rho}{2}\|\sum_{m=1}^M \mathcal{D}_m x_m^{t+1} - z^{t+1}\|^2\big) \\
&- \big(l(z^t) - \langle y^t, z^t\rangle + \frac{\rho}{2}\|\sum_{m=1}^M \mathcal{D}_m x_m^{t+1} - z^t\|^2\big)\Bigg) \quad \text{(by Lemma 16)} \\
=&\sum_{m=1}^M \big(g_m(x_m^{t+1}) - g_m(x_m^t)\big) + (h(z^{t+1}) - h(z^T))) + \sum_{m=1}^M \|\mathcal{D}_m(x_m^{t+1} - x_m^t)\|^2 \\
\le&\sum_{m=1}^M \big(\langle \nabla g_m(x_m^{t+1}), x_m^{t+1} - x_m^t\rangle - \frac{\gamma_m(\rho)}{2}\|x_m^{t+1} - x_m^t\|^2\big) + \langle \nabla h(z^{t+1}), z^{t+1} - z^t\rangle - \frac{\gamma(\rho)}{2}\|z^{t+1} - z^t\|^2 \\
&+ \sum_{m=1}^M \|\mathcal{D}_m(x_m^{t+1} - x_m^t)\|^2 \quad \text{(by strongly convexity from Assumption 2.2)}
\end{aligned}$$

$$\leq -\sum_{m=1}^{M} \frac{\gamma_m(\rho)}{2}\|x_m^{t+1} - x_m^t\|^2 - \frac{\gamma(\rho)}{2}\|z^{t+1} - z^t\|^2 + \sum_{m=1}^{M} \|\mathcal{D}_m(x_m^{t+1} - x_m^t)\|^2$$

(by optimality condition for subproblem in (3.6) and (3.7))

$$\leq \sum_{m=1}^{M} -\left(\frac{\gamma_m(\rho)}{2} - \sigma_{\max}(\mathcal{D}_m^\top \mathcal{D}_m)\right)\|x_m^{t+1} - x_m^t\|^2 - \frac{\gamma(\rho)}{2}\|z^{t+1} - z^t\|^2. \tag{7.29}$$

Note that we have abused the notation $\nabla g_m(x_m)$ and denote it as the subgradient when g is non-smooth but convex. Combining (7.27), (7.28) and (7.29), the lemma is proved. □

Lemma 18 *Suppose Assumption 2 holds. Then the following limit exists and is bounded from below:*

$$\lim_{t\to\infty} \mathcal{L}(\{x^{t+1}\}, z^{t+1}; y^{t+1}). \tag{7.30}$$

Proof.

$$\begin{aligned}
&\mathcal{L}(\{x^{t+1}\}, z^{t+1}; y^{t+1})\\
=&l(z^{t+1}) + \lambda\sum_{m=1}^{M} R_m(x_m^{t+1}) + \langle y^{t+1}, \sum_{m=1}^{M}\mathcal{D}_m x_m^{t+1} - z^{t+1}\rangle + \frac{\rho}{2}\|\mathcal{D}_m x_m^{t+1} - z^{t+1}\|^2\\
=&\lambda\sum_{m=1}^{M} R_m(x_m^{t+1}) + l(z^{t+1}) + \langle \nabla l(z^{t+1}), \sum_{m=1}^{M}\mathcal{D}_m x_m^{t+1} - z^{t+1}\rangle + \frac{\rho}{2}\|\mathcal{D}_m x_m^{t+1} - z^{t+1}\|^2 \quad \text{(by Lemma 15)}\\
\geq&\lambda\sum_{m=1}^{M} R_m(x_m^{t+1}) + l(\sum_{m=1}^{M}\mathcal{D}_m x_m^{t+1}) + \frac{\rho - L}{2}\|\mathcal{D}_m x_m^{t+1} - z^{t+1}\|^2.
\end{aligned} \tag{7.31}$$

Combined with Assumption 2.3, $\mathcal{L}(\{x^{t+1}\}, z^{t+1}; y^{t+1})$ is lower bounded. Furthermore, by Assumption 2.2 and Lemma 17, $\mathcal{L}(\{x^{t+1}\}, z^{t+1}; y^{t+1})$ is decreasing. These complete the proof. □

Now we are ready for the proof.

Part 1. By Assumption 2.2, Lemma 17 and Lemma 18, we have

$$\begin{aligned}
&\|x_m^{t+1} - x_m^t\| \to 0 \quad \forall m = 1, 2, \ldots, M,\\
&\|z^{t+1} - z^t\| \to 0.
\end{aligned}$$

Combined with Lemma 15, we have

$$\|y^{t+1} - y^t\|^2 \to 0.$$

Combined with (3.8), we complete the proof for Part 1.

Part 2. Due to the fact that $\|y^{t+1} - y^t\|^2 \to 0$, by taking limit in (3.8), we can get (3.12).

At each iteration $t+1$, by the optimality of the subproblem in (3.7), we have

$$\nabla l(z^{t+1}) - y^t + \rho(z^{t+1} - \sum_{m=1}^{M}\mathcal{D}_m x_m^{t+1}) = 0. \tag{7.32}$$

Combined with (3.12) and taking the limit, we get (3.11).

Similarly, by the optimality of the subproblem in (3.6), for $\forall m \in \mathcal{M}$ there exists $\eta_m^{t+1} \in \partial R_m(x_m^{t+1})$, such that

$$\big\langle x_m - x_m^{t+1}, \lambda \eta_m^{t+1} + \mathcal{D}_m^T y^t + \rho \mathcal{D}_m^T \big(\sum_{\substack{k=1 \\ k \le m}}^{M} \mathcal{D}_k x_k^{t+1} + \sum_{\substack{k=1 \\ k > j}}^{M} \mathcal{D}_k x_k^t - z^t \big) \big\rangle \ge 0 \quad \forall x_m \in X_m. \tag{7.33}$$

Since R_m is convex, we have

$$\lambda R_m(x_m) - \lambda R_m(x_m^{t+1}) + \big\langle x - x_m^{t+1}, \mathcal{D}_m^\top y^t + \rho \big(\sum_{\substack{k=1 \\ k \le m}}^{M} \mathcal{D}_k x_k^{t+1} + \sum_{\substack{k=1 \\ k > j}}^{M} \mathcal{D}_k x_k^t - z^t \big)^T \mathcal{D}_j \big\rangle \ge 0 \quad \forall x_m \in X_m. \tag{7.34}$$

Combined with (3.12) and the fact $\|x_m^{t+1} - x_m^t\| \to 0$, by taking the limit, we get

$$\lambda R_m(x_m) - \lambda R_m(x_m^*) + \big\langle x - x_m^*, \mathcal{D}_m^\top y^* \big\rangle \ge 0 \quad \forall x_m \in X_m, \forall m, \tag{7.35}$$

which is equivalent to

$$\lambda R_m(x) + \big\langle y^*, \mathcal{D}_m x \big\rangle - \lambda R_m(x_m^*) - \big\langle y^*, \mathcal{D}_m x_m^* \big\rangle \ge 0 \quad \forall x \in X_m, \forall m. \tag{7.36}$$

And we can get the result in (3.9).
When $m \notin \mathcal{M}$, we have

$$\big\langle x_m - x_m^{t+1}, \lambda \nabla R_m(x_m^{t+1}) + \mathcal{D}_m^\top y^t + \rho \big(\sum_{\substack{k=1 \\ k \le m}}^{M} \mathcal{D}_k x_k^{t+1} + \sum_{\substack{k=1 \\ k > j}}^{M} \mathcal{D}_k x_k^t - z^t \big)^T \mathcal{D}_m \big\rangle \ge 0 \quad \forall x_m \in X_m. \tag{7.37}$$

Taking the limit and we can get (3.10).

Part 3. We first show that there exists a limit point for each of the sequences $\{x_m^t\}$, $\{z^t\}$ and $\{y^t\}$. Since $X_m, \forall m$ is compact, $\{x_m^t\}$ must have a limit point. With Theorem 2.1, we can get that $\{z^t\}$ is also compact and has a limit point. Furthermore, with Lemma 15, we can get $\{y^t\}$ is also compact and has a limit point.

We prove Part 3 by contradiction. Since $\{x_m^t\}$, $\{z^t\}$ and $\{y^t\}$ lie in some compact set, there exists a subsequence $\{x_m^{t_k}\}$, $\{z^{t_k}\}$ and $\{y^{t_k}\}$, such that

$$(\{x_m^{t_k}\}, z^{t_k}; y^{t_k}) \to (\{\hat{x}_m\}, \hat{z}; \hat{y}), \tag{7.38}$$

where $(\{\hat{x}_m\}, \hat{z}; \hat{y})$ is some limit point and by part 2, we have $(\{\hat{x}_m\}, \hat{z}; \hat{y}) \in Z^*$. Suppose that $\{\{x_m^t\}, z^t; y^t\}$ does not converge to Z^*, since $(\{x_m^{t_k}\}, z^{t_k}; y^{t_k})$ is a subsequence of it, there exists some $\gamma > 0$, such that

$$\lim_{k \to \infty} \ \operatorname{dist}\big((\{x_m^{t_k}\}, z^{t_k}; y^{t_k}); Z^* \big) = \gamma > 0. \tag{7.39}$$

From (7.38), there exists some $J(\gamma) > 0$, such that

$$\|(\{x_m^{t_k}\}, z^{t_k}; y^{t_k}) - (\{\hat{x}_m\}, \hat{z}; \hat{y})\| \leq \frac{\gamma}{2}, \quad \forall k \geq J(\gamma). \tag{7.40}$$

Since $(\{\hat{x}_m\}, \hat{z}; \hat{y}) \in Z^*$, we have

$$\text{dist}\big((\{x_m^{t_k}\}, z^{t_k}; y^{t_k}); Z^*\big) \leq \text{dist}\big((\{x_m^{t_k}\}, z^{t_k}; y^{t_k}); (\{\hat{x}_m\}, \hat{z}; \hat{y})\big). \tag{7.41}$$

From the above two inequalities, we must have

$$\text{dist}\big((\{x_m^{t_k}\}, z^{t_k}; y^{t_k}); Z^*\big) \leq \frac{\gamma}{2}, \quad \forall k \geq J(\gamma), \tag{7.42}$$

which contradicts to (7.39), completing the proof. □

7.2.2 Proof of Theorem 3

We first show an upper bound for V^t.

1. Bound for $\tilde{\nabla}_{x_m}\mathcal{L}(\{x_m^t\}, z^t; y^t)$. When $m \in \mathcal{M}$, from the optimality condition in (3.6), we have

$$0 \in \lambda \partial_{x_m} R_j(x_m^{t+1}) + \mathcal{D}^\top y^t + \rho \mathcal{D}_j^\top \big(\sum_{\substack{k=1 \\ k \neq j}}^{M} \mathcal{D}_k x_k^t + \mathcal{D}_m x_m^{t+1} - z^t \big).$$

By some rearrangement, we have

$$\big(x_m^{t+1} - \mathcal{D}^\top y^t - \rho \mathcal{D}_m^\top \big(\sum_{\substack{k=1 \\ k \neq j}}^{M} \mathcal{D}_k x_k^t + \mathcal{D}_m x_m^{t+1} - z^t \big)\big) - x_m^{t+1} \in \lambda \partial_{x_m} R_m(x_m^{t+1}),$$

which is equivalent to

$$x_m^{t+1} = \text{prox}_{\lambda R_m} \big[x_m^{t+1} - \mathcal{D}^\top y^t - \rho \mathcal{D}_m^\top \big(\sum_{\substack{k=1 \\ k \neq j}}^{M} \mathcal{D}_k x_k^t + \mathcal{D}_m x_m^{t+1} - z^t \big) \big]. \tag{7.43}$$

Therefore,

$$\begin{aligned}
&\|x_m^t - \text{prox}_{\lambda R_m} \big[x_m^t - \nabla_{x_m} \big(\mathcal{L}(\{x_m^t\}, z^t; y^t) - \lambda \sum_{m=1}^{M} R_m(x_m^t) \big) \big] \| \\
=& \big\| x_m^t - x_m^{t+1} + x_m^{t+1} - \text{prox}_{\lambda R_m} \big[x_m^t - \mathcal{D}^\top y^t - \rho \mathcal{D}_m^\top \big(\sum_{k=1}^{M} \mathcal{D}_k x_k^t - z^t \big) \big] \big\| \\
\leq& \|x_m^t - x_m^{t+1}\| + \big\| \text{prox}_{\lambda R_m} \big[x_m^{t+1} - \mathcal{D}^\top y^t - \rho \mathcal{D}_m^T \big(\sum_{\substack{k=1 \\ k \neq m}}^{M} \mathcal{D}_k x_k^t + \mathcal{D}_m x_m^{t+1} - z^t \big) \big] \\
&- \text{prox}_{\lambda R_m} \big[x_m^t - \mathcal{D}^\top y^t - \rho \mathcal{D}_m^T \big(\sum_{k=1}^{M} \mathcal{D}_k x_k^t - z^t \big) \big] \big\| \\
&\leq 2\|x_m^t - x_m^{t+1}\| + \rho \|\mathcal{D}_m^\top \mathcal{D}_m (x_m^{t+1} - x_m^t)\|.
\end{aligned} \tag{7.44}$$

When $m \notin \mathcal{M}$, similarly, we have

$$\lambda \nabla_{x_m} R_m(x_m^{t+1}) + \mathcal{D}^\top y^t + \rho \mathcal{D}_m^\top \big(\sum_{\substack{k=1 \\ k \neq j}}^{M} \mathcal{D}_k x_k^t + \mathcal{D}_m x_m^{t+1} - z^t \big) = 0. \tag{7.45}$$

Therefore,

$$\begin{aligned}
&\|\nabla_{x_m} \mathcal{L}(\{x_m^t\}, z^t; y^t)\| \\
=&\|\lambda \nabla_{x_m} R_m(x_m^t) + \mathcal{D}^\top y^t + \rho \mathcal{D}_m^\top \big(\sum_{k=1}^{M} \mathcal{D}_k x_k^t - z^t \big)\| \\
=&\|\lambda \nabla_{x_m} R_m(x_m^t) + \mathcal{D}^\top y^t + \rho \mathcal{D}_m^T \big(\sum_{k=1}^{M} \mathcal{D}_k x_k^t - z^t \big) \\
&- \big(\lambda \nabla_{x_m} R_m(x_m^{t+1}) + \mathcal{D}^\top y^t + \rho \mathcal{D}_m^T \big(\sum_{\substack{k=1 \\ k \neq j}}^{M} \mathcal{D}_k x_k^t + \mathcal{D}_m x_m^{t+1} - z^t \big) \big)\| \\
\leq&\lambda \|\nabla_{x_m} R_m(x_m^t) - \nabla_{x_m} R_m(x_m^{t+1})\| + \rho \|\mathcal{D}_m^\top \mathcal{D}_m (x_m^{t+1} - x_m^t)\| \\
\leq&L_m \|x_m^{t+1} - x_m^t\| + \rho \|\mathcal{D}_m^\top \mathcal{D}_m (x_m^{t+1} - x_m^t)\|. \quad \text{(by Assumption 2.4)}
\end{aligned} \tag{7.46}$$

2. Bound for $\|\nabla_z \mathcal{L}(\{x_m^t\}, z^t; y^t)\|$. By optimality condition in (3.7), we have

$$\nabla l(z^{t+1}) - y^t + \rho \big(z^{t+1} - \sum_{m=1}^{M} \mathcal{D}_m x_m^{t+1} \big) = 0.$$

Therefore

$$\begin{aligned}
&\|\nabla_z \mathcal{L}(\{x_m^t\}, z^t; y^t)\| \\
=&\|l(z^t) - y^t + \rho \big(z^t - \sum_{m=1}^{M} \mathcal{D}_m x_m^t \big)\| \\
=&\|l(z^t) - y^t + \rho \big(z^t - \sum_{m=1}^{M} \mathcal{D}_m x_m^t \big) - \big(l(z^{t+1}) - y^t + \rho \big(z^{t+1} - \sum_{m=1}^{M} \mathcal{D}_m x_m^{t+1} \big) \big)\| \\
\leq&(L + \rho)\|z^{t+1} - z^t\| + \rho \sum_{m=1}^{M} \|\mathcal{D}_m (x_m^{t+1} - x_m^t)\|.
\end{aligned} \tag{7.47}$$

3. Bound for $\| \sum_{m=1}^{M} \mathcal{D}_m x_m^t - z^t \|$. According to Lemma 15, we have

$$\| \sum_{m=1}^{M} \mathcal{D}_m x_m^t - z^t \| = \frac{1}{\rho} \|y^{t+1} - y^t\| \leq \frac{L}{\rho} \|z^{t+1} - z^t\|. \tag{7.48}$$

Combining (7.44), (7.46), (7.47) and (7.48), we can conclude that there exists some $C_1 > 0$, such that

$$V^t \leq C_1 (\|z^{t+1} - z^t\|^2 + \sum_{m=1}^{M} \|x_m^{t+1} - x_m^t\|^2), \tag{7.49}$$

By Lemma 17, there exists some constant $C_2 = \min\{\sum_{m=1}^{M} \frac{\gamma_m(\rho)}{2}, \frac{\gamma(\rho)}{2} - \frac{L^2}{\rho}\}$, such that

$$\begin{aligned}&\mathcal{L}(\{x_m^t\}, z^t; y^t) - \mathcal{L}(\{x_m^{t+1}\}, z^{t+1}; y^{t+1})\\ \geq& C_2(\|z^{t+1} - z^t\|^2 + \sum_{m=1}^{M} \|x_m^{t+1} - x_m^t\|^2).\end{aligned} \tag{7.50}$$

By (7.49) and (7.50), we have

$$V^t \leq \frac{C_1}{C_2}\mathcal{L}(\{x_m^t\}, z^t; y^t) - \mathcal{L}(\{x_m^{t+1}\}, z^{t+1}; y^{t+1}). \tag{7.51}$$

Taking the sum over $t = 1, \ldots, T$, we have

$$\begin{aligned}\sum_{t=1}^{T} V^t \leq& \frac{C_1}{C_2}\mathcal{L}(\{x^1\}, z^1; y^1) - \mathcal{L}(\{x^{t+1}\}, z^{t+1}; y^{t+1})\\ \leq& \frac{C_1}{C_2}(\mathcal{L}(\{x^1\}, z^1; y^1) - \underline{f}).\end{aligned} \tag{7.52}$$

By the definition of $T(\epsilon)$, we have

$$T(\epsilon)\epsilon \leq \frac{C_1}{C_2}(\mathcal{L}(\{x^1\}, z^1; y^1) - \underline{f}). \tag{7.53}$$

By taking $C = \frac{C_1}{C_2}$, we complete the proof. □

7.2.3 Proof of Lemma 6

From the optimality condition of the x update procedure in (3.16), we can get

$$\begin{aligned}\mathcal{D}_m x_{m,\mathcal{D}_m}^{t+1} &= -\mathcal{D}_m(\rho\mathcal{D}_m^\top\mathcal{D}_m)^{-1}\left[\lambda R_m'(x_{m,\mathcal{D}_m}^{t+1}) + \mathcal{D}_m^\top y^t + \rho\mathcal{D}_m^\top(\sum_{\substack{k=1\\k\neq m}}^{M}\mathcal{D}_k\tilde{x}_k - z)\right],\\ \mathcal{D}_m' x_{m,\mathcal{D}_m'}^{t+1} &= -\mathcal{D}_m'(\rho\mathcal{D}_m'^\top\mathcal{D}_m')^{-1}\left[\lambda R_m'(x_{m,\mathcal{D}_m'}^{t+1}) + \mathcal{D}_m'^\top y^t + \rho\mathcal{D}_m'^\top(\sum_{\substack{k=1\\k\neq m}}^{M}\mathcal{D}_k\tilde{x}_k - z)\right].\end{aligned}$$

Therefore we have

$$
\begin{aligned}
&\mathcal{D}_m x^{t+1}_{m,\mathcal{D}_m} - \mathcal{D}'_m x^{t+1}_{m,\mathcal{D}'_m} \\
&= -\mathcal{D}_m(\rho\mathcal{D}_m^\top\mathcal{D}_m)^{-1}\left[\lambda R'_m(x^{t+1}_{m,\mathcal{D}_m}) + \mathcal{D}_m^\top y^t \mathcal{D}_m + \rho\mathcal{D}_m^\top(\sum_{\substack{k=1\\k\neq m}}^M \mathcal{D}_k\tilde{x}_k - z)\right] \\
&\quad + \mathcal{D}'_m(\rho\mathcal{D}'^\top_m\mathcal{D}'_m)^{-1}\left[\lambda R'_m(x^{t+1}_{m,\mathcal{D}'_m}) + \mathcal{D}'^\top_m y^t + \rho\mathcal{D}'^\top_m(\sum_{\substack{k=1\\k\neq m}}^M \mathcal{D}_k\tilde{x}_k - z)\right] \\
&= \mathcal{D}_m(\rho\mathcal{D}_m^\top\mathcal{D}_m)^{-1} \\
&\qquad \times \left[\lambda(R'_m(x^{t+1}_{m,\mathcal{D}'_m}) - R'_m(x^{t+1}_{m,\mathcal{D}_m})) + (\mathcal{D}'_m - \mathcal{D}_m)^\top y^t + \rho(\mathcal{D}'_m - \mathcal{D}_m)^\top(\sum_{\substack{k=1\\k\neq m}}^M \mathcal{D}_k\tilde{x}_k - z)\right] \\
&\quad + [\mathcal{D}'_m(\rho\mathcal{D}'^\top_m\mathcal{D}'_m)^{-1} - \mathcal{D}_m(\rho\mathcal{D}_m^\top\mathcal{D}_m)^{-1}] \\
&\qquad \times \left(\lambda R'_m(x^{t+1}_{m,\mathcal{D}'_m}) + \mathcal{D}'^\top_m y^t + \rho\mathcal{D}'^\top_m(\sum_{\substack{k=1\\k\neq m}}^M \mathcal{D}_k\tilde{x}_k - z)\right).
\end{aligned}
$$

Denote

$$
\begin{aligned}
\Phi_1 &= \mathcal{D}_m(\rho\mathcal{D}_m^\top\mathcal{D}_m)^{-1} \\
&\qquad \times \left[\lambda(R'_m(x^{t+1}_{m,\mathcal{D}'_m}) - R'_m(x^{t+1}_{m,\mathcal{D}_m})) + (\mathcal{D}'_m - \mathcal{D}_m)^\top y^t + \rho(\mathcal{D}'_m - \mathcal{D}_m)^\top(\sum_{\substack{k=1\\k\neq m}}^M \mathcal{D}_k\tilde{x}_k - z)\right], \\
\Phi_2 &= [\mathcal{D}'_m(\rho\mathcal{D}'^\top_m\mathcal{D}'_m)^{-1} - \mathcal{D}_m(\rho\mathcal{D}_m^\top\mathcal{D}_m)^{-1}] \\
&\qquad \times \left(\lambda R'_m(x^{t+1}_{m,\mathcal{D}'_m}) + \mathcal{D}'^\top_m y^t + \rho\mathcal{D}'^\top_m(\sum_{\substack{k=1\\k\neq m}}^M \mathcal{D}_k\tilde{x}_k - z)\right).
\end{aligned}
$$

As a result:

$$
\mathcal{D}_m x^{t+1}_{m,\mathcal{D}_m} - \mathcal{D}'_m x^{t+1}_{m,\mathcal{D}'_m} = \Phi_1 + \Phi_2. \tag{7.54}
$$

In the following, we will analyze the components in (7.54) term by term. The object is to prove $\max_{\substack{\mathcal{D}_m, D'_m\\ \|\mathcal{D}_m - D'_m\| \leq 1}} \|x^{t+1}_{m,\mathcal{D}_m} - x^{t+1}_{m,\mathcal{D}'_m}\|$ is bounded. To see this, notice that

$$
\begin{aligned}
&\max_{\substack{\mathcal{D}_m, D'_m\\ \|\mathcal{D}_m - D'_m\| \leq 1}} \|\mathcal{D}_m x^{t+1}_{m,\mathcal{D}_m} - \mathcal{D}'_m x^{t+1}_{m,\mathcal{D}'_m}\| \\
\leq &\max_{\substack{\mathcal{D}_m, D'_m\\ \|\mathcal{D}_m - D'_m\| \leq 1}} \|\Phi_1\| + \max_{\substack{\mathcal{D}_m, D'_m\\ \|\mathcal{D}_m - D'_m\| \leq 1}} \|\Phi_2\|.
\end{aligned}
$$

For $\max_{\substack{\mathcal{D}_m, D'_m \\ \|\mathcal{D}_m - D'_m\| \le 1}} \|\Phi_2\|$, from assumption 3.3, we have

$$\begin{aligned}
&\max_{\substack{\mathcal{D}_m, D'_m \\ \|\mathcal{D}_m - D'_m\| \le 1}} \|\Phi_2\| \\
&\le \left\| \frac{2}{d_m \rho} \left(\lambda R'_m(x^{t+1}_{m,\mathcal{D}'_m}) + \mathcal{D}'^{\top}_m y^t + \rho \mathcal{D}'^{\top}_m (\sum_{\substack{k=1 \\ k \ne m}}^{M} \mathcal{D}_k \tilde{x}_k - z) \right) \right\|.
\end{aligned}$$

By mean value theorem, we have

$$\begin{aligned}
&\left\| \frac{2}{d_m \rho} \left(\lambda \mathcal{D}'^{\top}_m R''_m(x_*) + \mathcal{D}'^{\top}_m y^t + \rho \mathcal{D}'^{\top}_m (\sum_{\substack{k=1 \\ k \ne m}}^{M} \mathcal{D}_k \tilde{x}_k - z) \right) \right\| \\
&\le \frac{2}{d_m \rho} \left[\lambda \|R''_m(\cdot)\| + \|y^t\| + \rho \|(\sum_{\substack{k=1 \\ k \ne m}}^{M} \mathcal{D}_k \tilde{x}_k - z)\| \right].
\end{aligned}$$

For $\max_{\substack{\mathcal{D}_m, D'_m \\ \|\mathcal{D}_m - D'_m\| \le 1}} \|\Phi_1\|$, we have

$$\begin{aligned}
&\max_{\substack{\mathcal{D}_m, D'_m \\ \|\mathcal{D}_m - D'_m\| \le 1}} \|\Phi_1\| \le \Bigg\| \mathcal{D}_m (\rho \mathcal{D}^{\top}_m \mathcal{D}_m)^{-1} \\
&\times \left[\lambda (R'_m(x^{t+1}_{m,\mathcal{D}'_m}) - R'_m(x^{t+1}_{m,\mathcal{D}_m})) + (\mathcal{D}'_m - \mathcal{D}_m)^{\top} y^t + \rho (\mathcal{D}'_m - \mathcal{D}_m)^{\top} (\sum_{\substack{k=1 \\ k \ne m}}^{M} \mathcal{D}_k \tilde{x}_k - z) \right] \Bigg\| \\
&\le \rho^{-1} \|(\mathcal{D}^{\top}_m \mathcal{D}_m)^{-1}\| \left[\lambda \|R''_m(\cdot)\| + \|y^t\| + \rho \|(\sum_{\substack{k=1 \\ k \ne m}}^{M} \mathcal{D}_k \tilde{x}_k - z)^{\top}\| \right] \\
&= \frac{1}{d_m \rho} \left[\lambda \|R''_m(\cdot)\| + \|y^t\| + \rho \|(\sum_{\substack{k=1 \\ k \ne m}}^{M} \mathcal{D}_k \tilde{x}_k - z)\| \right].
\end{aligned}$$

Thus by assumption 3.1-3.2

$$\begin{aligned}
&\max_{\substack{\mathcal{D}_m, D'_m \\ \|\mathcal{D}_m - D'_m\| \le 1}} \|\mathcal{D}_m x^{t+1}_{m,\mathcal{D}_m} - \mathcal{D}'_m x^{t+1}_{m,\mathcal{D}'_m}\| \\
&\le \frac{3}{d_m \rho}\left[\lambda c_1 + \|y^t\| + \rho\|(\sum_{\substack{k=1\\k\neq m}}^{M} \mathcal{D}_k \tilde{x}_k - z)^\top\|\right] \\
&\le \frac{3}{d_m \rho}\left[\lambda c_1 + \|y^t\| + \rho\|z\| + \rho\sum_{\substack{k=1\\k\neq m}}^{M} \|\tilde{x}_k\|\right] \\
&\le \frac{3}{d_m \rho}\left[\lambda c_1 + (1 + M\rho) b_1\right]
\end{aligned}$$

is bounded. □

7.2.4 Proof of Theorem 7

Proof: The privacy loss from $D_m \tilde{x}^{t+1}_m$ is calculated by:

$$\left|\ln \frac{P(\mathcal{D}_m \tilde{x}^{t+1}_m | \mathcal{D}_m)}{P(\mathcal{D}'_m \tilde{x}^{t+1}_m | \mathcal{D}'_m)}\right| = \left|\ln \frac{P(\mathcal{D}_m \tilde{x}^{t+1}_{m,\mathcal{D}_m} + \mathcal{D}_m \xi^{t+1}_m)}{P(\mathcal{D}'_m \tilde{x}^{t+1}_{m,\mathcal{D}'_m} + \mathcal{D}'_m \xi'^{,t+1}_m)}\right| = \left|\ln \frac{P(\mathcal{D}_m \xi^{t+1}_m)}{P(\mathcal{D}'_m \xi'^{,t+1}_m)}\right|.$$

Since $\mathcal{D}_m \xi^{t+1}_m$ and $\mathcal{D}'_m \xi'^{,t+1}_m$ are sampled from $\mathcal{N}(0, \sigma^2_{m,t+1})$, combine with lemma 6, we have

$$\begin{aligned}
&\left|\ln \frac{P(\mathcal{D}_m \xi^{t+1}_m)}{P(\mathcal{D}'_m \xi'^{,t+1}_m)}\right| \\
&= \left|\frac{2\xi^{t+1}_m \|\mathcal{D}_m x^{t+1}_{m,\mathcal{D}_m} - \mathcal{D}'_m x^{t+1}_{m,\mathcal{D}'_m}\| + \|\mathcal{D}_m x^{t+1}_{m,\mathcal{D}_m} - \mathcal{D}'_m x^{t+1}_{m,\mathcal{D}'_m}\|^2}{2\sigma^2_{m,t+1}}\right| \\
&\le \left|\frac{2\mathcal{D}_m \xi^{t+1}_m \mathbb{C} + \mathbb{C}^2}{2\frac{\mathbb{C}^2 \cdot 2\ln(1.25/\sigma)}{\varepsilon^2}}\right| \\
&= \left|\frac{(2\mathcal{D}_m \xi^{t+1}_m + \mathbb{C})\varepsilon^2}{4\mathbb{C}\ln(1.25/\sigma)}\right|.
\end{aligned}$$

In order to make $\left|\frac{(2\mathcal{D}_m \xi^{t+1}_m + \mathbb{C})\varepsilon^2}{4\mathbb{C}\ln(1.25/\sigma)}\right| \le \varepsilon$, we need to make sure

$$\left|\mathcal{D}_m \xi^{t+1}_m\right| \le \frac{2\mathbb{C}\ln(1.25/\sigma)}{\varepsilon} - \frac{\mathbb{C}}{2}.$$

In the following, we need to proof

$$P(\left|\mathcal{D}_m \xi^{t+1}_m\right| \ge \frac{2\mathbb{C}\ln(1.25/\sigma)}{\varepsilon} - \frac{\mathbb{C}}{2}) \le \delta \tag{7.55}$$

holds. However, we will proof a stronger result that lead to (7.55). Which is

$$P(\mathcal{D}_m\xi_m^{t+1} \geq \frac{2\mathbb{C}\ln(1.25/\sigma)}{\varepsilon} - \frac{\mathbb{C}}{2}) \leq \frac{\delta}{2}.$$

Since the tail bound of normal distribution $\mathcal{N}(0, \sigma_{m,t+1}^2)$ is:

$$P(\mathcal{D}_m\xi_m^{t+1} > r) \leq \frac{\sigma_{m,t+1}}{r\sqrt{2\pi}} e^{-\frac{r^2}{2\sigma_{m,t+1}^2}}.$$

Let $r = \frac{2\mathbb{C}\ln(1.25/\sigma)}{\varepsilon} - \frac{\mathbb{C}}{2}$, we then have

$$\begin{aligned}&P(\mathcal{D}_m\xi_m^{t+1} \geq \frac{2\mathbb{C}\ln(1.25/\sigma)}{\varepsilon} - \frac{\mathbb{C}}{2})\\ &\leq \frac{\mathbb{C}\sqrt{2\ln(1.25/\sigma)}}{r\sqrt{2\pi}\varepsilon} \exp\left[-\frac{[4\ln(1.25/\sigma) - \varepsilon]^2}{8\ln(1.25/\sigma)}\right].\end{aligned}$$

When δ is small and let $\varepsilon \leq 1$, we then have

$$\frac{\sqrt{2\ln(1.25/\sigma)}2}{(4\ln(1.25/\sigma) - \varepsilon)\sqrt{2\pi}} \leq \frac{\sqrt{2\ln(1.25/\sigma)}2}{(4\ln(1.25/\sigma) - 1)\sqrt{2\pi}} < \frac{1}{\sqrt{2\pi}}. \tag{7.56}$$

As a result, we can proof that

$$-\frac{[4\ln(1.25/\sigma) - \varepsilon]^2}{8\ln(1.25/\sigma)} < \ln(\sqrt{2\pi}\frac{\delta}{2})$$

by equation (7.56). Thus we have

$$P(\mathcal{D}_m\xi_m^{t+1} \geq \frac{2\mathbb{C}\ln(1.25/\sigma)}{\varepsilon} - \frac{\mathbb{C}}{2}) < \frac{1}{\sqrt{2\pi}} \exp(\ln(\sqrt{2\pi}\frac{\delta}{2}) = \frac{\delta}{2}.$$

Thus we proved (7.55) holds. Define

$$\begin{aligned}\mathbb{A}_1 &= \{\mathcal{D}_m\xi_m^{t+1} : |\mathcal{D}_m\xi_m^{t+1}| \leq \frac{1}{\sqrt{2\pi}} \exp(\ln(\sqrt{2\pi}\frac{\delta}{2}\},\\ \mathbb{A}_2 &= \{\mathcal{D}_m\xi_m^{t+1} : |\mathcal{D}_m\xi_m^{t+1}| > \frac{1}{\sqrt{2\pi}} \exp(\ln(\sqrt{2\pi}\frac{\delta}{2}\}.\end{aligned}$$

Thus we obtain the desired result:

$$\begin{aligned}&P(\mathcal{D}'_m\tilde{x}_m^{t+1}|\mathcal{D}_m)\\ &= P(\mathcal{D}_m x_{m,\mathcal{D}_m}^{t+1} + \mathcal{D}_m\xi_m^{t+1} : \mathcal{D}_m\xi_m^{t+1} \in \mathbb{A}_1)\\ &+P(\mathcal{D}_m x_{m,\mathcal{D}_m}^{t+1} + \mathcal{D}_m\xi_m^{t+1} : \mathcal{D}_m\xi_m^{t+1} \in \mathbb{A}_2)\\ &< e^{\varepsilon}P(\mathcal{D}_m x_{m,\mathcal{D}'_m}^{t+1} + \mathcal{D}_m\xi_m^{\prime,t+1}) + \delta = e^{\varepsilon}P(\mathcal{D}_m\tilde{x}_m^{t+1}|\mathcal{D}'_m) + \delta.\end{aligned}$$

□

7.3 Suplementary Materials for Chapter 5

7.3.1 Proof of Proposition 9

It is not hard to figure out that (CP) and (CMKP+) have the same region of feasible solutions. We will show that every feasible solution achieves the same object value in both problems to complete the proof.

For any solution $\{y_1, y_2, \ldots, y_n\}$, consider each particular $k \in \mathbb{Z}^+$ with $0 < k \le m$ and consider the corresponding group of coded blocks $\{i | y_i = k\}$, the contribution of the group to (5.1) is

$$\begin{aligned}\frac{1}{2}\mathbb{E}(L_k) =& \frac{1}{2}\mathbb{E}\big(\big(\sum_{j:y_j=k} D_j\big)^2\big)\\ =& \frac{1}{2}\sum_{i:y_i=k}\sum_{j:y_j=k;j\neq i} \mathbb{E}(D_i \cdot D_j) + \frac{1}{2}\sum_{i:y_i=k}\mathbb{E}(D_i^2)\\ =& \sum_{i<j:y_i=y_j=k} \boldsymbol{W}_{ij} + \frac{1}{2}\sum_{i:y_i=k}\boldsymbol{W}_{ii},\end{aligned}$$

which is exactly the contribution in (5.6). After summing up over k, we can get that the objective functions of (5.1) and (5.6) have the same value at the given feasible solution. □

7.3.2 Proof of Theorem 10

We will prove it by contradiction. Suppose $\{y_1, y_2, \ldots, y_n\}$ is a converged solution, in which, without loss of generality, there exists i_s and j_s such that $y_{i_s} = y_{j_s}$ and $G_{i_s} = G_{j_s}$.

We only consider the case that

$$\sum_{k:k\neq i_s;G_k\neq G_{i_s}} \boldsymbol{W}_{i_s k} \le \sum_{k:k\neq j_s;G_k\neq G_{j_s}} \boldsymbol{W}_{k j_s},$$

since the proof is similar for the other way around. For simplicity of presentation, we define $S(i_s)$ as the set of the indices of servers on which no block from the same group of i_m is placed. Considering the sum of the gains by moving i_s to all the servers in $S(i_s)$, we have

$$\begin{aligned}&\sum_{k\in S(i_s)} g_k(i_s)\\ &\ge (m-\alpha+1)\sum_{k:y_k=y_{i_s};k\neq i_s} \boldsymbol{W}'_{i_s k} - \sum_{k:G_k\neq G_{i_s}} \boldsymbol{W}'_{i_s k}\\ &\ge (m-\alpha+1)\boldsymbol{W}'_{i_s j_s} - \sum_{k:G_k\neq G_{i_s}} \boldsymbol{W}_{i_s k}\\ &\ge (m-\alpha+1)\epsilon > 0,\end{aligned}$$

which indicates that there exists a $k \in S(i_s)$, such that $g_k(i) > 0$. This contradicts the assumption that $\{y_1, y_2, \ldots, y_n\}$ is a solution from Algorithm 3 since the algorithm will only terminate when there is no positive $g_j(i)$. □

7.3.3 Proof of Theorem 11

The proof will depend on the following lemma and properties:

Lemma 19 *The solutions from the Local Block Migration Algorithm are solutions to (MKC) with an approximation ratio of* $1 - 1/m$.

Proof. Suppose that $\{y_1, y_2, \ldots, y_n\}$ is a converged solution. Due to local optimality, $\forall k' \neq k$, we have

$$\sum_{i:y_i=k} \sum_{j:y_j=k'} \boldsymbol{W}'_{ij} \geq \sum_{i:y_i=k} \sum_{j:y_j=k;j\neq i} \boldsymbol{W}'_{ij}. \tag{7.57}$$

Since there are $m-1$ possible k' values, by adding up (7.57) over all $k' \neq k$, we have

$$\sum_{i:y_i=k} \sum_{j:y_j\neq k} \boldsymbol{W}'_{ij} \geq (m-1) \sum_{i:y_i=k} \sum_{j:y_j=k;j\neq i} \boldsymbol{W}'_{ij}. \tag{7.58}$$

Dividing (7.58) by $m-1$ and adding $\sum_{i:y_i=k} \sum_{j:y_j\neq k} \boldsymbol{W}'_{ij}$ on both sides, we have

$$\frac{m}{m-1} \sum_{i:y_i=k} \sum_{j:y_j\neq k} \boldsymbol{W}'_{ij} \geq \sum_{i:y_i=k;j\neq i} \sum_{j} \boldsymbol{W}'_{ij}. \tag{7.59}$$

By summing up (7.59) over all k and dividing it by 2, we have

$$\frac{m}{m-1} \sum_{i<j} \boldsymbol{W}'_{ij}(1 - \delta(y_i - y_j)) \geq \sum_{i<j} \boldsymbol{W}'_{ij} \geq \mathrm{OPT}_C,$$

where OPT_C is the optimal value for the Max-k-Cut problem, completing the proof. ■

Lemma 20 *Given* $f_{ij}(\boldsymbol{W})$ *defined in (5.18), for any arbitrary positive* ϵ*, we have*

$$\sum_{i\neq j} \boldsymbol{W}'_{ij} \leq \epsilon + \frac{m}{m-\alpha+1} \sum_{i\neq j} \boldsymbol{W}_{ij}.$$

Proof. By (5.9) and (5.18), we have

$$\begin{aligned}
\sum_{i\neq j} \boldsymbol{W}'_{ij} \leq & \sum_{i\neq j} \boldsymbol{W}_{ij} + \sum_{i} \sum_{j:j\neq i;G_j=G_i} \boldsymbol{W}'ij \\
\leq & \sum_{i\neq j} \boldsymbol{W}_{ij} + \sum_{i} \sum_{j:j\neq i;G_j=G_i} (\epsilon + \frac{1}{m-\alpha+1} \sum_{k:k\neq i;G_k\neq G_i} \boldsymbol{W}_{ik}) \\
\leq & \sum_{i\neq j} \boldsymbol{W}_{ij} + \frac{\alpha-1}{m-\alpha+1} \sum_{i\neq j} \boldsymbol{W}_{ij} + (\alpha-1)n\epsilon \\
= & \epsilon' + \frac{m}{m-\alpha+1} \sum_{i\neq j} \boldsymbol{W}_{ij},
\end{aligned}$$

proving the lemma. ■

We have another property for the weight matrix $\boldsymbol{W}$:

$$\begin{aligned}
&\sum_{i\neq j}\boldsymbol{W}_{ij}/\sum_i \boldsymbol{W}_{ii} = \sum_{i\neq j}\mathbb{E}(D_iD_j)/\sum_i \mathbb{E}(D_i^2)\\
=&\Big(\mathbb{E}\big((\sum_i D_i)^2\big) - \sum_i \mathbb{E}(D_i^2)\Big)/\sum_i \mathbb{E}(D_i^2)\\
=&\frac{\mathbb{E}\big((\sum_i D_i)^2\big)}{\sum_i \mathbb{E}(D_i^2)} - 1. \qquad (7.60)
\end{aligned}$$

We are now ready to prove Theorem 11. Let OPT_P be the optimal value for (MKP). By Lemma 19 and (5.16), we have

$$\sum_{i<j}\boldsymbol{W}'_{ij} - \sum_{i<j}\boldsymbol{W}'_{ij}\delta(y_i-y_j) \geq \Big(1-\frac{1}{m}\Big)\big(\sum_{i<j}\boldsymbol{W}'_{ij} - \mathrm{OPT}_P\big). \qquad (7.61)$$

Reducing (7.61) and adding $\frac{1}{2}\sum_i \boldsymbol{W}_{ii}$, we have

$$\begin{aligned}
&\sum_{i<j}\boldsymbol{W}'_{ij}\delta(y_i-y_j) + \frac{1}{2}\sum_i \boldsymbol{W}_{ii}\\
\leq&\Big(1-\frac{1}{m}\Big)\Big(\mathrm{OPT}_P + \frac{1}{2}\sum_i \boldsymbol{W}_{ii}\Big) + \frac{1}{m}\Big(\sum_{i<j}\boldsymbol{W}'_{ij} + \frac{1}{2}\sum_i \boldsymbol{W}_{ii}\Big). \qquad (7.62)
\end{aligned}$$

By the property of $\boldsymbol{W}$ in (7.60) and Lemma 20, we have

$$\begin{aligned}
&\sum_{i<j}\boldsymbol{W}'_{ij} + \frac{1}{2}\sum_i \boldsymbol{W}_{ii}\\
\leq&\epsilon + \frac{m}{m-\alpha+1}\sum_{i<j}\boldsymbol{W}_{ij} + \frac{1}{2}\sum_i \boldsymbol{W}_{ii}\\
\leq&\epsilon + \frac{m}{m-\alpha+1}\Big(\frac{\mathbb{E}\big((\sum_i D_i)^2\big)}{\sum_i \mathbb{E}(D_i^2)} - 1\Big)\cdot\frac{1}{2}\sum_i \boldsymbol{W}_{ii} + \frac{1}{2}\sum_i \boldsymbol{W}_{ii}\\
\leq&\epsilon + \Big(\frac{m}{m-\alpha+1}\Big(\frac{\mathbb{E}\big((\sum_i D_i)^2\big)}{\sum_i \mathbb{E}(D_i^2)} - 1\Big) + 1\Big)\cdot\frac{1}{2}\sum_i \boldsymbol{W}_{ii}. \qquad (7.63)
\end{aligned}$$

By the feasibility of the solution, we also have

$$\sum_{i<j}\boldsymbol{W}'_{ij}\delta(y_i-y_j) = \sum_{i<j}\boldsymbol{W}_{ij}\delta(y_i-y_j). \qquad (7.64)$$

From (7.62), (7.63) and (7.64), we have

$$\begin{aligned}
&\sum_{i<j}\boldsymbol{W}_{ij}\delta(y_i-y_j) + \frac{1}{2}\sum_i \boldsymbol{W}_{ii}\\
\leq&\epsilon' + \Big(1 + \frac{1}{m-\alpha+1}\Big(\frac{\mathbb{E}\big((\sum_i D_i)^2\big)}{\sum_i \mathbb{E}(D_i^2)} - 1\Big)\Big)\cdot\Big(\frac{1}{2}\sum_i \boldsymbol{W}_{ii} + \mathrm{OPT}_P\Big). \qquad (7.65)
\end{aligned}$$

With the current setting of $\boldsymbol{W}'$, the optimal solution to (MKP) is also a feasible solution to the corresponding (CMKP+) problem, and also an optimal solution to (CMKP+). Combining it with the fact that ϵ is an arbitrary positive constant and the result in (7.65), we have completed the proof. □

7.3.4 Proof of Theorem 13

We show that in every iteration, if the solution from the previous iteration satisfies (5.7), in the current iteration, for every move that violates (5.7), there exists at least one other move that achieves a larger gain and yet does not violate (5.7).

Without loss of generality, consider all the gains achieved by moving the block i_s to other servers, respectively. There are two types of destination servers. One kind contains the blocks within the same group of i_s and the other kind does not. For all the destination servers not containing the blocks within the same group of i_s, denoted as $S(i_s)$, we have

$$\begin{aligned}
&\frac{1}{m-\alpha}\sum_{k\in S(i_s)} g_k(i_s)\\
&\geq \frac{1}{m-\alpha}\left((m-\alpha)\sum_{k:y_k=y_{i_s};k\neq i_s} \boldsymbol{W}'_{i_sk} - \sum_{k:G_k\neq G_{i_s}} \boldsymbol{W}'_{i_sk}\right)\\
&= \sum_{k:y_k=y_{i_s};k\neq i_s} \boldsymbol{W}'_{i_sk} - \frac{1}{m-\alpha}\sum_{k:G_k\neq G_{i_s}} \boldsymbol{W}'_{i_sk}\\
&= \sum_{k:y_k=y_{i_s};k\neq i_s} \boldsymbol{W}'_{i_sk} - \frac{1}{m-\alpha}\sum_{k:G_k\neq G_{i_s}} \boldsymbol{W}_{i_sk}.
\end{aligned}$$

By the definition of $\boldsymbol{W}'$ and (5.20), for any destination k_s containing a block j_s within the same group of i_s, we have

$$\begin{aligned}
g_{k_s}(i_s) &= \sum_{k:y_k=y_{i_s};k\neq i_s} \boldsymbol{W}'_{i_sk} - \sum_{k:y_k=k_s} \boldsymbol{W}'_{i_sk}\\
&\leq \sum_{k:y_k=y_{i_s};k\neq i_s} \boldsymbol{W}'_{i_sk} - \boldsymbol{W}'_{i_sj_s}\\
&\leq \sum_{k:y_k=y_i;k\neq i} \boldsymbol{W}'_{ik} - \epsilon - \frac{1}{m-\alpha}\sum_{k:k\neq i_s;G_k\neq G_{i_s}} \boldsymbol{W}_{i_sk}.
\end{aligned}$$

Therefore, we have

$$\frac{1}{m-\alpha}\sum_{k\in S(i_s)} g_k(i) > g_{k_s}(i),$$

which indicates that there exists at least one move with a better gain and yet does not violate the constraint, completing the proof. □

Bibliography

Abad, Cristina L, Nick Roberts, Yi Lu and Roy H Campbell (2012). A storage-centric analysis of mapreduce workloads: File popularity, temporal locality and arrival patterns. In: *Workload Characterization (IISWC), 2012 IEEE International Symposium on*. IEEE. pp. 100–109.

Abadi, Martin, Andy Chu, Ian Goodfellow, H Brendan McMahan, Ilya Mironov, Kunal Talwar and Li Zhang (2016*a*). Deep learning with differential privacy. In: *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*. ACM. pp. 308–318.

Abadi, Martín, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard et al. (2016*b*). Tensorflow: A System for Large-Scale Machine Learning. In: *Proc. USENIX Symposium on Operating System Design and Implementation (OSDI)*.

Agarwal, Sharad, John Dunagan, Navendu Jain, Stefan Saroiu, Alec Wolman and Harbinder Bhogan (2010). Volley: Automated data placement for geo-distributed cloud services.. In: *NSDI*. pp. 17–32.

ApS, MOSEK (2017). *The MOSEK Python optimizer API manual Version 7.1 (Revision 62)*.

Bellet, Aurélien, Yingyu Liang, Alireza Bagheri Garakani, Maria-Florina Balcan and Fei Sha (2015). A distributed frank-wolfe algorithm for communication-efficient sparse learning. In: *Proceedings of the 2015 SIAM International Conference on Data Mining*. SIAM. pp. 478–486.

Ben-Nun, Tal and Torsten Hoefler (2018). Demystifying parallel and distributed deep learning: An in-depth concurrency analysis. *arXiv preprint arXiv:1802.09941*.

Bonte, Charlotte and Frederik Vercauteren (2018). Privacy-preserving logistic regression training. Technical report. IACR Cryptology ePrint Archive 233.

Bonvin, Nicolas, Thanasis G Papaioannou and Karl Aberer (2010). A self-organized, fault-tolerant and scalable replication scheme for cloud storage. In: *Proceedings of the 1st ACM symposium on Cloud computing*. ACM. pp. 205–216.

Borthakur, Dhruba (2008). Hdfs architecture guide. *HADOOP APACHE PROJECT http://hadoop. apache. org/common/docs/current/hdfs design. pdf*.

Boyd, Stephen, Neal Parikh, Eric Chu, Borja Peleato, Jonathan Eckstein et al. (2011). Distributed optimization and statistical learning via the alternating direction method of multipliers. *Foundations and Trends® in Machine learning* **3**(1), 1–122.

Bradley, Joseph K, Aapo Kyrola, Danny Bickson and Carlos Guestrin (2011). Parallel coordinate descent for l1-regularized loss minimization. *arXiv preprint arXiv:1105.5379*.

Calder, Brad, Ju Wang, Aaron Ogus, Niranjan Nilakantan, Arild Skjolsvold, Sam McKelvie, Yikang Xu, Shashwat Srivastav, Jiesheng Wu, Huseyin Simitci et al. (2011). Windows azure storage: a highly available cloud storage service with strong consistency. In: *Proceedings of the Twenty-Third ACM Symposium on Operating Systems Principles*. ACM. pp. 143–157.

Chaudhuri, Kamalika and Claire Monteleoni (2009). Privacy-preserving logistic regression. In: *Advances in neural information processing systems*. pp. 289–296.

Chaudhuri, Kamalika, Claire Monteleoni and Anand D Sarwate (2011). Differentially private empirical risk minimization. *Journal of Machine Learning Research* **12**(Mar), 1069–1109.

Chen, Shengbo, Yin Sun, Longbo Huang, Prasun Sinha, Guanfeng Liang, Xin Liu, Ness B Shroff et al. (2014). When queueing meets coding: Optimal-latency data retrieving scheme in storage clouds. In: *IEEE INFOCOM 2014-IEEE Conference on Computer Communications*. IEEE. pp. 1042–1050.

Chen, Yanpei, Sara Alspaugh and Randy Katz (2012). Interactive analytical processing in big data systems: A cross-industry study of mapreduce workloads. *Proceedings of the VLDB Endowment* **5**(12), 1802–1813.

Chilimbi, Trishul M, Yutaka Suzue, Johnson Apacible and Karthik Kalyanaraman (2014). Project adam: Building an efficient and scalable deep learning training system.. In: *OSDI*. Vol. 14. pp. 571–582.

Dean, Jeffrey, Greg Corrado, Rajat Monga, Kai Chen, Matthieu Devin, Mark Mao, Andrew Senior, Paul Tucker, Ke Yang, Quoc V Le et al. (2012). Large scale distributed deep networks. In: *Advances in neural information processing systems*. pp. 1223–1231.

Dekel, Ofer, Ran Gilad-Bachrach, Ohad Shamir and Lin Xiao (2012). Optimal distributed online prediction using mini-batches. *Journal of Machine Learning Research* **13**(Jan), 165–202.

Dheeru, Dua and Efi Karra Taniskidou (2017). UCI machine learning repository.

Dua, Dheeru and Casey Graff (2017). UCI machine learning repository.

Dwork, Cynthia (2008). Differential privacy: A survey of results. In: *International Conference on Theory and Applications of Models of Computation*. Springer. pp. 1–19.

Dwork, Cynthia, Aaron Roth et al. (2014). The algorithmic foundations of differential privacy. *Foundations and Trends® in Theoretical Computer Science* **9**(3–4), 211–407.

Dwork, Cynthia and Kobbi Nissim (2004). Privacy-preserving datamining on vertically partitioned databases. In: *Annual International Cryptology Conference*. Springer. pp. 528–544.

Fiduccia, Charles M and Robert M Mattheyses (1982). A linear-time heuristic for improving network partitions. In: *Design Automation, 1982. 19th Conference on*. IEEE. pp. 175–181.

Ganjam, Aditya, Faisal Siddiqui, Jibin Zhan, Xi Liu, Ion Stoica, Junchen Jiang, Vyas Sekar and Hui Zhang (2015). C3: Internet-scale control plane for video quality optimization. In: *12th USENIX Symposium on Networked Systems Design and Implementation (NSDI 15)*. USENIX Association. Oakland, CA. pp. 131–144.

Ghemawat, Sanjay, Howard Gobioff and Shun-Tak Leung (2003). The google file system. In: *ACM SIGOPS operating systems review*. Vol. 37. ACM. pp. 29–43.

Gilad-Bachrach, Ran, Nathan Dowlin, Kim Laine, Kristin Lauter, Michael Naehrig and John Wernsing (2016). Cryptonets: Applying neural networks to encrypted data with high throughput and accuracy. In: *International Conference on Machine Learning*. pp. 201–210.

Gratton, Cristiano, Venkategowda Naveen KD, Reza Arablouei and Stefan Werner (2018). Distributed ridge regression with feature partitioning. In: *2018 52nd Asilomar Conference on Signals, Systems, and Computers*. IEEE. pp. 1423–1427.

Guyon, Isabelle, Steve Gunn, Asa Ben-Hur and Gideon Dror (2005). Result analysis of the nips 2003 feature selection challenge. In: *Advances in neural information processing systems*. pp. 545–552.

Hamm, Jihun, Yingjun Cao and Mikhail Belkin (2016). Learning privately from multiparty data. In: *International Conference on Machine Learning*. pp. 555–563.

Hitaj, Briland, Giuseppe Ateniese and Fernando Perez-Cruz (2017). Deep models under the gan: information leakage from collaborative deep learning. In: *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*. ACM. pp. 603–618.

Ho, Qirong, James Cipar, Henggang Cui, Seunghak Lee, Jin Kyu Kim, Phillip B Gibbons, Garth A Gibson, Greg Ganger and Eric P Xing (2013). More effective distributed ml via a stale synchronous parallel parameter server. In: *Advances in neural information processing systems*. pp. 1223–1231.

Hong, Mingyi, Zhi-Quan Luo and Meisam Razaviyayn (2016). Convergence analysis of alternating direction method of multipliers for a family of nonconvex problems. *SIAM Journal on Optimization* **26**(1), 337–364.

Hsieh, Kevin, Aaron Harlap, Nandita Vijaykumar, Dimitris Konomis, Gregory R Ganger, Phillip B Gibbons and Onur Mutlu (2017). Gaia: Geo-distributed machine learning approaching lan speeds.. In: *NSDI*. pp. 629–647.

Hu, Yaochen and Di Niu (2016). Reducing access latency in erasure coded cloud storage with local block migration. In: *IEEE INFOCOM 2016-The 35th Annual IEEE International Conference on Computer Communications*. IEEE. pp. 1–9.

Hu, Yaochen, Di Niu, Jianming Yang and Shengping Zhou (2019). Fdml: A collaborative machine learning framework for distributed features. In: *Proceedings of KDD '19*. ACM.

Hu, Yaochen, Yushi Wang, Bang Liu, Di Niu and Cheng Huang (2017). Latency reduction and load balancing in coded storage systems. In: *Proceedings of the 2017 Symposium on Cloud Computing*. ACM. pp. 365–377.

Huang, Cheng, Huseyin Simitci, Yikang Xu, Aaron Ogus, Brad Calder, Parikshit Gopalan, Jin Li, Sergey Yekhanin et al. (2012). Erasure coding in windows azure storage.. In: *Usenix annual technical conference*. Boston, MA. pp. 15–26.

Joshi, Gauri, Emina Soljanin and Gregory Wornell (2015). Efficient replication of queued tasks to reduce latency in cloud systems. In: *53rd Annual Allerton Conference on Communication, Control, and Computing*.

Kann, Viggo, Sanjeev Khanna, Jens Lagergren and Alessandro Panconesi (1997). On the hardness of approximating max k-cut and its dual. *Chicago Journal of Theoretical Computer Science* **2**, 1997.

Karp, Richard M (1972). *Reducibility among combinatorial problems*. Springer.

Kenthapadi, Krishnaram, Aleksandra Korolova, Ilya Mironov and Nina Mishra (2013). Privacy via the johnson-lindenstrauss transform. *Journal of Privacy and Confidentiality*.

Kernighan, Brian W and Shen Lin (1970). An efficient heuristic procedure for partitioning graphs. *Bell system technical journal* **49**(2), 291–307.

Khan, Osama, Randal C Burns, James S Plank, William Pierce and Cheng Huang (2012). Rethinking erasure codes for cloud file systems: minimizing i/o for recovery and degraded reads.. In: *FAST*. p. 20.

Kikuchi, Hiroaki, Chika Hamanaga, Hideo Yasunaga, Hiroki Matsui, Hideki Hashimoto and Chun-I Fan (2018). Privacy-preserving multiple linear regression of vertically partitioned real medical datasets. *Journal of Information Processing* **26**, 638–647.

Langford, John, Alexander J Smola and Martin Zinkevich (2009). Slow learners are fast. *Advances in Neural Information Processing Systems* **22**, 2331–2339.

Lee, Seunghak, Jin Kyu Kim, Xun Zheng, Qirong Ho, Garth A Gibson and Eric P Xing (2014). On model parallelization and scheduling strategies for distributed machine learning. In: *Advances in neural information processing systems*. pp. 2834–2842.

Li, Mu, David G Andersen, Alexander J Smola and Kai Yu (2014*a*). Communication efficient distributed machine learning with the parameter server. In: *Advances in Neural Information Processing Systems*. pp. 19–27.

Li, Mu, David G Andersen, Jun Woo Park, Alexander J Smola, Amr Ahmed, Vanja Josifovski, James Long, Eugene J Shekita and Bor-Yiing Su (2014*b*). Scaling distributed machine learning with the parameter server.. In: *OSDI*. Vol. 14. pp. 583–598.

Li, Mu, Ziqi Liu, Alexander J Smola and Yu-Xiang Wang (2016). Difacto: Distributed factorization machines. In: *Proceedings of the Ninth ACM International Conference on Web Search and Data Mining*. ACM. pp. 377–386.

Li, VOK, Q SHUAI and Y Zhu (2014*c*). Performance models of access latency in cloud storage systems. In: *Proc. Fourth Workshop on Architectures and Systems for Big Data*.

Lian, Xiangru, Yijun Huang, Yuncheng Li and Ji Liu (2015). Asynchronous parallel stochastic gradient for nonconvex optimization. In: *Advances in Neural Information Processing Systems*. pp. 2737–2745.

Liang, Guozheng and Ulas C Kozat (2014). Fast cloud: Pushing the envelope on delay performance of cloud storage with coding. *Networking, IEEE/ACM Transactions on* **22**(6), 2012–2025.

LIBSVM Data: Classification (Binary Class) (n.d.). `https://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/binary.html`. Accessed: 2019-05-23.

Lou, Jian and Yiu-ming Cheung (2018). Uplink communication efficient differentially private sparse optimization with feature-wise distributed data. In: *Thirty-Second AAAI Conference on Artificial Intelligence*.

McMahan, H Brendan, Eider Moore, Daniel Ramage, Seth Hampson et al. (2016). Communication-efficient learning of deep networks from decentralized data. *arXiv preprint arXiv:1602.05629*.

Mitzenmacher, Michael (2001). The power of two choices in randomized load balancing. *Parallel and Distributed Systems, IEEE Transactions on* **12**(10), 1094–1104.

Mitzenmacher, Michael David (1996). The Power of Two Choices in Randomized Load Balancing. PhD thesis. UNIVERSITY of CALIFORNIA at BERKELEY.

Mohassel, Payman and Yupeng Zhang (2017). Secureml: A system for scalable privacy-preserving machine learning. In: *2017 38th IEEE Symposium on Security and Privacy (SP)*. IEEE. pp. 19–38.

Morris, Paul (1993). The breakout method for escaping from local minima. In: *AAAI*. Vol. 93. pp. 40–45.

Ousterhout, Kay, Patrick Wendell, Matei Zaharia and Ion Stoica (2013). Sparrow: distributed, low latency scheduling. In: *Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles*. ACM. pp. 69–84.

Pathak, Manas, Shantanu Rane and Bhiksha Raj (2010). Multiparty differential privacy via aggregation of locally trained classifiers. In: *Advances in Neural Information Processing Systems*. pp. 1876–1884.

Rajkumar, Arun and Shivani Agarwal (2012). A differentially private stochastic gradient descent algorithm for multiparty classification. In: *Artificial Intelligence and Statistics*. pp. 933–941.

Rashmi, KV, Nihar B Shah, Dikang Gu, Hairong Kuang, Dhruba Borthakur and Kannan Ramchandran (2013). A solution to the network challenges of data recovery in erasure-coded distributed storage systems: A study on the facebook warehouse cluster. *Proc. USENIX HotStorage*.

Rashmi, KV, Nihar B Shah, Dikang Gu, Hairong Kuang, Dhruba Borthakur and Kannan Ramchandran (2014). A hitchhiker's guide to fast and efficient data reconstruction in erasure-coded data centers. In: *Proceedings of the 2014 ACM conference on SIGCOMM*. ACM. pp. 331–342.

Recht, Benjamin, Christopher Re, Stephen Wright and Feng Niu (2011). Hogwild: A lock-free approach to parallelizing stochastic gradient descent. In: *Advances in neural information processing systems*. pp. 693–701.

Ren, Kai, YongChul Kwon, Magdalena Balazinska and Bill Howe (2013). Hadoop's adolescence: an analysis of hadoop usage in scientific workloads. *Proceedings of the VLDB Endowment* **6**(10), 853–864.

Richa, Andrea W, M Mitzenmacher and R Sitaraman (2001). The power of two random choices: A survey of techniques and results. *Combinatorial Optimization* **9**, 255–304.

Rochman, Yuval, Hanoch Levy and Eli Brosh (2013). Resource placement and assignment in distributed network topologies. In: *INFOCOM, 2013 Proceedings IEEE*. IEEE. pp. 1914–1922.

Sarwate, Anand D and Kamalika Chaudhuri (2013). Signal processing and machine learning with differential privacy: Algorithms and challenges for continuous data. *IEEE signal processing magazine* **30**(5), 86–94.

Sathiamoorthy, Maheswaran, Megasthenis Asteris, Dimitris Papailiopoulos, Alexandros G Dimakis, Ramkumar Vadali, Scott Chen and Dhruba Borthakur (2013). Xoring elephants: Novel erasure codes for big data. In: *Proceedings of the VLDB Endowment*. Vol. 6. VLDB Endowment. pp. 325–336.

Scherrer, Chad, Ambuj Tewari, Mahantesh Halappanavar and David Haglin (2012). Feature clustering for accelerating parallel coordinate descent. In: *Advances in Neural Information Processing Systems*. pp. 28–36.

Schurman, Eric and Jake Brutlag (2009). The user and business impact of server delays, additional bytes, and http chunking in web search. In: *Velocity Web Performance and Operations Conference*.

Shah, Nihar B, Kangwook Lee and Kannan Ramchandran (2014). The mds queue: Analysing the latency performance of erasure codes. In: *2014 IEEE International Symposium on Information Theory*. IEEE. pp. 861–865.

Shah, Nihar B, Kangwook Lee and Kannan Ramchandran (2016). When do redundant requests reduce latency?. *IEEE Transactions on Communications* **64**(2), 715–722.

Shokri, Reza and Vitaly Shmatikov (2015). Privacy-preserving deep learning. In: *Proceedings of the 22nd ACM SIGSAC conference on computer and communications security*. ACM. pp. 1310–1321.

Sun, Yin, Zizhan Zheng, C Emre Koksal, Kyu-Han Kim and Ness B Shroff (2015). Provably delay efficient data retrieving in storage clouds. *arXiv preprint arXiv:1501.01661*.

Takabi, Hassan, Ehsan Hesamifard and Mehdi Ghasemi (2016). Privacy preserving multi-party machine learning with homomorphic encryption. In: *29th Annual Conference on Neural Information Processing Systems (NIPS)*.

Tamo, Itzhak and Alexander Barg (2014). A family of optimal locally recoverable codes. *Information Theory, IEEE Transactions on* **60**(8), 4661–4676.

Vaidya, Jaideep and Chris Clifton (2002). Privacy preserving association rule mining in vertically partitioned data. In: *Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM. pp. 639–644.

Wan, Li, Wee Keong Ng, Shuguo Han and Vincent Lee (2007). Privacy-preservation for gradient descent methods. In: *Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM. pp. 775–783.

Wang, Yu, Wotao Yin and Jinshan Zeng (2019). Global convergence of admm in nonconvex nonsmooth optimization. *Journal of Scientific Computing* **78**(1), 29–63.

Weatherspoon, Hakim and John D Kubiatowicz (2002). Erasure coding vs. replication: A quantitative comparison. In: *Peer-to-Peer Systems*. pp. 328–337. Springer.

Xia, Mingyuan, Mohit Saxena, Mario Blaum and David A Pease (2015). A tale of two erasure codes in hdfs. In: *To appear in Proceedings of 13th Usenix Conference on File and Storage Technologies*.

Xiang, Yu, Tian Lan, Vaneet Aggarwal and Yih Farn R Chen (2014). Joint latency and cost optimization for erasurecoded data center storage. *ACM SIGMETRICS Performance Evaluation Review* **42**(2), 3–14.

Xing, Eric P, Qirong Ho, Pengtao Xie and Dai Wei (2016). Strategies and principles of distributed machine learning on big data. *Engineering* **2**(2), 179–195.

Xu, Hong and Baochun Li (2013). Joint request mapping and response routing for geo-distributed cloud services. In: *INFOCOM, 2013 Proceedings IEEE*. IEEE. pp. 854–862.

Yan, Shiqin, Huaicheng Li, Mingzhe Hao, Michael Hao Tong, Swaminathan Sundararaman, Andrew A. Chien and Haryadi S. Gunawi (2017). Tiny-tail flash: Near-perfect elimination of garbage collection tail latencies in NAND ssds. In: *15th USENIX Conference on File and Storage Technologies (FAST 17)*. USENIX Association. Santa Clara, CA. pp. 15–28.

Ying, Bicheng, Kun Yuan and Ali H Sayed (2018). Supervised learning under distributed features. *IEEE Transactions on Signal Processing* **67**(4), 977–992.

Ying, Lei, R Srikant and Xiaohan Kang (2015). The power of slightly more than one sample in randomized load balancing. In: *2015 IEEE Conference on Computer Communications (INFOCOM)*. IEEE. pp. 1131–1139.

Yu, Boyang and Jianping Pan (2015). Location-aware associated data placement for geo-distributed data-intensive applications. In: *Proc. of IEEE Infocom 2015*.

Zhang, Chunlei, Muaz Ahmad and Yongqiang Wang (2019). Admm based privacy-preserving decentralized optimization. *IEEE Transactions on Information Forensics and Security* **14**(3), 565–580.

Zhang, Tao and Quanyan Zhu (2016). A dual perturbation approach for differential private admm-based distributed empirical risk minimization. In: *Proceedings of the 2016 ACM Workshop on Artificial Intelligence and Security*. ACM. pp. 129–137.

Zhang, Tao and Quanyan Zhu (2017). Dynamic differential privacy for admm-based distributed classification learning. *IEEE Transactions on Information Forensics and Security* **12**(1), 172–187.

Zhang, Xueru, Mohammad Mahdi Khalili and Mingyan Liu (2018). Improving the privacy and accuracy of admm-based distributed algorithms. In: *International Conference on Machine Learning*. pp. 5791–5800.

Zhou, Minqi, Rong Zhang, Wei Xie, Weining Qian and Aoying Zhou (2010). Security and privacy in cloud computing: A survey. In: *2010 Sixth International Conference on Semantics, Knowledge and Grids*. IEEE. pp. 105–112.

Zhou, Yi, Yaoliang Yu, Wei Dai, Yingbin Liang and Eric Xing (2016). On convergence of model parallel proximal gradient algorithm for stale synchronous parallel system. In: *Artificial Intelligence and Statistics*. pp. 713–722.

Zhu, Wenxing, Geng Lin and MM Ali (2013). Max-k-cut by the discrete dynamic convexized method. *INFORMS Journal on Computing* **25**(1), 27–40.

Zhu, Yujia, James Lin, Patrick PC Lee and Yan Xu (n.d.). Boosting degraded reads in heterogeneous erasure-coded storage systems.

Zinkevich, Martin, Markus Weimer, Lihong Li and Alex J Smola (2010). Parallelized stochastic gradient descent. In: *Advances in neural information processing systems*. pp. 2595–2603.