University of California
Santa Barbara

Robust and Efficient Algorithms for Federated Learning and Distributed Computing

A dissertation submitted in partial satisfaction
of the requirements for the degree

Doctor of Philosophy
in
Electrical and Computer Engineering

by

Amirhossein Reisizadehmobarakeh

September 2021

The Dissertation of Amirhossein Reisizadehmobarakeh is approved.

Professor Joao Hespanha

Professor Upamanyu Madhow

Professor Keneth Rose

Professor Salman Avestimehr, University of Southern California

Professor Ramtin Pedarsani, Committee Chair

August 2021

Robust and Efficient Algorithms for Federated Learning and Distributed Computing

To my Mom.

Acknowledgements

First and foremost, I would like to express my sincerest gratitude and appreciation to my advisor, Prof. Ramtin Pedarsani. He has been a wonderful source of help and support to me, in research and otherwise. I was first inspired by his unique approach to research. Ramtin always looks for exciting and interesting problems and is open to explore new areas. He taught me to put a great deal of effort into identifying practical challenges and formulating them in abstract senses. While we may not be able to solve *the whole* abstract problem, he believed that formulating a well-defined theoretical problem rooted in practical engineering and solving it *partially* may even offer more value than solving an extremely hard abstract one with little practical impacts. I was also inspired by his ability to identifying the student's interest and I deeply appreciate him letting me explore different research areas and establish fruitful collaborations. Ramtin is also extremely kind, patient and understanding. He genuinely cares about the well-being of his students and graciously offers his support. I simply could not wish for a better advisor and friend.

My PhD journey would not have been as joyful without the fruitful collaborations with an extraordinarily talented group of mentors and colleagues. My first collaborative PhD project was a joint work with Prof. Salman Avestimehr from USC. As a leading expert in the area, I was lucky to have the opportunity to learn the basics of fundamental research and presentation from him. Most importantly, Prof. Avestimehr is an amazing presenter from whom I learned how to keep the audience engaged in an impactful research presentation. I am also thankful to Prof. Aryan Mokhtari from UT Austin. I first found my interest in optimization theory and machine learning in our collaboration with Aryan while he was a postdoc at MIT. He always came up with brilliant ideas and was amazingly on top of all the tiny technical details of the proofs. He has been a wonderful source of help and support to me during the last few years. I also thank Prof. Farzan Farnia from CUHK. I was always amazed by his deep and versatile expertise on different areas. Farzan is always full of novel ideas and it has been very joyful to work with him. I thank Prof. Hamed Hassani from UPenn from whom I learned how to effectively present the story of an idea and write an engaging paper. I also thank Saurav Prakash from USC with whom we managed to publish quite a number of interesting works and were able to make the collection of our works and ideas to the Qualcomm Innovation Fellowship's finals. Lastly, I thank Prof. Ali Jadbabaie from MIT. I was lucky enough to have the opportunity to collaborate with such an experienced expert and learn a great deal about research. While we discussed different approaches to writing the story of a paper, Prof. Jadbabaie always came up with compelling and motivating big picture ideas while he was thoroughly on top of the technical details and challenges.

I would like to extend my gratitude to my thesis committee members: Prof. Joao Hespanha, Prof. Upamanyu Madhow, Prof. Keneth Rose and Prof. Salman Avestimehr. I also thank Prof. Christos Thrampoulidis for serving on my qualifying exam committee.

Studying abroad, thousands of miles far from home is not easy particularly without having a supportive group of friends. My wholehearted appreciation goes to my dear friend, Behnam. He has been extremely kind and supportive, always making sure to let me know that I can count on his help. I will always be grateful and indebted for that. I also thank my amazing LA friends, particularly Leili, Katy and Sepehr. Such joyful trips, parties and memories we have been together! I also thank my wonderful friends and neighbors at Anita's village. I will miss our tea times, hangouts and volleyballs.

Last but definitely not least, I owe this to my beloved family. Mom, I could not have gotten this far without your help, support and sacrifice. I have been extremely lucky to always feel your support by my side, though we have been away for the last few years. This thesis is dedicated to you.

Curriculum Vitæ
Amirhossein Reisizadehmobarakeh

Education

2021	Doctor of Philosophy, Electrical and Computer Engineering University of California, Santa Barbara, CA, USA.
2016	Master of Science, Electrical and Computer Engineering University of California, Los Angeles, CA, USA.
2014	Bachelor of Science, Electrical Engineering Sharif University of Technology, Tehran, Iran.

Publications

[1] **A. Reisizadeh**, A. Mokhtari, H. Hassani, A. Jadbabaie, R. Pedarsani, "FedPAQ: A Communication-Efficient Federated Learning Method with Periodic Averaging and Quantization," *International Conference on Artificial Intelligence and Statistics (**AISTATS**)*, pp. 2021–2031, 2020.

[2] **A. Reisizadeh***, F. Farnia*, R. Pedarsani, A. Jadbabaie, "Robust Federated Learning: The Case of Affine Distribution Shifts," *Advances in Neural Information Processing Systems (**NeurIPS**)*, vol. 33, 2020.

[3] **A. Reisizadeh**, H. Taheri, A. Mokhtari, H. Hassani, R. Pedarsani, "Robust and Communication-Efficient Collaborative Learning," *Advances in Neural Information Processing Systems (**NeurIPS**)*, pp. 8386–8397, 2019.

[4] **A. Reisizadeh**, A. Mokhtari, H. Hassani, R. Pedarsani, "An Exact Quantized Decentralized Gradient Descent Algorithm," *IEEE Transactions on Signal Processing*, 67 , issue 19 (2019), pp. 4934-4947.

[5] **A. Reisizadeh**, I. Tziotis, A. Mokhtari, H. Hassani, and R. Pedarsani, "Straggler-Resilient Federated Learning: Leveraging the Interplay Between Statistical Accuracy and System Heterogeneity," *Preprint*, 2020.

[6] **A. Reisizadeh**, S. Prakash, R. Pedarsani, and A. S. Avestimehr, "Coded Computation over Heterogeneous Clusters," *IEEE Transactions on Information Theory*, 65, no. 7 (2019), pp. 4227-4242.

[7] S. Prakash*, **A. Reisizadeh***, R. Pedarsani, and A. S. Avestimehr, "Coded Computing for Distributed Graph Analytics," *IEEE Transactions on Information Theory*, vol. 66, no. 10, pp. 6534–6554, 2020.

[8] **A. Reisizadeh**, S. Prakash, R. Pedarsani, and A. S. Avestimehr, "CodedReduce: A Fast and Robust Framework for Gradient Aggregation in Distributed Learning," accepted to appear in *IEEE/ACM Transactions on Networking*, *arXiv preprint arXiv:1902.01981*, 2019.

[9] **A. Reisizadeh**, A. Mokhtari, H. Hassani, R. Pedarsani, "Quantized Decentralized Consensus Optimization," *Proc. IEEE Conference on Decision and Control* (*CDC*), pp. 5838–5843, IEEE, 2018.

[10] **A. Reisizadeh***, S. Prakash*, R. Pedarsani, and A. S. Avestimehr, "Tree Gradient Coding," to appear in *Proc. IEEE International Symposium on Information Theory* (*ISIT*), pp. 2808–2812, IEEE, 2019.

[11] S. Prakash*, **A. Reisizadeh***, R. Pedarsani, and A. S. Avestimehr, "Coded Computing for Distributed Graph Analytics," *Proc. IEEE International Symposium on Information Theory* (*ISIT*), pp. 1221–1225, IEEE, 2018.

[12] **A. Reisizadeh**, S. Prakash, R. Pedarsani, and A. S. Avestimehr, "Coded Computation over Heterogeneous Clusters," *Proc. IEEE International Symposium on Information Theory* (*ISIT*), pp. 2408–2412, IEEE, 2017.

[13] **A. Reisizadeh**, R. Pedarsani, "Latency Analysis of Coded Computation Schemes over Wireless Networks," Allerton Conference on Communication, Control, and Computing, pp. 1256–1263, IEEE, 2017.

[14] S. Prakash*, **A. Reisizadeh***, R. Pedarsani, and A. S. Avestimehr, "Hierarchical Coded Gradient Aggregation for Learning at the Edge," *Proc. IEEE International Symposium on Information Theory* (*ISIT*), pp. 2616–2621, IEEE, 2020.

[15] **A. Reisizadeh**, P. Abdalla, R. Pedarsani, "Sub-linear Time Stochastic Threshold Group Testing via Sparse-Graph Codes," *IEEE Information Theory Workshop*, 2018.

[16] P. Abdalla, **A. Reisizadeh**, R. Pedarsani, "Multilevel Group Testing via Sparse-graph Codes," 51th Asilomar Conference on Signals, Systems and Computers, 2017.

[17] B. Amiri, **A. Reisizadeh**, H. Esfahanizadeh, J. Kliewer, and L. Dolecek, "Optimized Design of Finite-Length Spatially-Coupled Codes: An Absorbing Set-Based Analysis," *IEEE Transactions on Communications*, 64, issue 10 (2016), pp 4029-4043.

[18] **A. Reisizadeh**, C. Schoeny, C.-Y. Tsai, and L. Dolecek, "Approximate File Synchronization: Upper Bounds and Interactive Algorithms," *Information Theory Workshop* (*ITW*), pp. 320–324, IEEE, 2016.

[19] B. Amiri, **A. Reisizadeh**, J. Kliewer, and L. Dolecek, "Optimized Array-Based Spatially-Coupled LDPC Codes: An Absorbing Set Approach," in *Proc. IEEE International Symposium on Information Theory* (*ISIT*), pp. 51–55, IEEE, 2015.

*: Equal Contributions.

Abstract

Robust and Efficient Algorithms for Federated Learning and Distributed Computing

by

Amirhossein Reisizadehmobarakeh

Training a large-scale model over a massive data set is an extremely computation and storage intensive task, e.g. training ResNet with hundreds of millions of parameters over the data set ImageNet with millions of images. As a result, there has been significant interest in developing distributed learning strategies that speed up the training of learning models. Due to the growing computational power of the ecosystem of billions of mobile and computing devices, many future distributed learning systems operate based on storing data locally and pushing computation to the network edge. Unlike traditional centralized machine learning environments, however, machine learning at the edge is characterized by significant challenges including (1) scalability due to severe constraints on communication bandwidth and other resources including storage and energy, (2) robustness to stragglers, and edge failures due to slow edge nodes, (3) models generalizing to non-i.i.d. and heterogeneous data.

In this thesis, we focus on two important distributed learning frameworks: Federated Learning and Distributed Computing, with a shared goal in mind: how to provably address the critical challenges in such paradigms using novel techniques from distributed optimization, statistical learning theory, probability theory, and communication and coding theory to advance the state-of-the-art in efficiency, resiliency, and scalability.

In the first part of the thesis, we devise three methods to mitigate communication cost, straggler resiliency and robustness to heterogeneous data in federated learning paradigms. Our main ideas are to employ model compression, adaptive device participation and distributionally robust minimax optimization, respectively for such challenges. We characterize provable improvements for the proposed algorithms in terms of convergence speed, expected runtime, and generalization gaps.

Moving on to the second part, we consider important instances of distributed computing frameworks such as distributed gradient aggregation, matrix-vector multiplication and MapReduce-type computing tasks and propose several algorithms to mitigate the aforementioned bottlenecks in these settings. The key idea in our designs is to introduce redundant and coded computation in an elaborate fashion in order to benefit in communication cost and the total runtime. We also support our theoretical results in both parts by significant improvements in numerical experiments.

Contents

Chapter 1

Introduction

Human and industrial automation, powered by artificial intelligence (AI) and the growing ecosystem of billions of computing devices with sensors connected through the network edge, is shaping the future of our society. It also provides a platform to handle training of large-scale machine learning models over massive data sets. Unlike traditional centralized machine learning environments, however, machine learning at the edge is characterized by significant challenges including (1) scalability due to severe constraints on communication bandwidth and other resources including storage and energy, (2) robustness to stragglers, and edge failures due to slow edge nodes, (3) models generalizing to non-i.i.d. and heterogeneous data.

In this thesis, we focus on two important distributed learning frameworks: *Federated Learning* and *Distributed Computing*, with a shared goal in mind:

How to *provably* address the critical challenges in such paradigms using novel techniques from distributed optimization, statistical learning theory, probability theory, and communication and coding theory to advance the state-of-the-art in efficiency, resiliency, and scalability.

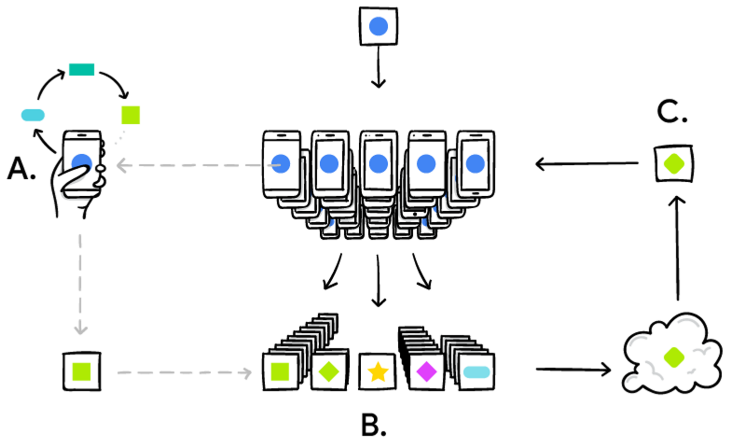


Figure 1.1: Illustration of a Federated Learning (FL) architecture (Figure adapted from [1]). A shared global is deployed to participating devices (A) where it's trained locally (A) and local models are aggregated (B) to provide an improved global model (C). This procedure continues till reaching accurate enough models.

This thesis consists of two main parts. In Part I, we study federated learning frameworks and elaborate on the aforementioned challenges therein. We propose our algorithms to mitigate such bottlenecks and study their theoretical and experimental characteristics. We shift our focus in Part II to distributed computing paradigms and lay out our designs to address their critical bottlenecks. In each chapter, we provide our main theoretical and numerical results and defer the details and proofs to corresponding appendices.

1.1 Algorithms for Federated Learning

In many large-scale machine learning applications, data is acquired and processed at the edge nodes of the network such as mobile devices, users' devices, and IoT sensors. Federated Learning (FL) is a novel paradigm that aims to train a learning model at the edge nodes as opposed to traditional distributed computing systems such as data centers. As privacy becomes a selling point, federated learning is poised to grow in popularity among both tech giants and industries where privacy protection is critical for personal data, like health care. To put it briefly, instead of bringing data all to one place for training, federated learning is done by bringing the model to the data. This allows a data owner to maintain the only copy of their information (See Figure 1.1). *Forbes* magazine enumerates FL as one of the emerging areas in the next generation of AI and highlights its future:

> "Federated learning may one day play a central role in the development of any AI application that involves sensitive data: from financial services to autonomous vehicles, from government use cases to consumer products of all kinds."
>
> *- Forbes, Oct. 12, 2020*

In addition to its recent and surging attention gained in the academia, federated learning is already being implemented in several high-tech companies such as Google. In particular, Google has implemented a next-word prediction application (GBoard) via federated learning which further emphasizes the practicality of such promising and novel paradigm [1]. This powerful framework is yet prone to multiple critical challenges [2,3].

(1) **Expensive Communication:** A federated network is potentially comprised of millions of devices, e.g. smart phones, IoT devices. Training on such massive network induces a dramatically heavy communication burden over the bandwidth-limited network.

(2) **System Heterogeneity:** Federated devices admit a wide range of storage, computational, and communication capabilities which exacerbate challenges such as straggler latency (slow devices) and fault tolerance.

(3) **Statistical Heterogeneity:** The data stored at federated devices are far from homogeneous statistical distributions and admit a variety of different distributions

which makes it crucial for the federated methods to properly generalize to heterogeneous data distributions.

In Chapters 2–4, we respectively target these three critical challenges in FL frameworks and in the following, we briefly describe our approaches to address them. In Chapter 2, we propose to employ model compression in order to reduce the communication load of the message passing over the network. Our `FedPAQ` design [4] incorporates local model training and partial device participation as well, enabling communication-efficient federated learning. In Chapter 3, we devise an adaptive node participation approach, namely `FLANP` to mitigate the stragglers by gradually growing the size of participating devices with respect to the statistical accuracy of the trained models [5]. Lastly, in Chapter 4, we tackle statistical heterogeneity challenge in FL and propose a minimax approach to make the trained model robust to worst-case distribution shifts [6]. For the case of affine distribution shifts, we propose a gradient descent-ascent optimization routine, namely `FedRobust`, and characterize its convergence and generalization implication.

1.1.1 Communication Efficiency in FL

In a federated learning architecture, a parameter server aims at finding a model that performs well with respect to the data points that are available at different nodes (users) of the network, while nodes exchange their local model with the server. However, privacy and communication concerns do not allow moving the raw data between the nodes and the server. Therefore, it is essential in federated learning to train globally while the data remains local. In an abstract form, we consider a federated learning framework where a network of n users are connected to a parameter server. Each user $i \in [n] \coloneqq \{1, \cdots, n\}$ stores a set of m data samples drawn from distribution $\mathcal{D}_i$ denoted by $S^i = \{z^i_j = (\mathbf{x}^i_j, y^i_j) : 1 \leq j \leq m\}$. Further, we define a loss function $\ell(\mathbf{w}; z) : \mathcal{W} \to \mathbb{R}$

where $\ell(\mathbf{w}; z_j^i)$ indicates how well the parameter model $\mathbf{w} \in \mathcal{W}$ performs with respect to the sample z_j^i. We also define the expected risk for each node i w.r.t its data distribution $\mathcal{D}_i$ as follows

$$f_i(\mathbf{w}) := \mathbb{E}_{z \sim \mathcal{D}_i}[\ell(\mathbf{w}, z)].$$

We focus on solving the following population risk minimization problem in order to find the optimal model $\mathbf{w}^*$ that minimizes the aggregate loss across the network:

$$\min_{\mathbf{w}} f(\mathbf{w}) := \frac{1}{n} \sum_{i=1}^{n} f_i(\mathbf{w}), \tag{1.1}$$

where $f_i(\mathbf{w})$ denotes the local loss function corresponding to data samples on node i.

The main goal in a federated network is to *collaboratively* learn a *good* model, in the sense that that the n users of the network with the help of the parameter server exchange iterative messages in order to reach a shared global model $\mathbf{w}$ *close* to the optimal model $\mathbf{w}^*$ which minimizes the overall loss function $f(\mathbf{w})$ defined in (1.1).

In Chapter 2, our main goal is to develop FL algorithms that induce low communication complexity while training the accurate enough model. Communication bottleneck is indeed one of the most critical challenges in scaling up FL methods [2, 3, 7]. To reduce the communication overhead in federated learning methods, we proposed `FedPAQ`, a communication-efficient federated learning algorithm [4]. Our proposed method consists of three main modules: (1) Quantized message-passing; (2) periodic averaging; and (3) partial device participation. These features address the communications and scalability challenges in federated learning. Next, we explain the main building blocks of our method in more detail.

Quantized message-passing: Due to the communication bottleneck, it is critical to reduce the size of the uploaded messages from the federated devices. Our proposal is to employ quantization operators on the transmitted massages. Depending on the accuracy of the quantizer, the network communication overhead is reduced by exchanging the quantized updates. More precisely, each node compresses its model update using a stochastic quantizer $Q(\cdot) : \mathcal{W} \to \mathcal{W}$ before uploading to the parameter server. We assume the quantizer $Q(\cdot)$ to be unbiased and variance bounded.

Periodic averaging: To incorporate all the available data samples on the nodes, any training method should synchronize the intermediate models obtained at local devices. One approach is to let the participating nodes synchronize their models through the server in each iteration which results in communication contention over the network. Instead, we let the devices conduct a number of local updates and synchronize through the server periodically, imposing less communication rounds. For instance, each node updates its own local model for τ consecutive SGD iterations after which local models are aggregated at the parameter server and the new global model is deployed back to the devices for next round of local updates. This implies that the number of communication rounds is slashed by τ hence improving the bandwidth efficiency.

Partial node participation: In a typical federated network, only a few number of devices are able to simultaneously upload their messages to the parameter server due to limited bandwidth. Moreover, in practical federated environments, not all the devices contribute in each round of the training. Our proposed `FedPAQ` method captures the restrictions mentioned above and we assume that among the total of n devices, only r nodes ($r < n$) are able to participate in each round.

We rigorously provide theoretical convergence guarantees for `FedPAQ` method ans show that after T iterations, the suboptimality of the final model $\mathbf{w}_T$ is of order $\mathbb{E}[\|\mathbf{w}_T - \mathbf{w}^*\|^2] \leq \mathcal{O}(\tau^2/T)$ for strongly convex loss functions. We also characterize the convergence rate for nonconvex losses (e.g. neural network) and prove that after T iterations of `FedPAQ`, there exists an iteration t with the average model of devices $\overline{\mathbf{w}}_t$ such that

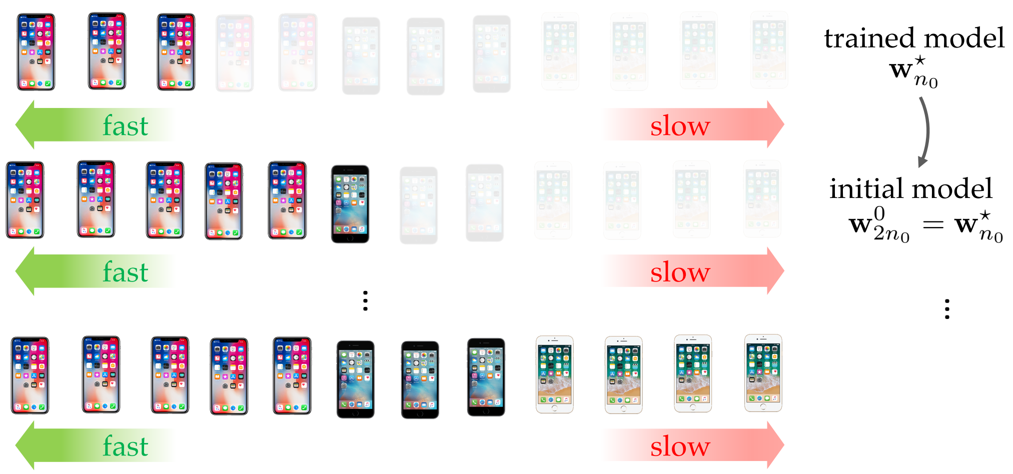


Figure 1.2: An overview of FLANP with $N = 12$ nodes. Training begins with $n_0 = 3$ participating nodes, doubling in each stage. The trained model in each stage is used as the initial model for the next stage with double participants.

$\mathbb{E}[\|\nabla f(\overline{\mathbf{w}}_t)\|^2] \leq \mathcal{O}(\frac{1}{\sqrt{T}} + \frac{\tau}{T})$. We also highlight the communication-computation trade-off introduced by FedPAQ via numerical experiments.

1.1.2 Stragglers and System Heterogeneity in FL

As mentioned earlier, a practical federated learning framework consists of thousands of devices with a wide range of computation, communication and storage characteristic which we refer to as system heterogeneity. A primal system challenge arisen from such device heterogeneity is that there are slower users or *stragglers* in the network which cause unexpected delays in the training time [2,3]. To address system heterogeneity and straggler resiliency challenge in federated learning paradigms, we propose an *adaptive node participation* approach described as follows.

In Chapter 3, we propose FLANP, a straggler-resilient FL algorithm that incorporates statistical characteristics of the clients' data to adaptively select the clients in order to speed up the learning procedure [5]. The key idea of this scheme is to start the model training procedure with only a few clients which are the fastest among all the nodes.

These participating clients continue to train their shared models while interacting with the parameter server. Note that since the server waits only for the participating nodes, it takes a short time for the participating (and fast) clients to promptly train a shared model. This model is, however, not accurate as it is trained over only a fraction of samples. We next double the number of participating clients and include the next fastest subset of nonparticipating nodes in the training. Note that the model trained from the previous stage can be a warm-start initialization for the current stage (Figure 1.2).

The proposed approach *provably* reduces the overall runtime required to achieve the statistical accuracy of data of all nodes. We are able to show that this adaptive method can provide up to $\mathcal{O}(\ln(Ns))$ speedup compared to standard benchmarks which employ all the N nodes during the training. Here, N and s respectively denote the total number of available nodes and the number of data samples per node. Experimentally as well, `FLANP` demonstrates speedup in wall-clock time compared to standard FL benchmarks.

1.1.3 Statistical Heterogeneity in FL

A critical and yet less-addressed challenge in scaling up the federated learning methods is their capability to learn from heterogeneous data. Most of the existing federated methods are designed to handle homogeneous data; the case that the data distributions among the devices are statistically homogeneous (i.e. i.i.d. data). Such frameworks are shown to fail in generalizing to heterogeneous data distributions [8]. This challenge is critical in making the federated learning methods practical as well, given that in practical scenarios the data generated or stored at user devices are highly heterogeneous, such as the photos taken by mobile devices.

In Chapter 4, we target the data heterogeneity challenge in federated learning frameworks and propose a new federated learning scheme called `FLRA`, a **F**ederated **L**earning

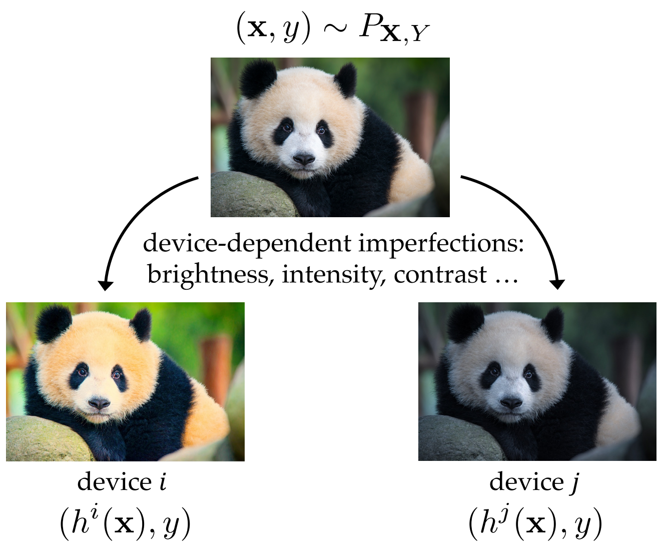

$(\mathbf{x}, y) \sim P_{\mathbf{X},Y}$
device-dependent imperfections:
brightness, intensity, contrast ...
device i
$(h^i(\mathbf{x}), y)$
device j
$(h^j(\mathbf{x}), y)$

Figure 1.3: An overview of `FLRA`. Samples across the devices are transformations of i.i.d. samples.

framework with **R**obustness to **A**ffine distribution shifts. `FLRA` has a small communication overhead and a low computation complexity. The key insight in `FLRA` is model the heterogeneity of training data in a device-dependent manner, according to which the samples stored on the ith device $\mathbf{x}^i$ are shifted from a ground distribution by an affine transformation $\mathbf{x}^i \to h^i(\mathbf{x}^i) = \Lambda^i \mathbf{x}^i + \delta^i$. To further illustrate this point, consider a federated image classification task where each mobile device maintains a collection of images. The images taken by a camera are similarly distorted depending on the intensity, contrast, blurring, brightness and other characteristics of the camera [9, 10], while these features vary across cameras (Figure 1.3). In addition to camera imperfections, such unseen distributional shifts also originate from changes in the physical environment, e.g. weather conditions [11].

We propose a novel minimax formulation that makes the learned model parameters robust to worst-case affine transformations. To efficiently solve this minimax problem, we proposed a robust federated learning routine named as `FedRobust` that enables the federated devices to individually learn their respective transformations Λ^i, δ^i while learning the global model and ensuring communication and computation efficiency. In addition, our method provably ensures that when a new device with unseen data joins the federated network, which is the case for federated mobile networks, the learned model properly generalizes to the new device. Our numerical results also demonstrate significant improvements both in accuracy (up to 54%) and computation time (by $4\times$) compared to well-known federated method `FedAvg` and adversarial projected gradient descent.

1.2 Algorithms for Distributed Computing

General distributed computing frameworks, such as MapReduce [12] and Spark [13], along with the availability of large-scale commodity servers, such as Amazon EC2, have made it possible to carry out large-scale data analytics at the production level. These "virtualized data centers" enjoy an abundance of storage space and computing power, and are cheaper to rent by the hour than maintaining dedicated data centers round the year. However, these systems suffer from various forms of "system noise" which reduce their efficiency: system failures, limited communication bandwidth, straggler nodes, etc.

A key distinction of this paradigm to federated learning is that the data is now offloaded from a central server to the edge nodes for computation purpose. Therefore, there exist coding opportunities in data allocation and communication strategy in order to tackle the communication bottleneck at the master, as well as to provide straggler resiliency. This is in contrast with federated learning where the local data cannot be encoded or repeated in other edge nodes due to privacy constraints.

We focus on the problem where a user off-loads a machine learning task (e.g. learning a face recognition model) via an access point of the edge cloud (See Figure 1.4). The underlying edge framework then facilitates the computational task by utilizing the edge data and carrying out the task over a large collection of edge devices. As illustrated in

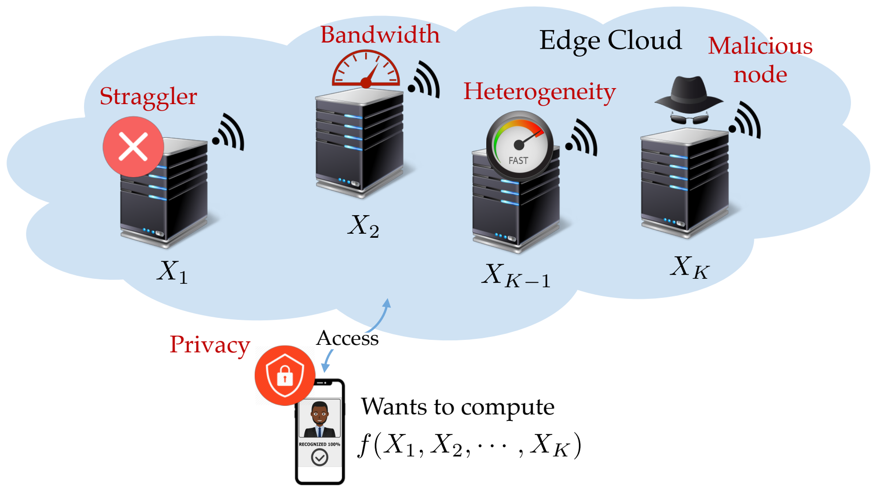


Figure 1.4: Illustration of a Distributed Computing (DC) framework. Here the mobile device intends to implement a machine learning algorithm by leveraging the data set available at the computing nodes. Three challenges need to be tackled – (1) Bandwidth, (2) Stragglers, (3) Security and Privacy.

Figure 1.4, there are three major concerns with distributed computing:

(1) **Communication bandwidth**, which is severely constrained due to low throughput and shared communication channels,

(2) **Straggling nodes**, arising out of node failures and re-transmission due to communication link failures, and

(3) **Security and Privacy**, due to malicious nodes, non-centralized computation and data sensitivities.

The current state-of-the-art approaches to mitigate the impact of system noise in cloud computing environments involve creation of some form of "computation redundancy". For example, *replicating* the straggling task on another available node is a common approach to deal with stragglers [14, 15]. However, there have been recent results demonstrating that *coding* can play a transformational role for creating and exploiting computation redundancy to effectively alleviate the impact of system noise.

In Chapters 5–7, we target the communication bandwidth and straggler nodes challenges in distributed computing frameworks and propose our *coded* distributed computing designs to mitigate such bottlenecks, enabling scalable DC methods. In Chapter 5, we propose a coding framework for speeding up distributed matrix multiplication in heterogeneous clusters with straggling servers. In Chapter 6, we simultaneously consider the two communication and straggler challenges in distributed gradient aggregation and present our joint computation–communication design approach to mitigate them [16]. Lastly in Chapter 7, we focus on large-scale graph processing paradigms and develop a coding scheme that systematically injects structured redundancy in the computation phase to enable coded multicasting opportunities during message exchange between servers, substantially reducing the communication load [17].

1.2.1 System Heterogeneity in Distributed Computing

Matrix multiplication is a fundamental component of many popular machine learning algorithms such as logistic regression, reinforcement learning and gradient descent-based algorithms. Implementations that speed up matrix multiplication would naturally speed up the execution of a wide variety of popular algorithms. In Chapter 5, we focus on general *heterogeneous* distributed computing clusters consisting of a variety of computing machines with different computational capabilities. We propose a coding framework for speeding up distributed matrix-vector multiplication in heterogeneous clusters with straggling servers, named **H**eterogeneous **C**oded **M**atrix **M**ultiplication (`HCMM`).

We consider the problem of matrix-vector multiplication, in which given a large matrix $\mathbf{A} \in \mathbb{R}^{r \times m}$ with large r, we want to compute the output $\mathbf{y} = \mathbf{A}\mathbf{x}$ for an input vector $\mathbf{x} \in \mathbb{R}^m$. Due to limited computing power, the computation cannot be carried out at a single server and a distributed implementation is required, i.e. horizontally split the large

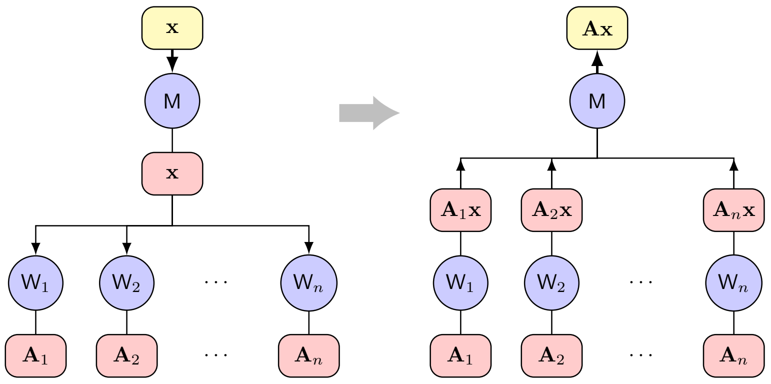


Figure 1.5: Master-worker setup of the computing clusters: The master node receives the input vector $\mathbf{x}$ and broadcasts it to all the worker nodes. Upon receiving the input, worker node i starts computing the inner products of the input vector with the locally assigned rows, i.e., $\mathbf{y}_i = \mathbf{A}_i\mathbf{x}$, and unicasts the output vector $\mathbf{y}_i$ to the master node upon completing the computation. The results are aggregated at the master node until r inner products are received and the desired output $\mathbf{Ax}$ is recovered.

matrix $\mathbf{A} = [\mathbf{A}_1; \cdots ; \mathbf{A}_n]$ to smaller matrices $\mathbf{A}_i$ and assign computing $\mathbf{A}_i\mathbf{x}$ to server i. Figure 1.5 illustrates an uncoded implementation of distributed computing, in which results from all the worker nodes are required to recover the final result.

We propose to design *coded* computation tasks for worker nodes and *decode* the computation results at the master node. For matrix-vector multiplication tasks in particular, local data blocks $\mathbf{A}_i \in \mathbb{R}^{\ell_i \times m}$ are matrices consisting of *coded* combinations of the rows in $\mathbf{A}$, for non-negative integers ℓ_i. To assign the computation tasks to each worker, we use random linear combinations of the r rows of the matrix $\mathbf{A}$, such that the master node can recover the result $\mathbf{Ax}$ from any r inner products received from the worker nodes with probability 1. We look for optimal load allocation $\boldsymbol{\ell}^* = (\ell_1^*, \cdots , \ell_n^*)$ that minimizes the expected waiting time at the master node in order to receive enough worker computations and recover the final result $\mathbf{y} = \mathbf{Ax}$. While finding $\boldsymbol{\ell}^*$ is computationally intractable, we present an alternative formulation and show that the solution to the alternative formulation – which we shall name `HCMM` – is tractable and provably asymptotically optimal. Moreover, we consider a shifted-exponential computation time model for workers' computation time and demonstrate that `HCMM` slashes the expected waiting time in uncoded benchmarks by $\Theta(\log(n))$.

1.2.2 Straggler and Communication Bottlenecks in Distributed Gradient Aggregation

In Chapter 6, we consider a collaborative learning setting where a machine learning model is trained over edge nodes of a network collaboratively. In particular, we consider a machine learning task that involves fitting a model over a training data set by minimizing a loss function. For a given labeled data set $\mathcal{D} = \{\mathbf{x}_j \in \mathbb{R}^{p+1} : j = 1, \cdots, d\}$, the goal is to solve the following optimization problem:

$$\min_{\mathbf{w}} \sum_{\mathbf{x} \in \mathcal{D}} \ell(\mathbf{w}; \mathbf{x}) + \lambda R(\mathbf{w}), \tag{1.2}$$

where $\ell(\cdot)$ and $R(\cdot)$ respectively denote the loss and regularization functions, and the optimization problem is parameterized by λ. One of the most popular ways of solving (1.2) in distributed learning is to use the Gradient Descent algorithm that is based on finding the gradient vector $\mathbf{g} = \sum_{\mathbf{x} \in \mathcal{D}} \nabla \ell(\mathbf{w}^{(t)}; \mathbf{x})$ over the data set $\mathcal{D}$ an for each iteration t. At scale, due to limited storage and computation capabilities of the computing nodes, gradient aggregation has to be carried out over distributed nodes. However, this parallelization introduces two major bottlenecks: stragglers and communication bottleneck.

To address the straggler bottleneck, Gradient Coding [18] was recently proposed in a master-worker topology with one master node and N distributed worker nodes. To explain our preliminary results, we first overview Gradient Coding with the following illustrative example. Consider a distributed learning problem with a master node and $N = 3$ worker nodes. To make gradient aggregation robust to one straggler, the proposed

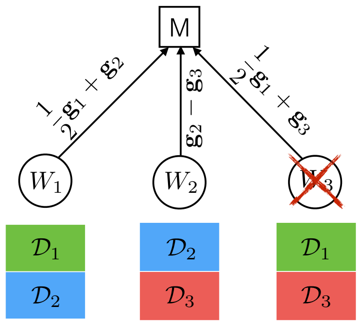
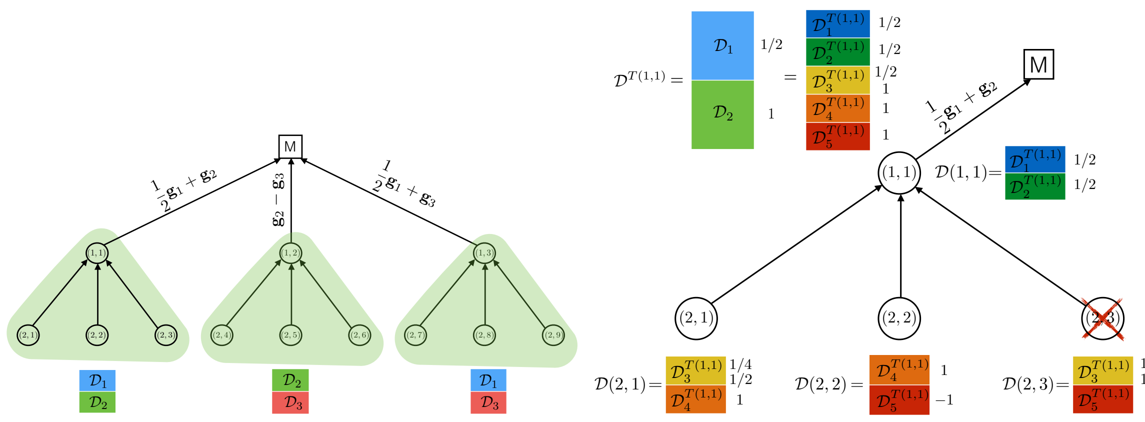

M
$\frac{1}{2}\mathbf{g}_1 + \mathbf{g}_2$
$\mathbf{g}_2 - \mathbf{g}_3$
$\frac{1}{2}\mathbf{g}_1 + \mathbf{g}_3$
W_1
W_2
W_3
$\mathcal{D}_1$
$\mathcal{D}_2$
$\mathcal{D}_2$
$\mathcal{D}_3$
$\mathcal{D}_1$
$\mathcal{D}_3$

Figure 1.6: Illustration of data allocation and communication strategy in Gradient Coding for $N = 3$ workers.

Gradient Coding algorithm partitions the data set $\mathcal{D}$ uniformly to $\{\mathcal{D}_1, \mathcal{D}_2, \mathcal{D}_3\}$ with corresponding gradient vectors $\mathbf{g}_1$, $\mathbf{g}_2$ and $\mathbf{g}_3$, and assigns two partitions to each worker specified by the encoding coefficients as depicted in Figure 1.6. One can then design an encoding scheme such that the total gradient can be recovered from the computation results of *any* two workers. For example, if node 3 fails, the master node can aggregate (decode) the gradient vector by forming the following: $\mathbf{g} = 2(\frac{1}{2}\mathbf{g}_1 + \mathbf{g}_2) - (\mathbf{g}_2 - \mathbf{g}_3)$.

While Gradient Coding is robust to stragglers and failures, in large-scale distributed systems it suffers from a significant bandwidth bottleneck at the master, as multiple workers concurrently send their computation results to the master, which yields communication load of $\mathcal{O}(N)$. To alleviate this communication bottleneck, we propose to develop a novel framework that achieves bandwidth efficiency and straggler toleration simultaneously. Our key insight is to design a joint data allocation, communication strategy, and encoding/decoding scheme that is robust to stragglers, alongside being communication-efficient

As the communication topology design, we propose a tree structure for distributed gradient aggregation. The advantage of having a tree communication topology is quite straightforward to see: Suppose that the computing nodes form a symmetric tree such that each parent has n children and there are a total of $N = n+n^2+\cdots+n^L$ worker nodes,



(a) Tree topology. (b) Data allocation and coding in `CR`.

Figure 1.7: Illustration of `CodedReduce` design for $N = 12$ worker nodes. (a) We design the communication topology as a tree with $L = 2$ layers and $n = 3$ children noes per parent. (b) Each node computes local gradients associated with the allocated data batch and uploads a linear combination (with specified weight) to its parent node.

where the tree has L layers with the master node as the main root. Then, each parent node has only n partial gradients to collect from its children, reducing the communication load from $\mathcal{O}(N)$ to $\mathcal{O}(n) = \mathcal{O}(N^{1/L})$ as communication across different nodes can be parallelized. However, this communication-efficiency does not come for free since one wants to still design a straggler-resilient coding scheme for gradient aggregation. We next explain our achievable coding scheme over the tree structure, namely as `CodedReduce` (`CR`) through a simple example. Consider the tree topology for $N = 12$ workers ($n = 3$) in Figure 1.7(a). The goal is to develop a data allocation and communication design such that the gradient aggregation at master has a resiliency of 1 straggler per parent. Similar to Gradient Coding, we partition $\mathcal{D}$ into $\mathcal{D}_1$, $\mathcal{D}_2$, $\mathcal{D}_3$; however, this time we assign them to the three sub-trees, as illustrated in Figure 1.7(a).

Without loss of generality, we focus on sub-tree $T(1,1)$ with route $(1,1)$. We note that the master needs to receive $\frac{1}{2}\mathbf{g}_1 + \mathbf{g}_2$ from $(1,1)$ (left sub-tree in Figure 1.7(a). The task allocation in this sub-tree is as follows: $\mathcal{D}^{T(1,1)} = \{\mathcal{D}_1, \mathcal{D}_2\}$ is further partitioned into five batches and assigned to the nodes as shown in Figure 1.7(b). The coding scheme is then cleverly designed such that from the computation results of *any* two children, node $(1,1)$ together with its own local gradient computation can recover $\frac{1}{2}\mathbf{g}_1 + \mathbf{g}_2$.

In a nutshell, `CodedReduce` parallelizes the communications over a tree topology leading to efficient bandwidth utilization, and carefully designs a redundant data set allocation and coding strategy at the nodes to make the proposed gradient aggregation scheme robust to stragglers. In particular, we quantify the communication parallelization gain and resiliency of the proposed `CR` scheme, and prove its optimality when the communication topology is a regular tree.

1.2.3 Bandwidth Bottleneck in Distributed Graph Analytics

Graphs are widely used to identify and incorporate the relationship patterns and anomalies inherent in real-life datasets, prompting the development of various large-scale distributed graph processing frameworks, such as Pregel [19], PowerGraph [20] and GraphLab [21]. The underlying theme in these systems is the *think like a vertex* approach [22] where the computation at each vertex requires only the data available in the neighborhood of the vertex (Figure 7.1).

In these distributed graph processing systems, for carrying out the graph computation for a given vertex at a particular server, the intermediate values corresponding to the neighboring vertices whose files are not available at the server have to be communicated from other servers. These distributed graph processing systems, therefore, require many messages to be exchanged among servers during job execution resulting in communication bottleneck [23] which accounts for more than 50% of the overall execution time in representative cases [24].

In Chapter 7, we target the communication bottleneck in distributed graph processing and develop a new framework that leverages *computation redundancy* by computing

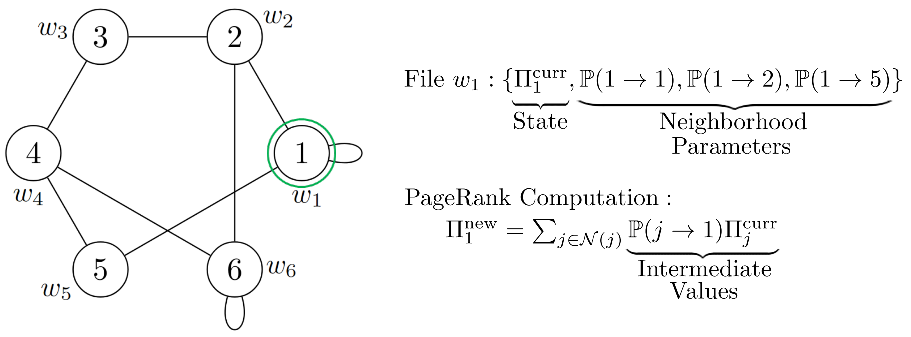

w_3
3
2
w_2
4
1
w_4
w_1
5
6
w_5
w_6
File w_1 : $\{\underbrace{\Pi_1^{\text{curr}}}_{\text{State}}, \underbrace{\mathbb{P}(1 \to 1), \mathbb{P}(1 \to 2), \mathbb{P}(1 \to 5)}_{\text{Neighborhood Parameters}}\}$
PageRank Computation :
$\Pi_1^{\text{new}} = \sum_{j \in \mathcal{N}(j)} \underbrace{\mathbb{P}(j \to 1)\Pi_j^{\text{curr}}}_{\text{Intermediate Values}}$

Figure 1.8: Illustrating the *think like a vertex* paradigm prevalent in common parallel graph computing frameworks. The computation associated with a vertex only depends on its neighbors. In this example, we consider the PageRank computation over a graph with six vertices. Using vertex 1 for representation, we illustrate the file and PageRank update at each vertex. File w_1 contains the state (current PageRank Π_1^{curr}) and the neighborhood parameters (probabilities of transitioning to neighbors $\{\mathbb{P}(1 \to 1), \mathbb{P}(1 \to 2), \mathbb{P}(1 \to 5)\}$). The PageRank update associated with vertex 1 is a function of only the neighborhood files (specifically, of the PageRanks of neighboring vertices and the transition probabilities from neighbors to vertex 1).

the intermediate values at multiple servers via *redundant subgraph allocation* [17]. The redundancy in computation of intermediate values at multiple servers allows coded multicasting opportunities during exchange of messages between servers, thus reducing the communication load. Our proposed framework comprises of a mathematical model for MapReduce decomposition [12] of the graph computation task. The Map computation for a vertex corresponds to computing the intermediate values for the vertices in its neighborhood, while the Reduce computation for a vertex corresponds to combining the intermediate values from the neighboring vertices to obtain the final result of graph computation. Referring to the example in Figure 7.1, the Map and Reduce computations associated with vertex 1 are as follows:

$$\text{Map: } \Pi_1^{\text{curr}} \to \{\mathbb{P}(1 \to 1)\Pi_1^{\text{curr}}, \mathbb{P}(1 \to 2)\Pi_1^{\text{curr}}, \mathbb{P}(1 \to 5)\Pi_1^{\text{curr}}\},$$

$$\text{Reduce: } \Pi_1^{\text{new}} = v_{1,1} + v_{1,2} + v_{1,5},$$

where $v_{1,i} = \mathbb{P}(i \to 1)\Pi_i^{\text{curr}}$ is the intermediate value obtained from the Map computation of vertex $i \in \mathcal{N}(1)$.

In distributed graph based MapReduce, each server is allocated a subgraph for Map computations and Reduce tasks for a subset of graph vertices, and the overall execution takes place in three phases – Map, Shuffle, and Reduce. Our framework proposes to trade redundant computations in the Map phase with communication load during the Shuffle phase. The key idea is to leverage the graph structure and create coded messages during the Shuffle phase that simultaneously satisfy the data demand of multiple computing servers in the Reduce phase.

For two popular random graph models, Erdös-Rényi model and power law model, we prove that our proposed coded scheme asymptotically achieves an inverse-linear trade-off between computation load in the Map phase and average normalized communication load in the Shuffle phase. Furthermore, for the Erdös-Rényi model, we develop an information-theoretic converse for the average communication load given a computation load. We also specialize our coded scheme and extend the achievability results to two additional random graph models, random bi-partite model and stochastic block model.

Part I

Algorithms for Federated Learning

Chapter 2

Communication-Efficient Federated Learning

Federated learning is a distributed framework according to which a model is trained over a set of devices, while keeping data localized. This framework faces several systems-oriented challenges which include (i) *communication bottleneck* since a large number of devices upload their local updates to a parameter server, and (ii) *scalability* as the federated network consists of millions of devices. Due to these systems challenges as well as issues related to statistical heterogeneity of data and privacy concerns, designing a provably efficient federated learning method is of significant importance yet it remains challenging. In this chapter, we present `FedPAQ`, a communication-efficient **Fed**erated Learning method with **P**eriodic **A**veraging and **Q**uantization. `FedPAQ` relies on three key features: (1) periodic averaging where models are updated locally at devices and only periodically averaged at the server; (2) partial device participation where only a fraction of devices participate in each round of the training; and (3) quantized message-passing where the edge nodes quantize their updates before uploading to the parameter server. These features address the communications and scalability challenges in federated learning. We also show that `FedPAQ` achieves near-optimal theoretical guarantees for strongly convex and non-convex loss functions and empirically demonstrate the communication-computation tradeoff provided by our method.

2.1 Introduction

In many large-scale machine learning applications, data is acquired and processed at the edge nodes of the network such as mobile devices, users' devices, and IoT sensors. *Federated Learning* is a novel paradigm that aims to train a statistical model at the "edge" nodes as opposed to the traditional distributed computing systems such as data centers [7,25]. The main objective of federated learning is to fit a model to data generated from network devices without continuous transfer of the massive amount of collected data from edge of the network to back-end servers for processing.

Federated learning has been deployed by major technology companies with the goal of providing privacy-preserving services using users' data [26]. Examples of such applications are learning from wearable devices [27], learning sentiment [28], and location-based services [29]. While federated learning is a promising paradigm for such applications, there are several challenges that remain to be resolved. In this chapter, we focus on two significant challenges of federated learning, and propose a novel federated learning algorithm that addresses the following two challenges:

(1) **Communication bottleneck.** Bandwidth is a major bottleneck in federated learning as a large number of devices attempt to communicate their local updates to a central parameter server. Thus, for a communication-efficient federated learning algorithm, it is crucial that such updates are sent in a compressed manner and infrequently.

(2) **Scale.** A federated network typically consists of thousands to millions of devices that may be active, slow, or completely inactive during the training procedure. Thus, a

proposed federated learning algorithm should be able to operate efficiently with partial device participation or random sampling of devices.

The goal of this chapter is to develop a provably efficient federated learning algorithm that addresses the above-mentioned systems challenges. More precisely, we consider the task of training a model in a federated learning setup where we aim to find an *accurate* model over a collection of n distributed nodes. In this setting, each node contains m independent and identically distributed samples from an unknown probability distribution and a parameter server helps coordination between the nodes. We focus on solving the population risk minimization problem for a federated architecture while addressing the challenges mentioned above. In particular, we consider both strongly convex and non-convex settings and provide guarantees on the performance of our proposed algorithm.

Contributions. In this chapter, we propose `FedPAQ`, a communication-efficient **Fed**erated learning algorithm with **P**eriodic **A**veraging and **Q**uantization, which addresses federated learning systems' bottlenecks. In particular, `FedPAQ` has three key features that enable efficient federated learning implementation:

(1) `FedPAQ` allows the nodes of the network to run local training before synchronizing with the parameter server. In particular, each node iteratively updates its local model for a period of iterations using the stochastic gradient descent (SGD) method and then uploads its model to the parameter server where all the received models are averaged periodically. By tuning the parameter which corresponds to the number of local iterations before communicating to the server, periodic averaging results in slashing the number of communication rounds and hence the total communication cost of the training process.

(2) `FedPAQ` captures the constraint on availability of active edge nodes by allowing a partial node participation. That is, in each round of the method, only a fraction of the total devices–which are the active ones–contribute to train the model. This procedure

not only addresses the scalability challenge, but also leads to smaller communication load compared to the case that all nodes participate in training the learning model.

(3) In `FedPAQ`, nodes only send a quantized version of their local information to the server at each round of communication. As the training models are of large sizes, quantization significantly helps reducing the communication overhead on the network.

Though these features have been proposed in the literature, to the best of our knowledge, `FedPAQ` is the first federated learning algorithm that simultaneously incorporates these features and provides theoretical convergence guarantees, while being communication-efficient via periodic averaging, partial node participation and quantization.

In particular, we analyze our proposed `FedPAQ` method for two general class of loss functions: strongly-convex and non-convex. For the strongly-convex setting, we show that after T iterations the squared norm of the distance between the solution of our method and the optimal solution is of $\mathcal{O}(1/T)$ in expectation. We also show that `FedPAQ` approaches a first-order stationary point for non-convex losses at a rate of $\mathcal{O}(1/\sqrt{T})$. This demonstrates that our method significantly improves the communication-efficiency of federated learning while preserving the optimality and convergence guarantees of the baseline methods. In addition, we would like to highlight that our theoretical analysis is based on few relaxed and customary assumptions which yield more technical challenges compared to the existing works with stronger assumptions and hence acquires novel analytical techniques. More explanations will be provided in Section 2.4.

Related Work. The main premise of federated learning has been collective learning using a network of common devices such as phones and tablets. This framework potentially allows for smarter models, lower latency, and less power consumption, all while ensuring privacy. Successfully achieving these goals in practice requires addressing key challenges of federated learning such as communication complexity, systems heterogeneity, privacy, robustness, and heterogeneity of the users. Recently, many federated methods have been considered in the literature which mostly aim at reducing the communication cost. [30] proposed the `FedAvg` algorithm, where the global model is updated by averaging local SGD updates. [31] proposed one-shot federated learning in which the master node learns the model after a single round of communication.

Optimization methods for federated learning are naturally tied with tools from stochastic and distributed optimization. Minibatch stochastic gradient descent distributed optimization methods have been largely studied in the literature without considering the communication bottleneck. Addressing the communication bottleneck via quantization and compression in distributed learning has recently gained considerable attention for both master-worker [32–35] and masterless topologies [36–39]. Moreover, [39] reduces the communication delay by decomposing the graph.

Local updates, as another approach to reduce the communication load in distributed learning has been studied in the literature, where each learning node carries out multiple local updates before sharing with the master or its neighboring nodes. [40] considered a master-worker topology and provides theoretical analysis for the convergence of local-SGD method. [41] introduced a variant of local-SGD namely post-local-SGD which demonstrates empirical improvements over local-SGD. [42] provided a general analysis of such cooperative method for decentralized settings as well.

2.2 Federated Learning Setup

In this chapter, we focus on a federated architecture where a parameter server (or server) aims at finding a model that performs well with respect to the data points that are available at different nodes (users) of the network, while nodes exchange their local information with the server. We further assume that the data points for all nodes in the network are generated from a common probability distribution. In particular, we consider the following stochastic learning problem

$$\min_{\mathbf{w}} f(\mathbf{w}) \coloneqq \frac{1}{n} \sum_{i=1}^{n} f_i(\mathbf{w}), \tag{2.1}$$

where the local objective function of each node i is defined as the expected loss of its local sample distributions

$$f_i(\mathbf{w}) \coloneqq \mathbb{E}_{\xi \sim \mathcal{P}^i} \left[\ell(\mathbf{w}, \xi) \right]. \tag{2.2}$$

Here $\ell : \mathbb{R}^p \times \mathbb{R}^u \to \mathbb{R}$ is a stochastic loss function, $\mathbf{w} \in \mathbb{R}^p$ is the model vector, and $\xi \in \mathbb{R}^u$ is a random variable with unknown probability distribution $\mathcal{P}^i$. Moreover, $f : \mathbb{R}^p \to \mathbb{R}$ denotes the expected loss function also called population risk. In our considered federated setting, each of the n distributed nodes generates a local loss function according to a distribution $\mathcal{P}^i$ resulting in a local stochastic function $f_i(\mathbf{w}) \coloneqq \mathbb{E}_{\xi \sim \mathcal{P}^i} \left[\ell(\mathbf{w}, \xi) \right]$. A special case of this formulation is when each node i maintains a collection of m samples from distribution $\mathcal{P}^i$ which we denote by $\mathcal{D}^i = \{\xi_1^i, \cdots, \xi_m^i\}$ for $i \in [n]$. This results in the following empirical risk minimization problem over the collection of nm samples in $\mathcal{D} \coloneqq \mathcal{D}^1 \cup \cdots \cup \mathcal{D}^n$:

$$\min_{\mathbf{w}} L(\mathbf{w}) = \min_{\mathbf{w}} \frac{1}{nm} \sum_{\xi \in \mathcal{D}} \ell(\mathbf{w}, \xi), \tag{2.3}$$

We denote the optimal model $\mathbf{w}^*$ as the solution to the expected risk minimization problem in (4.1) and denote the minimum loss $f^* \coloneqq \min_{\mathbf{w}} f(\mathbf{w}) = f(\mathbf{w}^*)$ as the optimal objective function value of the expected risk minimization problem in (4.1). In this work, we focus on the case that the data over the n nodes is independent and identically distributed (i.i.d.), which implies the local distributions are common.

As stated above, our goal is to minimize the expected loss $f(\mathbf{w})$. However, due to the fact that we do not have access to the underlying distribution $\mathcal{P}$, there have been prior works that focus on minimizing the empirical risk $L(\mathbf{w})$ which can be viewed as an approximation of the expected loss $f(\mathbf{w})$. The accuracy of this approximation is determined by the number of samples nm. It has been shown that for convex losses ℓ, the population risk f is at most $\mathcal{O}(1/\sqrt{nm})$ distant from the empirical risk L, uniformly and with high probability [43]. That is, $\sup_{\mathbf{w}} |f(\mathbf{w}) - L(\mathbf{w})| \leq \mathcal{O}(1/\sqrt{nm})$ with high probability. This result implies that if each of the n nodes separately minimizes its local empirical loss function, the expected deviation from the local solution and the solution to the population risk minimization problem is of $\mathcal{O}(1/\sqrt{m})$ (note that each node has access to m data samples). However, if the nodes manage to somehow share or synchronize their solutions, then a more accurate solution can be achieved, that is a solution with accuracy of order $\mathcal{O}(1/\sqrt{nm})$. Therefore, when all the mn available samples are leveraged, one can obtain a solution $\hat{\mathbf{w}}$ that satisfies $\mathbb{E}[L(\hat{\mathbf{w}}) - L(\mathbf{w}^*)] \leq \mathcal{O}(1/\sqrt{nm})$. This also implies that $\mathbb{E}[f(\hat{\mathbf{w}}) - \min_{\mathbf{w}} f(\mathbf{w})] \leq \mathcal{O}(1/\sqrt{nm})$.

For the case of non-convex loss function ℓ, however, finding the solution to the expected risk minimization problem in (4.1) is hard. Even further, finding (or testing) a local optimum is NP-hard in many cases [44]. Therefore, for non-convex losses we relax our main goal and instead look for first-order optimal solutions (or stationary points) for (4.1). That is, we aim to find a model $\hat{\mathbf{w}}$ that satisfies $\left\|\nabla f(\hat{\mathbf{w}})\right\| \leq \epsilon$ for an arbitrarily small approximation error ϵ. [45] characterized the gap for the gradients of the two expected risk and empirical risk functions. That is, if the gradient of loss is sub-Gaussian, then with high probability $\sup_{\mathbf{w}}\left\|\nabla L(\mathbf{w}) - \nabla f(\mathbf{w})\right\| \leq \mathcal{O}(1/\sqrt{nm})$. This result further implies that having all the nodes contribute in minimizing the empirical risk results in better approximation for a first-order stationary point of the expected risk L. In summary, our goal in non-convex setting is to find $\hat{\mathbf{w}}$ that satisfies $\left\|\nabla f(\mathbf{w})\right\| \leq \mathcal{O}(1/\sqrt{nm})$ which also implies $\left\|\nabla L(\mathbf{w})\right\| \leq \mathcal{O}(1/\sqrt{nm})$.

2.3 Proposed `FedPAQ` Method

In this section, we present our proposed communication-efficient federated learning method called `FedPAQ`, which consists of three main modules: (1) periodic averaging, (2) partial node participation, and (3) quantized message passing.

2.3.1 Periodic averaging

As explained in Section 2.2, to leverage from all the available data samples on the nodes, any training method should incorporate synchronizing the intermediate models obtained at local devices. One approach is to let the participating nodes synchronize their models through the parameter server in each iteration of the training. This, however, implies many rounds of communication between the federated nodes and the parameter server which results in communication contention over the network. Instead, we let the participating nodes conduct a number of local updates and synchronize through the parameter server periodically. To be more specific, once nodes pull an updated model from the server, they update the model locally by running τ iterations of the SGD method and then send proper information to the server for updating the aggregate model. Indeed, this periodic averaging scheme reduces the rounds of communication between server and the nodes and consequently the overall communication cost of training the model. In particular, for the case that we plan to run T iterations of SGD at each node, nodes need to communicate with the server $K = T/\tau$ rounds, hence reducing the total communication cost by a factor of $1/\tau$.

Choosing a larger value of τ indeed reduces the rounds of communication for a fixed number of iterations T. However, if our goal is to obtain a specific accuracy ε, choosing a very large value for τ is not necessarily optimal as by increasing τ the noise of the system increases and the local models approach the local optimal solutions instead of the global optimal solution. Hence, we might end up running more iterations T to achieve a specific accuracy ε comparing to a case that τ is small. Indeed, a crucial question that we need to address is finding the optimal choice of τ for minimizing the overall communication cost of the process.

2.3.2 Partial node participation

In a federated network, often there is a large number of devices such as smart phones communicating through a base station. On one hand, base stations have limited download bandwidth and hence only a few of devices are able to simultaneously upload their messages to the base station. Due to this limitation the messages sent from the devices will be pipelined at the base station which results in a dramatically slow training. On the other hand, having all of the devices participate through the whole training process induces a large communication overhead on the network which is often costly. Moreover, in practice not all the devices contribute in each round of the training. Indeed, there are multiple factors that determine whether a device can participate in the training [1]: a device should be available in the reachable range of the base station; a device should be idle, plugged in and connected to a free wireless network during the training; etc.

Our proposed `FedPAQ` method captures the restrictions mentioned above. In particular, we assume that among the total of n devices, only r nodes ($r \leq n$) are available in each round of the training. We can also assume that due to the availability criterion described before, such available devices are randomly and uniformly distributed over the network [46]. In summary, in each period $k = 0, 1, \cdots, K-1$ of the training algorithm, the parameter server sends its current model $\mathbf{w}_k$ to all the r nodes in subset $\mathcal{S}_k$, which are distributed uniformly at random among the total n nodes, i.e., $\Pr[\mathcal{S}_k] = 1/\binom{n}{r}$.

2.3.3 Quantized message-passing

Another aspect of the communication bottleneck in federated learning is the limited uplink bandwidth at the devices which makes the communication from devices to the parameter server slow and expensive. Hence, it is critical to reduce the size of the uploaded messages from the federated devices [25]. Our proposal is to employ quantization operators on the transmitted massages. Depending on the accuracy of the quantizer, the network communication overhead is reduced by exchanging the quantized updates.

In the proposed `FedPAQ`, each node $i \in \mathcal{S}_k$ obtains the model $\mathbf{w}_{k,\tau}^{(i)}$ after running τ local iterations of an optimization method (possibly SGD) on the most recent model $\mathbf{w}_k$ that it has received form the server. Then each node i applies a quantizer operator $Q(\cdot)$ on the difference between the received model and its updated model, i.e., $\mathbf{w}_{k,\tau}^{(i)} - \mathbf{w}_k$, and uploads the quantized vector $Q(\mathbf{w}_{k,\tau}^{(i)} - \mathbf{w}_k)$ to the parameter server. Once these quantized vectors are sent to the server, it decodes the quantized signals and combines them to come up with a new model $\mathbf{w}_{k+1}$.

Next, we describe a widely-used random quantizer.

Example 2.1 (Low-precision quantizer [32]) *For any variable $\mathbf{w} \in \mathbb{R}^p$, the low precision quantizer $Q^{LP} : \mathbb{R}^p \to \mathbb{R}^p$ is defined as below*

$$Q_i^{LP}(\mathbf{w}) = \|\mathbf{w}\| \cdot \operatorname{sign}(x_i) \cdot \xi_i(\mathbf{w}, s), \quad i \in [p], \tag{2.4}$$

where $\xi_i(\mathbf{w}, s)$ is a random variable taking on value $\frac{l+1}{s}$ with probability $\frac{|x_i|}{\|\mathbf{w}\|}s - l$ and $\frac{l}{s}$ otherwise. Here, the tuning parameter s corresponds to the number of quantization levels and $l \in [0, s)$ is an integer such that $\frac{|x_i|}{\|\mathbf{w}\|} \in [\frac{l}{s}, \frac{l+1}{s})$.

2.3.4 Algorithm update

Now we use the building blocks developed in Sections 2.3.1-2.3.3 to precisely present `FedPAQ`. Our proposed method consists of K periods, and during a period, each node performs τ local updates, which results in total number of $T = K\tau$ iterations. In each period $k = 0, \cdots, K-1$ of the algorithm, the parameter server picks $r \leq n$ nodes uniformly at random which we denote by $\mathcal{S}_k$. The parameter server then broadcasts its current model $\mathbf{w}_k$ to all the nodes in $\mathcal{S}_k$ and each node $i \in \mathcal{S}_k$ performs τ local SGD updates using its local dataset. To be more specific, let $\mathbf{w}_{k,t}^{(i)}$ denote the model at node i at t-th iteration of the k-th period. At each local iteration $t = 0, \cdots, \tau - 1$, node i updates its local model according to the following rule:

$$\mathbf{w}_{k,t+1}^{(i)} = \mathbf{w}_{k,t}^{(i)} - \eta_{k,t} \widetilde{\nabla} f_i \left(\mathbf{w}_{k,t}^{(i)} \right), \tag{2.5}$$

where the stochastic gradient $\widetilde{\nabla} f_i$ is computed using a random sample[1] picked from the local dataset $\mathcal{D}^i$. Note that all the nodes begin with a common initialization $\mathbf{w}_{k,0}^{(i)} = \mathbf{w}_k$. After τ local updates, each node computes the overall update in that period, that is $\mathbf{w}_{k,\tau}^{(i)} - \mathbf{w}_k$, and uploads a quantized update $Q(\mathbf{w}_{k,\tau}^{(i)} - \mathbf{w}_k)$ to the parameter server. The parameter server then aggregates the r received quantized local updates and computes the next model according to

$$\mathbf{w}_{k+1} = \mathbf{w}_k + \frac{1}{r} \sum_{i \in \mathcal{S}_k} Q\left(\mathbf{w}_{k,\tau}^{(i)} - \mathbf{w}_k \right), \tag{2.6}$$

and the procedure is repeated for K periods. The proposed method is formally summarized in Algorithm 2.1.

[1]The method can be easily made compatible with using a mini-batch during each iteration.

Algorithm 2.1: `FedPAQ`

```
for $k = 0, 1, \cdots, K-1$ do
    server picks $r$ nodes $\mathcal{S}_k$ uniformly at random
    server sends $\mathbf{w}_k$ to nodes in $\mathcal{S}_k$
    for node $i \in \mathcal{S}_k$ do
        $\mathbf{w}_{k,0}^{(i)} \leftarrow \mathbf{w}_k$
        for $t = 0, 1, \cdots, \tau - 1$ do
            compute stochastic gradient $\widetilde{\nabla} f_i(\mathbf{w}) = \nabla \ell(\mathbf{w}, \xi)$ for a $\xi \in \mathcal{P}^i$
            set $\mathbf{w}_{k,t+1}^{(i)} \leftarrow \mathbf{w}_{k,t}^{(i)} - \eta_{k,t} \widetilde{\nabla} f_i(\mathbf{w}_{k,t}^{(i)})$
        end
        send $Q(\mathbf{w}_{k,\tau}^{(i)} - \mathbf{w}_k)$ to the server
    end
    server finds $\mathbf{w}_{k+1} \leftarrow \mathbf{w}_k + \frac{1}{r} \sum_{i \in \mathcal{S}_k} Q(\mathbf{w}_{k,\tau}^{(i)} - \mathbf{w}_k)$
end
```

2.4 Convergence Analysis

In this section, we present our theoretical results on the guarantees of the `FedPAQ` method. We first consider the strongly convex setting and state the convergence guarantee of `FedPAQ` for such losses in Theorem 2.1. Then, in Theorem 2.2, we present the overall complexity of our method for finding a first-order stationary point of the aggregate objective function f, when the loss function ℓ is non-convex (All proofs are provided in the supplementary material). Before that, we first mention three customary assumptions required for both convex and non-convex settings.

Assumption 2.1 *The random quantizer $Q(\cdot)$ is unbiased and its variance grows with the squared of l_2-norm of its argument, i.e.,*

$$\mathbb{E}\left[Q(\mathbf{w})|\mathbf{w}\right] = \mathbf{w}, \quad \mathbb{E}\left[\left\|Q(\mathbf{w}) - \mathbf{w}\right\|^2 |\mathbf{w}\right] \leq q\|\mathbf{w}\|^2, \tag{2.7}$$

for some positive real constant q and any $\mathbf{w} \in \mathbb{R}^p$.

Assumption 2.2 *The loss functions f_i are L-smooth with respect to $\mathbf{w}$, i.e., for any $\mathbf{w}, \mathbf{w}' \in \mathbb{R}^p$, we have $\left\|\nabla f_i(\mathbf{w}) - \nabla f_i(\mathbf{w}')\right\| \leq L\|\mathbf{w} - \mathbf{w}'\|$.*

Assumption 2.3 *Stochastic gradients $\widetilde{\nabla} f_i(\mathbf{w})$ are unbiased and variance bounded, i.e., $\mathbb{E}_\xi[\widetilde{\nabla} f_i(\mathbf{w})] = \nabla f_i(\mathbf{w})$ and $\mathbb{E}_\xi[\|\widetilde{\nabla} f_i(\mathbf{w}) - \nabla f_i(\mathbf{w})\|^2] \leq \sigma^2$.*

The conditions in Assumption 2.1 ensure that output of quantization is an unbiased estimator of the input with a variance that is proportional to the norm-squared of the input. This condition is satisfied with most common quantization schemes including the low-precision quantizer introduced in Example 1. Assumption 2.2 implies that the gradients of local functions ∇f_i and the aggregated objective function ∇f are also L-Lipschitz continuous. The conditions in Assumption 2.3 on the bias and variance of stochastic gradients are also customary. Note that this is a much weaker assumption compared to the one that uniformly bounds the expected norm of the stochastic gradient.

Challenges in analyzing the `FedPAQ` method.

Here, we highlight the main theoretical challenges in proving our main results. As outlined in the description of the proposed method, in the k-th round of `FedPAQ`, each participating node i updates its local model for τ iterations via SGD method in (2.5). Let us focus on a case that we use a constant stepsize for the purpose of this discussion. First consider the naive parallel SGD case which corresponds to $\tau = 1$. The updated local model after $\tau = 1$ local update is

$$\mathbf{w}_{k,\tau}^{(i)} = \mathbf{w}_{k,0}^{(i)} - \eta \widetilde{\nabla} f_i\left(\mathbf{w}_{k,0}^{(i)}\right). \tag{2.8}$$

Note that $\mathbf{w}_{k,0}^{(i)} = \mathbf{w}_k$ is the parameter server's model sent to the nodes. Since we assume the stochastic gradients are unbiased estimators of the gradient, it yields that the local update $\mathbf{w}_{k,\tau}^{(i)} - \mathbf{w}_k$ is an unbiased estimator of $-\eta\nabla f(\mathbf{w}_k)$ for every participating node.

Hence, the aggregated updates at the server and the updated model $\mathbf{w}_{k+1}$ can be simply related to the current model $\mathbf{w}_k$ as one step of parallel SGD. However, this is not the case when the period length τ is larger than 1. For instance, in the case that $\tau = 2$, the local updated model after $\tau = 2$ iterations is

$$\mathbf{w}_{k,\tau}^{(i)} = \mathbf{w}_k - \eta \widetilde{\nabla} f_i\left(\mathbf{w}_k\right) - \eta \widetilde{\nabla} f_i\left(\mathbf{w}_k - \eta \widetilde{\nabla} f_i\left(\mathbf{w}_k\right)\right). \tag{2.9}$$

Clearly, $\mathbf{w}_{k,\tau}^{(i)} - \mathbf{w}_k$ is not an unbiased estimator of $-\eta\nabla f(\mathbf{w}_k)$ or $-\eta\nabla f(\mathbf{w}_k - \eta\nabla f(\mathbf{w}_k))$. This demonstrates that the aggregated model at server cannot be treated as τ iterations of parallel SGD, since each local update contains a bias. Indeed, this bias gets propagated when τ gets larger. For our running example $\tau = 2$, the variance of the bias, i.e. $\mathbb{E}\|\eta\widetilde{\nabla} f_i(\mathbf{w}_k - \eta\widetilde{\nabla} f_i(\mathbf{w}_k))\|^2$ is not uniformly bounded either (Assumption 2.3), which makes the analysis even more challenging compared to the works with bounded gradient assumption (e.g. [40, 47]).

2.4.1 Strongly convex setting

Now we proceed to establish the convergence rate of the proposed `FedPAQ` method for a federated setting with strongly convex and smooth loss function ℓ. We first formally state the strong convexity assumption.

Assumption 2.4 *The loss functions f_i are μ-strongly convex, i.e., for any $\mathbf{w}, \mathbf{w}' \in \mathbb{R}^p$ we have that $\langle \nabla f_i(\mathbf{w}) - \nabla f_i(\mathbf{w}'), \mathbf{w} - \mathbf{w}' \rangle \geq \mu \|\mathbf{w} - \mathbf{w}'\|^2$.*

Theorem 2.1 (Strongly convex loss) *Consider the sequence of iterates $\mathbf{w}_k$ at the parameter server generated according to the* `FedPAQ` *method outlined in Algorithm 2.1. Suppose the conditions in Assumptions 2.1–2.4 are satisfied. Further, let us define the* *constant B_1 as*

$$B_1 = 2L^2\left(\frac{q}{n} + \frac{n-r}{r(n-1)}4(1+q)\right), \tag{2.10}$$

where q is the quantization variance parameter defined in (2.7) and r is the number of active nodes at each round of communication. If we set the stepsize in `FedPAQ` *as $\eta_{k,t} = \eta_k = \frac{4\mu^{-1}}{k\tau+1}$, then for any $k \geq k_0$ where k_0 is the smallest integer satisfying*

$$k_0 \geq 4\max\left\{\frac{L}{\mu}, 4\left(\frac{B_1}{\mu^2}+1\right), \frac{1}{\tau}, \frac{4n}{\mu^2\tau}\right\}, \tag{2.11}$$

the expected error $\mathbb{E}[\|\mathbf{w}_k - \mathbf{w}^*\|]^2$ is bounded above by

$$\mathbb{E}\|\mathbf{w}_k - \mathbf{w}^*\|^2 \leq \frac{(k_0\tau+1)^2}{(k\tau+1)^2}\|\mathbf{w}_{k_0} - \mathbf{w}^*\|^2 + C_1\frac{\tau}{k\tau+1} + C_2\frac{(\tau-1)^2}{k\tau+1} + C_3\frac{\tau-1}{(k\tau+1)^2}, \tag{2.12}$$

where the constants in (2.12) are defined as

$$\begin{aligned} C_1 &= \frac{16\sigma^2}{\mu^2 n}\left(1 + 2q + 8(1+q)\frac{n(n-r)}{r(n-1)}\right), \\ C_2 &= \frac{16eL^2\sigma^2}{\mu^2 n}, \\ C_3 &= \frac{256eL^2\sigma^2}{\mu^4 n}\left(n + 2q + 8(1+q)\frac{n(n-r)}{r(n-1)}\right). \end{aligned} \tag{2.13}$$

Proof: We defer the proof to Appendix A.1. ■

Remark 2.1 *Under the same conditions as in Theorem 2.1 and for a total number of iterations $T = K\tau \geq k_0\tau$ we have the following convergence rate*

$$\mathbb{E}\|\mathbf{w}_K - \mathbf{w}^*\|^2 \leq \mathcal{O}\left(\frac{\tau}{T}\right) + \mathcal{O}\left(\frac{\tau^2}{T^2}\right) + \mathcal{O}\left(\frac{(\tau-1)^2}{T}\right) + \mathcal{O}\left(\frac{\tau-1}{T^2}\right). \tag{2.14}$$

As expected, the fastest convergence rate is attained when the contributing nodes syn-

chronize with the parameter server in each iteration, i.e. when $\tau = 1$. Theorem 2.1 however characterizes how large the period length τ can be picked. In particular, any pick of $\tau = o(\sqrt{T})$ ensures the convergence of the `FedPAQ` *to the global optimal for strongly convex losses.*

Remark 2.2 *By setting $\tau = 1$, $q = 0$ and $r = n$, Theorem 2.1 recovers the convergence rate of vanilla parallel SGD, i.e., $\mathcal{O}(1/T)$ for strongly-convex losses. Our result is however more general since we remove the uniformly bounded assumption on the norm of stochastic gradient. For $\tau \geq 1$, Theorem 2.1 does not recover the result in [40] due to our weaker condition in Assumption 2.3. Nevertheless, the same rate $\mathcal{O}(1/T)$ is guaranteed by* `FedPAQ` *for constant values of τ.*

2.4.2 Non-convex setting

We now present the convergence result of `FedPAQ` for smooth non-convex losses.

Theorem 2.2 (Non-convex Losses) *Consider the sequence of iterates $\mathbf{w}_k$ at the parameter server generated according to the* `FedPAQ` *method outlined in Algorithm 2.1. Suppose the conditions in Assumptions 2.1–2.3 are satisfied. Further, let us define the constant B_2 as*

$$B_2 \coloneqq \frac{q}{n} + \frac{4(n-r)}{r(n-1)}(1+q), \tag{2.15}$$

where q is the quantization variance parameter defined in (2.7) and r is the number of active nodes at each round. If the total number of iterations T and the period length τ satisfy the following conditions,

$$T \geq 2, \qquad \tau \leq \frac{\sqrt{B_2^2 + 0.8} - B_2}{8}\sqrt{T}, \tag{2.16}$$

and we set the stepsize as $\eta_{k,t} = \frac{1}{L\sqrt{T}}$, then the following first-order stationary condition *holds*

$$\frac{1}{T}\sum_{k=0}^{K-1}\sum_{t=0}^{\tau-1}\mathbb{E}\left\|\nabla f(\overline{\mathbf{w}}_{k,t})\right\|^2 \leq \frac{2L(f(\mathbf{w}_0)-f^*)}{\sqrt{T}} + N_1\frac{1}{\sqrt{T}} + N_2\frac{\tau-1}{T}, \tag{2.17}$$

where the constants in (7.4) *are defined as*

$$N_1 := (1+q)\frac{\sigma^2}{n}\left(1+\frac{n(n-r)}{r(n-1)}\right), \quad N_2 := \frac{\sigma^2}{n}(n+1).$$

Proof: We defer the proof to Appendix A.2. ■

Remark 2.3 *The result in Theorem 2.2 implies the following order-wise rate*

$$\frac{1}{T}\sum_{k=0}^{K-1}\sum_{t=0}^{\tau-1}\mathbb{E}\left\|\nabla f(\overline{\mathbf{w}}_{k,t})\right\|^2 \leq \mathcal{O}\left(\frac{1}{\sqrt{T}}\right) + \mathcal{O}\left(\frac{\tau-1}{T}\right).$$

Clearly, the fastest convergence rate is achieved for the smallest possible period length, i.e., $\tau = 1$. This however implies that the edge nodes communicate with the parameter server in each iteration, i.e. T rounds of communications which is costly. On the other hand, the conditions (2.16) *in Theorem 2.2 allow the period length τ to grow up to $\mathcal{O}(\sqrt{T})$ which results in an overall convergence rate of $\mathcal{O}(1/\sqrt{T})$ in reaching an stationary point. This result shows that with only $\mathcal{O}(\sqrt{T})$ rounds of communication* `FedPAQ` *can still ensure the convergence rate of $\mathcal{O}(1/\sqrt{T})$ for non-convex losses.*

Remark 2.4 *Theorem 2.2 recovers the convergence rate of the vanilla parallel SGD [47] for non-convex losses as a special case of $\tau = 1$, $q = 0$ and $r = n$. Nevertheless, we remove the uniformly bounded assumption on the norm of the stochastic gradient in our theoretical analysis. We also recover the result in [42] when there is no quatization $q = 0$ and we have a full device participation $r = n$.*

It is worth mentioning that for Theorems 2.1 and 2.2, one can use a batch of size m for each local SGD update and the same results hold by changing $\frac{\sigma^2}{n}$ to $\frac{\sigma^2}{mn}$.

2.5 Numerical Results and Discussions

The proposed `FedPAQ` method reduces the communication load by employing three modules: periodic averaging, partial node participation, and quantization. This communication reduction however comes with a cost in reducing the convergence accuracy and hence requiring more iterations of the training, which we characterized in Theorems 2.1 and 2.2. In this section, we empirically study this communication-computation trade-off and evaluate `FedPAQ` in comparison to other benchmarks. To evaluate the total cost of a method, we first need to specifically model such cost. We consider the total training time as the cost objective which consists of communication and computation time [48,49]. Consider T iterations of training with `FedPAQ` that consists of $K = T/\tau$ rounds of communication. In each round, r workers compute τ iterations of SGD with batchsize B and send a quantized vector of size p to the server.

Communication time. We fix a bandwidth `BW` and define the communication time in each round as the total number of uploaded bits divided by `BW`. Total number of bits in each round is $r \cdot |Q(p,s)|$, where $|Q(p,s)|$ denotes the number of bits required to encode a quantized vector of dimension p according to a specific quantizer with s levels. In our simulations, we use the low-precision quantizer described in Example 2.1 and assume it takes pF bits to represent an unquantized vector of length p, where F is typically 32 bits.

Computation time. We consider the well-known shifted-exponential model for gradient computation time [50]. In particular, we assume that for any node, computing the gradients in a period with τ iterations and using batchsize B takes a deterministic shift $\tau \cdot B \cdot \texttt{shift}$ plus a random exponential time with mean value $\tau \cdot B \cdot \texttt{scale}^{-1}$, where

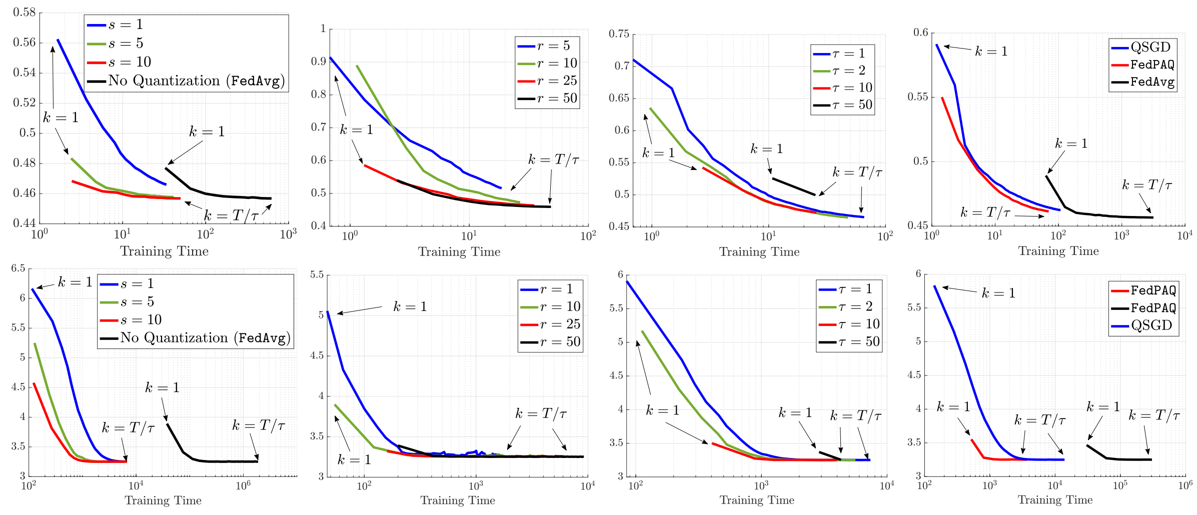


Figure 2.1: Training Loss vs. Training Time: Logistic Regression on MNIST (top). Neural Network on CIFAR-10 (bottom).

`shift` and `scale` are respectively shift and scale parameters of the shifted-exponential distribution. Total computation time of each round is then the largest local computation time among the r contributing nodes. We also define a communication-computation ratio

$$\frac{C_{\texttt{comm}}}{C_{\texttt{comp}}} = \frac{pF/\texttt{BW}}{\texttt{shift} + 1/\texttt{scale}}$$

as the communication time for a length-p-vector over the average computation time for one gradient vector. This ratio captures the relative cost of communication and computation, and since communication is a major bottleneck, we have $C_{\texttt{comm}}/C_{\texttt{comp}} \gg 1$. In the experiments, we use batchsize $B = 10$ and finely tune the stepsize's coefficient.

Logistic Regression on MNIST: In Figure 2.1, the top four plots demonstrate the training time for a regularized logistic regression problem over MNIST dataset ('0' and '8' digits) for $T = 100$ iterations. The network has $n = 50$ nodes each loaded with 200 samples. We set $C_{\texttt{comm}}/C_{\texttt{comp}} = 100/1$ to capture the communication bottleneck. Among the three parameters quantization levels s, number of active nodes in each round r, and period length τ, we fix two and vary the third one. First plot demonstrates the relative training loss for different quantization levels $s \in \{1, 5, 10\}$ and the case with no quantization which corresponds to the FedAvg method [30]. The other two parameters are fixed to $(\tau, r) = (5, 25)$. Each curve shows the training time versus the achieved training loss for the aggregated model at the server for each round $k = 1, \cdots, T/\tau$. In the second plot, $(s, \tau) = (1, 5)$ are fixed. The third plot demonstrates the effect of period length τ in the communication-computation tradeoff. As demonstrated, after T/τ rounds, smaller choices for τ (e.g. $\tau = 1, 2$) result in slower convergence while the larger ones (e.g. $\tau = 50$) run faster though providing less accurate models. Here $\tau = 10$ is the optimal choice. The last plot compares the training time of FedPAQ with two other benchmarks FedAvg and QSGD. For both FedPAQ and FedAvg, we set $\tau = 2$ while FedPAQ and QSGD use quantization with $s = 1$ level. All three methods use $r = n = 50$ nodes in each round.

Neural Network training over CIFAR-10: We conduct another set of numerical experiments to evaluate the performance of FedPAQ on non-convex and smooth objectives. Here we train a neural network with four hidden layers consisting of $n = 50$ nodes and more thatn 92K parameters, where we use 10K samples from CIFAR-10 dataset with 10 labels. Since models are much larger than the previous setup, we increase the communication-computation ratio to $C_{\texttt{comm}}/C_{\texttt{comp}} = 1000/1$ to better capture the communication bottleneck for large models. The bottom four plots in Figure 2.1 demonstrate the training loss over time for $T = 100$ iterations. In the first plot, $(\tau, r) = (2, 25)$ are fixed and we vary the quantization levels. The second plot shows the effect of r while $(s, \tau) = (1, 2)$. The communication-computation tradeoff in terms of period length τ is demonstrated in the third plot, where picking $\tau = 10$ turns out to attain the fastest convergence. Lastly, we compare FedPAQ with other benchmarks in the forth plot. Here, we set $(s, r, \tau) = (1, 20, 10)$ in FedPAQ, $(r, \tau) = (20, 10)$ in FedAvg and $(s, r, \tau) = (1, 50, 1)$ for QSGD.

Additional numerical results: To further illustrate the practical performance of the proposed `FedPAQ` method, here we provide more numerical results using different and more complicated datasets and model parameters. The network settings, communication and computation time models remain the same as those in Section 6.4. The following figures demonstrate the training time corresponding to the following scenarios:

- Figure 2.2: Training time of a neural network with four hidden layers and more than 248K parameters over 10K samples of the CIFAR-10 dataset with 10 labels.
- Figure 2.3: Training time of a neural network with one hidden layer over 10K samples of the CIFAR-100 dataset with 100 labels.
- Figure 2.4: Training time of a neural network with one hidden layer over 10K samples of the Fashion-MNIST dataset with 10 labels.

Here as well, in all of the above scenarios, the data samples are uniformly distributed among $n = 50$ nodes. We also keep the communication-computation ratio and the batchsize to be $C_{\texttt{comm}}/C_{\texttt{comp}} = 1000/1$ and $B = 10$ respectively, and finely tune the stepsize for every training.

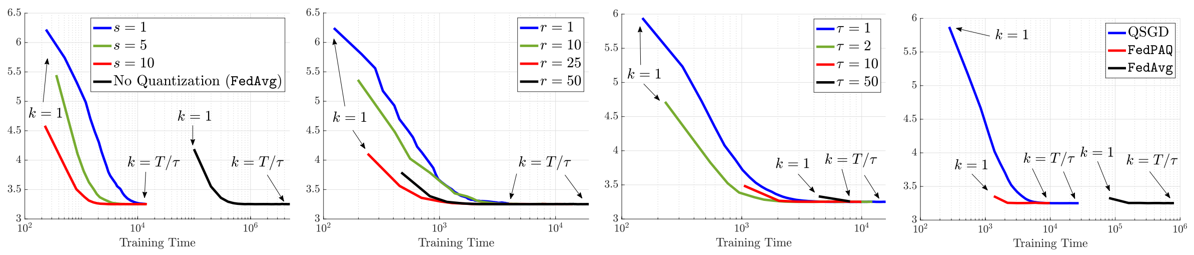


Figure 2.2: Training Loss vs. Training Time: Neural Network on CIFAR-10 dataset with 248K parameters.

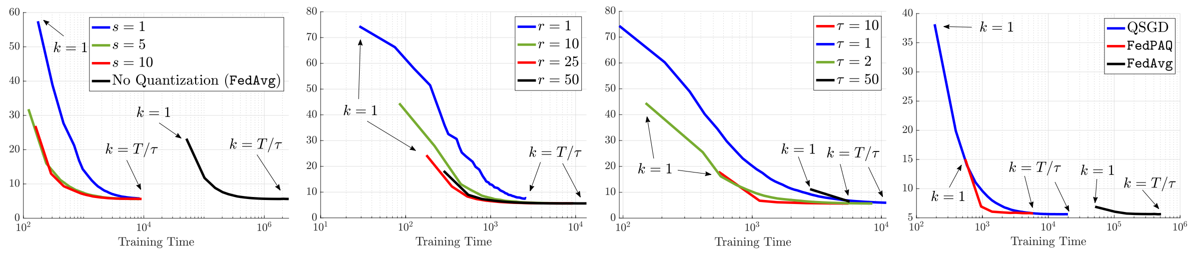


Figure 2.3: Training Loss vs. Training Time: Neural Network on CIFAR-100 dataset.

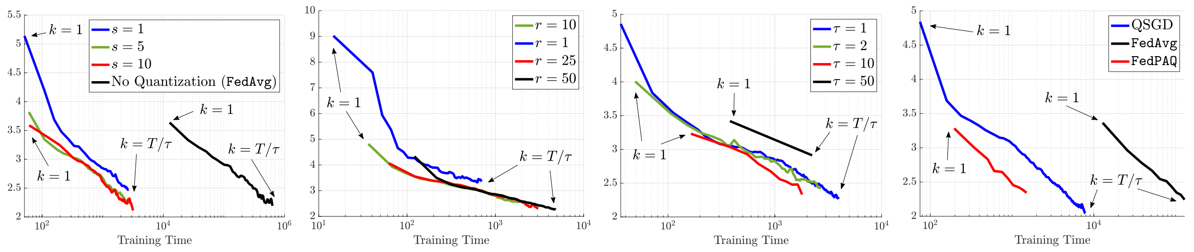


Figure 2.4: Training Loss vs. Training Time: Neural Network on Fashion-MNIST dataset.

2.6 Concluding Remarks

In this chapter, we addressed some of the communication and scalability challenges of federated learning and proposed `FedPAQ`, a communication-efficient federated learning method with provable performance guarantees. `FedPAQ` is based on three modules: (1) periodic averaging in which each edge node performs local iterative updates; (2) partial node participation which captures the random availability of the edge nodes; and (3) quantization in which each model is quantized before being uploaded to the server. We provided rigorous analysis for `FedPAQ` for two general classes of strongly-convex and non-convex losses. We further provided numerical results evaluating the performance of `FedPAQ`, and discussing the trade-off between communication and computation.

Chapter 3

Straggler-Resilient Federated Learning

Federated learning is prone to multiple system challenges including system heterogeneity where clients have different computation and communication capabilities. Such heterogeneity in clients' computation speeds has a negative effect on the scalability of federated learning algorithms and causes significant slow-down in their runtime due to the existence of stragglers. In this chapter, we propose a novel straggler-resilient federated learning method that incorporates statistical characteristics of the clients' data to adaptively select the clients in order to speed up the learning procedure. The key idea of our algorithm is to start the training procedure with faster nodes and gradually involve the slower nodes in the model training once the statistical accuracy of the data corresponding to the current participating nodes is reached. The proposed approach reduces the overall runtime required to achieve the statistical accuracy of data of all nodes, as the solution for each stage is close to the solution of the subsequent stage with more samples and can be used as a warm-start. Our theoretical results characterize the speedup gain in comparison to standard federated benchmarks for strongly convex objectives and i.i.d.

samples (system heterogeneous and data homogeneous), and our numerical experiments also demonstrate significant speedups in wall-clock time of our straggler-resilient method compared to other federated learning benchmarks.

3.1 Introduction

Federated learning is a distributed framework whose objective is to train a model using the data of many clients (nodes), while keeping each node's data local. In contrast with centralized learning, the federated learning architecture allows for preserving the clients' privacy as well as reducing the communication burden caused by transmitting data to a cloud. Nevertheless, as we move towards deploying federated learning in practice, it is becoming apparent that several major challenges still remain and the existing frameworks need to be rethought to address them. Important among these challenges is system (device) heterogeneity due to existence of *straggling nodes* – slow nodes with low computational capability – that significantly slow down the model training [3, 25].

In this chapter, we focus on system heterogeneity in federated learning and we leverage the interplay between statistical accuracy and system heterogeneity to design a straggler-resilient federated learning method that carefully and adaptively selects a subset of available nodes in each round of training. Federated networks consist of thousands of devices with a wide range of computational, communication, battery power, and storage characteristics. Hence, deploying traditional federated learning algorithms such as `FedAvg` [51] on such a highly heterogeneous cluster of devices results in significant and unexpected delays due to existence of slow clients or stragglers. In most of such algorithms, *all* the *available* clients participate in the model training –regardless of their computational capabilities. Consequently, in each communication round of such methods, the server has to wait for the slowest node to complete its local updates and upload its local model which significantly slows down the training process.

In this chapter, we aim to mitigate the effect of stragglers in federated learning based on an adaptive node participation approach in which clients are selected to participate in different stages of training according to their computation speed. We call our straggler-resilient scheme a **F**ederated **L**earning method with **A**daptive **N**ode **P**articipation or `FLANP`. The key idea of this scheme is to start the model training procedure with only a few clients which are the fastest among all the nodes. These participating clients continue to train their shared models while interacting with the parameter server. Note that since the server waits only for the participating nodes, it takes a short time for the participating (and fast) clients to promptly train a shared model. This model is, however, not accurate as it is trained over only a fraction of samples. We next increase the number of participating clients and include the next fastest subset of nonparticipating nodes in the training. Note that the model trained from the previous stage can be a warm-start initialization for the current stage.

To discuss our main idea more precisely, consider a federated network of N available nodes each storing s data samples and suppose that we start the learning procedure with only m clients. Once we solve the empirical risk minimization (ERM) problem corresponding to $m \times s$ samples of these nodes up to its statistical accuracy, we geometrically increase the number of participating nodes to $n = \alpha m$ where $\alpha > 1$, by adding the next $n - m$ fastest clients in the network. By doing so, the new ERM problem that we aim to solve contains the samples from the previous stage as well as the samples of the newly participating nodes. Moreover, the solution for the ERM problem at the previous stage (with m clients) could be used as a warm-start for the ERM problem at the current stage with $n = \alpha m$ nodes. This is due to the fact that all samples are drawn from a common distribution, and as a result, the optimal solution of the ERM problem with fewer samples is not far from the optimal solution of the ERM problems with more samples, as long as the larger set contains the smaller set.

In the proposed FLANP algorithm, as time progresses, we gradually increase the number of participating clients until we reach the full training set and all clients are involved. Note that in this procedure, the slower clients are only used towards the end of the learning process, where the model is already close to the optimal model of the aggregate loss. Another essential observation is that since the model trained in previous rounds already has a reasonable statistical accuracy and this model serves as the initial point of the next round of the iterative algorithm, the slower nodes are only needed to contribute in the final rounds of training, leading to a smaller wall-clock time. This is in contrast with having all nodes participate in training from the beginning, which leads to computation time of each round being determined by the slowest node. In this chapter, we formally characterize the gain obtained by using the proposed adaptive node participation scheme compared to the case that all available nodes contribute to training at each round. Next, we state a summary of our main contributions:

• We present a straggler-resilient federated learning meta-algorithm that leverages the interplay between statistical accuracy and device heterogeneity by adaptively activating heterogeneous clients.

• We specify the proposed meta-algorithm with a federated learning subroutine and present its optimization guarantees for strongly convex risks. Further, we characterize the wall-clock time of the proposed straggler-resilient scheme and demonstrate analytically that it achieves up to $\mathcal{O}(\ln(Ns))$ speedup gain compared to standard benchmarks.

• Our numerical results show that our framework significantly improves the wall-clock time compared to federated learning benchmarks –with either full or partial node participation– for both convex and non-convex risks.

Related Work. System (device) heterogeneity challenge, which refers to the case that clients have different computational, communication and storage characteristics, has been studied in the literature. Asynchronous methods have demonstrated improvements in distributed data centers. However, such methods are less desirable in federated settings as they rely on bounded staleness of slow clients [52, 53]. The active sampling approach is another direction in which the server aims for aggregating as many local updates as possible within a predefined time span [54]. More recently, [55] proposed a normalized averaging method to mitigate stragglers in federated systems and the objective inconsistency due to mismatch in clients' local updates. Deadline-based computation has also been studied to mitigate stragglers in decentralized settings [49]. In a different yet related direction, various federated algorithms have been studied to address the heterogeneity in clients' *data* distributions [6, 8, 56–59].

The idea of adaptive sample size training in which we solve a sequence of geometrically increasing ERM problems has been used previously for solving large-scale ERM problems. In particular, it has been shown that this scheme improves the overall computational cost of both first-order [60, 61] and second-order [62–64] methods for achieving the statistical accuracy of the full training set. In this chapter, we exploit this idea to develop `FLANP` for a different setting to address the issue of device heterogeneity in federated learning.

As mentioned, `FLANP` is a general meta-algorithm that can be employed with any federated learning subroutine studied in the literature [42, 51, 65–76]. In this chapter, we showcase the gain obtained by combining `FLANP` with the `FedGATE` method [56].

3.2 Federated Learning Setup

In this section, we state our setup. Consider a federated architecture where N nodes interact with a central server, and each node $i \in [N] = \{1, \cdots, N\}$ has access to s data samples denoted by $\{z_1^i, \cdots, z_s^i\}$. These samples are drawn at the beginning of the training process, and nodes cannot draw new samples during training. Further, define

$\ell(\cdot,\cdot) : \mathbb{R}^d \times \mathcal{Z} \to \mathbb{R}$ as a loss function where $\ell(\mathbf{w}, z_j^i)$ indicates how well the model $\mathbf{w}$ performs with respect to the sample z_j^i. Also, define the empirical loss of node i as $L^i(\mathbf{w}) \coloneqq \frac{1}{s}\sum_{j=1}^{s} \ell(\mathbf{w}, z_j^i)$. For any $1 \leq n \leq N$, we denote by $L_n(\mathbf{w})$ the collective empirical risk corresponding to samples of all nodes $\{1, \cdots, n\}$, which is defined as

$$L_n(\mathbf{w}) \coloneqq \frac{1}{n}\sum_{i=1}^{n} L^i(\mathbf{w}). \tag{3.1}$$

$L_n(\mathbf{w})$ represents the average loss over the $n \times s$ samples stored at nodes $\{1, \cdots, n\}$. We let $\mathbf{w}_n^*$ denote the optimal minimizer of the loss $L_n(\mathbf{w})$, i.e., $\mathbf{w}_n^* = \arg\min_{\mathbf{w}} L_n(\mathbf{w})$. We assume that the samples z_j^i are i.i.d. realizations of a random variable Z with probability distribution $\mathcal{P}$. The problem of finding a global model for the aggregate loss of all available N nodes, which can be considered as the empirical risk minimization (ERM) in (3.1) for $n = N$, i.e.,

$$\min_{\mathbf{w}} L_N(\mathbf{w}) = \frac{1}{N}\sum_{i=1}^{N} L^i(\mathbf{w}) = \frac{1}{Ns}\sum_{i=1}^{N}\sum_{j=1}^{s} \ell(\mathbf{w}, z_j^i) \tag{3.2}$$

is a surrogate for the expected risk minimization problem $\min_{\mathbf{w}} L(\mathbf{w}) \coloneqq \mathbb{E}_{Z\sim\mathcal{P}}[\ell(\mathbf{w}, Z)]$. Our ultimate goal is to find the optimal solution of the expected risk $\mathbf{w}^* = \arg\min_{\mathbf{w}} L(\mathbf{w})$; however, since distribution $\mathcal{P}$ is unknown and we only have access to a finite number of realizations of the random variable Z, i.e., $\{z_1^i, \cdots, z_s^i\}_{i=1}^N$, we settle for solving problem (3.2).

Let us further clarify the data heterogeneity model used in our setting. Data samples $\{z_1^i, \cdots, z_s^i\}_{i=1}^N$ are i.i.d. (data homogeneous); however, the realized samples are fixed through the learning process. More precisely, empirical risk functions L^i are realized and fixed which yields that the local gradient directions ∇L^i are *not* necessarily an unbiased estimator for the aggregate loss gradient ∇L_N. In other words, nodes do not have access to unbiased estimators of the expected loss and therefore federated learning methods designed merely for i.i.d. data settings would not be useful here. This is indeed a significant challenge in designing fast convergent optimization methods in such settings which we will highlight in Section 3.3.

Statistical Accuracy. The difference of expected and empirical risks $L_n(\mathbf{w}) - L(\mathbf{w})$ is referred to as the estimation error and can be bounded by a function of the sample size. In particular, since $L_n(\mathbf{w})$ captures ns samples, we assume that there exists a constant V_{ns} bounding the estimation error with high probability, $\sup_{\mathbf{w}} |L_n(\mathbf{w}) - L(\mathbf{w})| \leq V_{ns}$.

The estimation error V_{ns} has been deeply studied in the statistical learning literature [77,78]. In particular, it has been shown that for strongly convex functions the estimation error is proportional to the inverse of sample size [79, 80]. In this work, we also assume that $V_{ns} = \frac{c}{ns}$ for a constant c. Note that for the loss function L_n once we find a point $\tilde{\mathbf{w}}$ that has an optimization error of V_{ns}, i.e., $L_n(\tilde{\mathbf{w}}) - L_n(\mathbf{w}_n^*) \leq V_{ns}$, there is no gain in improving the optimization error as the overall error with respect to the expected risk L would not improve. Hence, when we find a point $\tilde{\mathbf{w}}$ such that $L_n(\tilde{\mathbf{w}}) - L_n(\mathbf{w}_n^*) \leq V_{ns}$, we state that it has reached the statistical accuracy of L_n. Our goal is to find a solution $\mathbf{w}_N$ that is within the statistical accuracy of the ERM problem of the full training set defined in (3.2).

System Heterogeneity Model. As mentioned earlier, federated clients attribute a wide range of computational powers leading to significantly different processing times for a fixed computing task such as gradient computation and local model update. To be more specific, for each node $i \in [N]$, we let T_i denote the (expected) time to compute one local model update. The time for such update is mostly determined by the computation time of a fixed batch-size stochastic gradient of the local empirical risk $L_i(\mathbf{w})$. Clearly, larger T_i corresponds to slower clients or stragglers. Without loss of generality, we assume that the nodes are sorted from faster to slower, that is, $T_1 \leq \cdots \leq T_N$ with node 1 and

N respectively identifying the fastest and slowest nodes in the network.

3.3 Adaptive Node Participation Approach

Several federated learning algorithms have been proposed to solve the ERM problem in (3.2) such as `FedAvg` [51], `FedProx` [57], `SCAFFOLD` [8], `DIANA` [81], and `FedGATE` [56]. These methods consist of several rounds of local computations by the clients and communication with the server. Alas, in all such approaches, *all the available nodes* in the network –regardless of their computational capabilities– contribute to model learning throughout the entire procedure. As explained earlier, federated clients operate in a wide range of computational characteristics, and therefore, the server has to wait for the slowest node in each communication round to complete its local computation task. All in all, the slowest nodes determine the overall runtime of such federated algorithms which causes significant slow-down.

In this section, we describe our proposed approach to mitigate stragglers in federated learning, and lay out the intuition behind that. Our proposal, `FLANP`, is essentially a meta-algorithm that can be specified with the choice of any particular federated learning subroutine. The rest of the section focuses on a particular case of `FLANP` where the federated learning subroutine is picked to be `FedGATE` proposed by [56].

3.3.1 `FLANP`: A Straggler-Resilient FL Meta-Algorithm

Our proposal to address the device heterogeneity and mitigate the stragglers is as follows.The server first solves the ERM problem corresponding to n_0 fastest nodes, where n_0 is much smaller than the total number of available nodes N. To identify the n_0 fastest nodes, the server first broadcasts a short hand-shake message to all nodes and waits for the first n_0 nodes that respond. These n_0 nodes will participate in the training process

Algorithm 3.1: `FLANP`

```
Initialize fast-to-slow nodes $\{1, \cdots, N\}$, $n = n_0$ participating nodes with initial
  global model $\mathbf{w}_{n_0}$
while $n \leq N$ do
    while $L_n(\mathbf{w}_n) - L_n(\mathbf{w}_n^*) > V_{ns}$ do
        nodes $\{1, \cdots, n\}$ are participating and update local models via
          Federated_Solver
        server aggregates local models from nodes $\{1, \cdots, n\}$ and updates global
          model $\mathbf{w}_n$
    end
    $n \leftarrow \min\{2n, N\}$                    % doubling the participants
end
```

in the first stage. Using `Federated_Solver` which is a federated learning subroutine of choice, e.g., `FedAvg` or `FedGATE`, the n_0 participating nodes proceed to minimize the empirical risk corresponding to their data points, which we denote by $L_{n_0}(\mathbf{w})$ as defined in (3.1). This continues until the n_0 nodes reach their corresponding statistical accuracy, that is, they reach a global model $\mathbf{w}_{n_0}$ such that $L_{n_0}(\mathbf{w}_{n_0}) - L_{n_0}(\mathbf{w}_{n_0}^*) \leq V_{n_0 s}$. Note that at this stage the server has to wait only for the slowest client among the participating ones, i.e., node n_0, which is potentially much faster than the network's slowest node N.

Per our discussion in Section 3.2, a more accurate solution than $\mathbf{w}_{n_0}$ would not help improving the optimality gap. Therefore, once statistical accuracy is achieved, the procedure is terminated and we increase the number of participating nodes from n_0 to $2n_0$. To select the $2n_0$ fastest nodes, we repeat the hand-shaking communication protocol that we discussed. Then, the selected nodes use `Federated_Solver` to find the minimizer of the loss corresponding to $2n_0$ participating nodes, while using the solution of the previous stage $\mathbf{w}_{n_0}$ as their starting point. Note that since the samples of nodes come from the same distribution, we can show that the solutions of two successive stages with n_0 and $2n_0$ participants are close to each other, as we discuss in Section 3.4.

Again, in this stage, the training process terminates when we find a point $\mathbf{w}_{2n_0}$

within the statistical accuracy of the loss corresponding to $2n_0$ participating nodes, i.e., $L_{2n_0}(\mathbf{w}_{2n_0}) - L_{2n_0}(\mathbf{w}^*_{2n_0}) \leq V_{2n_0s}$. In the stage with $2n_0$ participating nodes, the computation delay is determined by the slowest participating node among $2n_0$ nodes, which is slower than the previous stage with n_0 participating nodes, but still faster than the slowest node of the network. The procedure of geometrically increasing the number of participating nodes continues till the set of participating nodes contains all the available N nodes, and these nodes find the final global model $\mathbf{w}_N$ within the statistical accuracy of the global loss function $L_N(\mathbf{w})$. Algorithm 3.1 summarizes the straggler-resilient meta-algorithm.

Remark 3.1 *In our proposed scheme, clients' computation speeds are not needed, and the parameter server figures out the fastest n nodes only by following the presented handshaking protocol.*

From a high-level perspective, Algorithm 3.1 exploits faster nodes in the beginning of the learning procedure to promptly reach a global model withing their statistical accuracy. By doing so, the server avoids waiting for slower nodes to complete their local updates; however, the optimality gap of such models are relatively large since only a fraction of data samples have contributed in the global model. By gradually increasing the number of participating nodes and activating slower nodes, the quality of the global model improves while the synchronous computation slows down due to slower nodes. The key point is that slower nodes join the learning process towards the end.

The criterion in Algorithm 3.1, that is $L_n(\mathbf{w}_n) - L_n(\mathbf{w}^*_n) > V_{ns}$, verifies that the current global model satisfies the statistical accuracy corresponding to n participating nodes $\{1, \cdots, n\}$. This condition, however, is not easy to check since the optimal solution $\mathbf{w}^*_n$ is unknown. A sufficient and computationally feasible criterion is to check if $\|\nabla L_n(\mathbf{w}_n)\|^2 \leq 2\mu V_{ns}$, when ℓ is μ-strongly convex.

3.3.2 FLANP via FedGATE

As FLANP in Algorithm 3.1 is a general mechanism to mitigate stragglers in federated settings, one needs to specify the inner optimization subroutine Federated_Solver to quantify the speedup of the proposed approach. This subroutine could be any federated learning algorithm, but here we focus on FedGATE [56], a federated learning algorithm that employs gradient tracking variables to provide tight convergence guarantees for nodes with heterogeneous data distributions.

Why FedGATE? We would like to reiterate that FLANP is a meta-procedure that can be used for *any* federated learning solver other than FedGATE to make it resilient against straggling nodes. Nevertheless, we use FedGATE as the subroutine since it can handle the case that local gradients are not an unbiased estimator of the global loss gradient, which is the case in our setting. Algorithm 3.2 demonstrates how adaptive node participation in FLANP is adopted to mitigate straggler delays in FedGATE.

We begin the first stage of Algorithm 3.2 with activating the $n = n_0$ fastest nodes $\{1, \cdots, n_0\}$ and initialize them with global model $\mathbf{w}_{n_0}$. We also reset the gradient tracking variables $\delta_i^{(0)}$ to be zero for all participating nodes at the beginning of each stage. Variables δ_i aim to correct the directions of local updates at node i by tracking the difference of local gradients $\widetilde{\nabla} L^i$ and global gradients ∇L_n such that directions d_i closely follow the correct global gradient direction. After τ_n iterations of local updates at any participating node in round r, accumulations of local gradients $\Delta_i^{(r)}$ are uploaded to the server where it updates the global model $\mathbf{w}_n$ using two stepsizes η_n, γ_n. Note that the stepsizes η_n, γ_n are fixed throughout each stage with n participating nodes but vary for different stages as n increases. After updating the global model $\mathbf{w}_n$ at the end of each round, participating nodes upload their local gradients $\nabla L^i(\mathbf{w}_n)$ such that the server aggregates and computes the global gradient $\nabla L_n(\mathbf{w}_n)$ and checks whether $\|\nabla L_n(\mathbf{w}_n)\|^2 \leq 2\mu V_{ns}$. After R_n

Algorithm 3.2: `FLANP` via `FedGATE`

```
Initialize $n = n_0$ participating nodes, initial model $\mathbf{w}_{n_0}$, initial gradient tracking
  $\delta_i^{(0)} = 0$ for participating nodes $i \in \{1, \cdots, n_0\}$
while $n \le N$ do
    $r = 0$                                          % reset round counter for each stage
    for participating nodes $i \in \{1, \cdots, n\}$ do
        $\delta_i^{(0)} = 0$                         % reset gradient tracking
    end
    while $\|\nabla L_n(\mathbf{w}_n)\|^2 > 2\mu V_{ns}$ do
        for participating nodes $i \in \{1, \cdots, n\}$ do
            $\mathbf{w}_i^{(0,r)} = \mathbf{w}_n$
            for $c = 0, \cdots, \tau_n - 1$ do
                set $d_i^{(c,r)} = \widetilde{\nabla} L^i(\mathbf{w}_i^{(c,r)}) - \delta_i^{(r)}$
                update $\mathbf{w}_i^{(c+1,r)} = \mathbf{w}_i^{(c,r)} - \eta_n d_i^{(c,r)}$
            end
            send $\Delta_i^{(r)} = (\mathbf{w}_n - \mathbf{w}_i^{(\tau_n,r)})/\eta_n$ to server
            update $\delta_i^{(r+1)} = \delta_i^{(r)} + \frac{1}{\tau_n}(\Delta_i^{(r)} - \Delta^{(r)})$
        end
        server broadcasts $\Delta^{(r)} = \frac{1}{n}\sum_{i=1}^{n} \Delta_i^{(r)}$
        server broadcasts $\mathbf{w}_n \leftarrow \mathbf{w}_n - \eta_n \gamma_n \Delta^{(r)}$
        participating nodes $i \in \{1, \cdots, n\}$ upload gradients $\nabla L^i(\mathbf{w}_n)$ to server
        $r \leftarrow r + 1$
    end
    $n \leftarrow \min\{2n, N\}$                      % doubling the participants
end
```

rounds of communications, this condition is satisfied and the set of n participating nodes reach the model $\mathbf{w}_n$ within their statistical accuracy. Therefore, we augment the set of participating nodes (from faster to slower) from $\{1, \cdots, n\}$ to $\{1, \cdots, 2n\}$ leading to a new stage. The above procedure continues until the set of participating nodes contains all N available nodes.

3.4 Theoretical Results

In this section, we provide rigorous analysis for `FLANP` outlined in Algorithm 3.2, which employs `FedGATE` as its subroutine. We first characterize optimization guarantees of Algorithm 3.2. Using such results, we derive the expected runtime of our proposed algorithm and the speedup gain it provides compared to naive methods.

3.4.1 Optimization Guarantees

Next, we analyze `FLANP` outlined in Algorithm 3.2, which employs `FedGATE` as its subroutine. We first characterize optimization guarantees of Algorithm 3.2. Using such results, we derive the expected runtime of our proposed algorithm and the speedup it provides compared to naive methods.

Connection between two successive stages. As we discussed in Section 3.3.1, we expect the solution of each stage with m participating nodes to be close to the solution of the next stage with n nodes, where $n > m$, if the larger set of nodes contain the smaller set. This is due to the fact that within each cluster, samples are drawn from the same distribution. To formalize this claim, consider a subset of m participating nodes and a model $\mathbf{w}_m^*$ within their statistical accuracy, i.e., $L_m(\mathbf{w}_m) - L_m(\mathbf{w}_m^*) \leq V_{ms}$. Next, we show that the suboptimality error of $\mathbf{w}_m$ for the next loss with n nodes is small, when the set of n nodes contains m nodes.

Proposition 3.1 *Consider two subsets of nodes $\mathcal{N}_m \subseteq \mathcal{N}_n$ and assume that model $\mathbf{w}_m$ attains the statistical accuracy for the empirical risk associated with nodes in $\mathcal{N}_m$, i.e., $\|\nabla L_m(\mathbf{w}_m)\|^2 \leq 2\mu V_{ms}$ where the loss function ℓ is μ-strongly convex. Then the suboptimality of $\mathbf{w}_m$ for risk L_n is w.h.p. bounded above by $L_n(\mathbf{w}_m) - L_n(\mathbf{w}_n^*) \leq \frac{2(n-m)}{n}(V_{(n-m)s} + V_{ms}) + V_{ms}$.*

Proof: We defer the proof to Appendix B.1. ■

Proposition 3.1 demonstrates that a model attaining the statistical accuracy for m nodes can be used as an initial model for the ERM corresponding to a larger set with n nodes. In particular, when the number of participating nodes is doubled, i.e., $n = 2m$, then the initial sub-optimality error is bounded above by $L_n(\mathbf{w}_m) - L_n(\mathbf{w}_n^*) \leq 3V_{ms}$.

Next, we characterize the required communication and computation for solving each subproblem. Specifically, consider the case that we are given a model $\mathbf{w}_m$ which is within the statistical accuracy of L_m corresponding to m fastest nodes in each cluster, and the goal is to find a new model $\mathbf{w}_n$ that is within the statistical accuracy of L_n corresponding to n fastest nodes of each cluster, where $n = 2m$. To analyze this procedure, we must specify three parameters: the choice of stepsizes η_n, γ_n, the number of local updates τ_n at each participating node, and the number of communication rounds with the server R_n. For these parameters we use index n, as they refer to the case that n nodes participate in the training. Next, we state our main assumptions.

Assumption 3.1 *The loss $\ell(\mathbf{w}, z)$ is μ-strongly convex with respect to $\mathbf{w}$, and the gradient $\nabla_{\mathbf{w}}\ell(\mathbf{w}, z)$ is L-Lipschitz continuous. The condition number is defined as $\kappa := L/\mu$.*

The conditions in Assumption 3.1 imply that the empirical risks $L_n(\mathbf{w})$ and local loss functions $L^i(\mathbf{w})$ are μ-strongly convex and have L-Lipschitz gradients. As we discussed in Section 3.2, the gap between the expected and the empirical risks corresponding to ns data samples can be bounded as $|L_n(\mathbf{w}) - L(\mathbf{w})| \leq V_{ns}$, with high probability. Next, we formalize this assumption.

Assumption 3.2 *The approximation error for the expected loss $L(\mathbf{w})$ using ns samples of n nodes in the empirical risk $L_n(\mathbf{w})$ is w.h.p. upper-bounded as $\sup_{\mathbf{w}} |L_n(\mathbf{w}) - L(\mathbf{w})| \leq V_{ns}$, where $V_{ns} = \mathcal{O}(1/ns)$. Moreover, we assume that the approximation error for gradients is upper-bounded by $\sup_{\mathbf{w}} \left\|\nabla L_n(\mathbf{w}) - \nabla L(\mathbf{w})\right\| \leq \sqrt{V_{ns}}$, w.h.p.*

We now turn our focus to the proposed Algorithm 3.2.

Theorem 3.1 *Consider the federated ERM problem in (3.2) and suppose Assumptions 3.1 and 3.2 hold. Let the proposed* `FLANP` *in Algorithm 3.2 be initialized with the fastest n_0 nodes in $\{1, \cdots, n_0\}$ and the model $\mathbf{w}_{n_0}$. Moreover, suppose the variance of stochastic local gradients is bounded above by σ^2, i.e., $\mathbb{E}[\|\widetilde{\nabla} L^i(\mathbf{w}) - \nabla L^i(\mathbf{w})\|^2] \leq \sigma^2$ for all nodes i. At any stage of Algorithm 3.2 with n participating nodes, if for sufficiently small α_n the stepsizes are $\eta_n = \frac{\alpha_n}{\tau_n \sqrt{n}}, \gamma_n = \frac{\sqrt{n}}{2\alpha_n L}$, and each node runs $\tau_n = 1.5s\sigma^2/c$ local updates, where c captures the constant term in the statistical accuracy $V_{ns} = \frac{c}{ns}$, then nodes reach the statistical accuracy of L_n after $R_n = 12\kappa \ln(6)$ rounds of communication (precise characterization of α_n in the proof).*

Proof: We defer the proof to Appendix B.2. ■

The result in Theorem 3.1 guarantees that if we initialize Algorithm 3.2 with n_0 fastest nodes and in each stage the participating nodes update their local models according to Algorithm 3.2 for $\tau = \mathcal{O}(s)$ iterations and $R = \mathcal{O}(\kappa)$ rounds, before doubling the number of participating nodes, then at the end of the final stage in which all N nodes are participating, we reach a model $\mathbf{w}_N$ that attains the statistical accuracy of the empirical risk $L_N(\mathbf{w})$. Specifically, we have $\mathbb{E}[L_N(\mathbf{w}_N) - L_N(\mathbf{w}_N^*)] \leq V_{Ns}$. Note that to obtain the best guarantee, τ_n and R_n are independent of number of participating nodes n, while the stepsizes η_n and γ_n change as the number of participating nodes increases.

3.4.2 Wall-Clock Time Analysis

We have thus far established the convergence properties of Algorithm 3.2. It is, however, equally important to show that it provably mitigates stragglers in a federated learning framework and hence speeds up the overall wall-clock time. In the following, we first characterize the running time of Algorithm 3.2 and then compare it with the one for straggler-prone `FedGATE` benchmarks.

Let $T_1 \leq \cdots \leq T_N$ denote the computation times of the N available nodes. We also denote by $\bar{T}_{\texttt{FLANP}}$ the average runtime of `FLANP` in Algorithm 3.2 to reach the overall statistical accuracy of the ERM problem corresponding to all nodes defined in (3.2). As discussed before, at the stage of `FLANP` with n participating nodes, the slowest node determines the computation time of that stage. More precisely, the computation time of each iteration of `FLANP` with n participating node per cluster is $T_n = \max\{T_1, \cdots, T_n\}$. Since each stage consists of R communication rounds each with τ local updates, the average run-time of each stage is $R\tau T_n$. Therefore, the overall wall-clock time of Algorithm 3.2 is on average $\bar{T}_{\texttt{FLANP}} = R\tau(T_{n_0} + T_{2n_0} + \cdots + T_N)$ with $R = 12\kappa \ln(6)$ and $\tau = 1.5s\sigma^2/c$ as characterized in Theorem 3.1.

This further demonstrates how the adaptive node participation approach incorporates faster nodes in order to save in the overall wall-clock time. As Theorem 3.1 shows, it suffices for each participating node in the straggler-resilient Algorithm 3.2 to run $R = \mathcal{O}(\kappa)$ rounds of local updates and $\tau = \mathcal{O}(s)$ iterations per round to reach the final statistical accuracy. Therefore, the overall wall-clock time of Algorithm 3.2 is order-wise $\bar{T}_{\texttt{FLANP}} = \mathcal{O}(\kappa s\sigma^2(T_{n_0} + T_{2n_0} + \cdots + T_N))$.

To quantify the speedup gain provided by our proposed method, we need to characterize the wall-clock time for the non-adaptive benchmark `FedGATE`. Note that in this benchmark, all the N available nodes participate in the training from the beginning.

Proposition 3.2 *The average runtime for the non-adaptive benchmark* `FedGATE` *to solve the federated ERM problem* (3.2) *and to reach the statistical accuracy of all the samples of the N nodes is $\bar{T}_{\texttt{FedGATE}} = \mathcal{O}(\kappa s\sigma^2 \ln(Ns) T_N)$ where T_N is the unit computation time of the slowest node.*

Proof: We defer the proof to Appendix B.3. ■

As expected, the result in Proposition 3.2 indicates that as all N nodes participate in training since the beginning of the algorithm, the overall wall-clock time depends only on the slowest node with computation time $T_N = \max\{T_1, \cdots, T_N\}$.

Thus far, we have characterized the order-wise expressions of the average wall-clock time for `FLANP` and `FedGATE` methods as follows

$$\bar{T}_{\texttt{FLANP}} = \mathcal{O}(\kappa s \sigma^2 (T_{n_0} + T_{2n_0} + \cdots + T_N)), \quad \bar{T}_{\texttt{FedGATE}} = \mathcal{O}(\kappa s \sigma^2 \ln(Ns) T_N). \tag{3.3}$$

To establish the speedup for the straggler-resilient method, we consider a random exponential time model for clients computation times, which has been widely used to capture the computation delay for distributed clusters [50, 82]. We assume that nodes computation time are independent realizations of an exponential random variable and characterize the speedup of the resilient Algorithm 3.2 compared to the benchmark `FedGATE`.

Theorem 3.2 *Suppose the clients' computation times are i.i.d. random variables drawn from an exponential distribution with parameter λ. That is, $T_1, \cdots, T_N \sim \exp(\lambda)$. Then, the speedup gain of the* `FLANP` *Algorithm 3.2 compared to the naive* `FedGATE` *method is*

$$\frac{\mathbb{E}[\bar{T}_{\texttt{FLANP}}]}{\mathbb{E}[\bar{T}_{\texttt{FedGATE}}]} \leq \mathcal{O}\left(\frac{1}{\ln(Ns)}\right).$$

Proof: We defer the proof to Appendix B.4. ■

Theorem 3.2 establishes a $\mathcal{O}(\ln(Ns))$ speedup gain for `FLANP` compared to its non-adaptive and straggler-prone benchmark `FedGATE`, when the clients' computation times are drawn from a random exponential time model.

We have so far considered device heterogeneous federated clients with potentially well-spread computation speeds and demonstrated the speedup gain obtained by adaptive node participation approach, particularly in Theorem 3.2. We would like to add that our method provides provable speedups even for device *homogeneous* clients with identical speeds, i.e., $T_1 = \cdots = T_N$. Comparing the expected runtimes in (3.3) yields that `FLANP` in Algorithm 3.2 slashes the expected wall-clock time of `FedGATE` by a factor $\ln(Ns)/\ln(N)$. This observation demonstrates that the adaptive node participation approach results in two different speedup gains: (i) leveraging faster nodes to speedup the learning and (ii) adaptively increase the effective sample size by participating more clients.

3.5 Numerical Experiments

We conduct various numerical experiments for convex and nonconvex risks and evaluate the performance of the proposed method versus other benchmarks.

Benchmarks. Bellow is a brief description for multiple federated learning benchmarks that we use to compare with the proposed `FLANP` in Algorithm 3.2. Note that in all these benchmarks all the available N nodes participate in the training process.

• `FedAvg` [51]. Nodes update their local model using a simple SGD rule for τ local iterations before uploading to the server.

• `FedGATE` [56]. This is the subroutine used in Algorithm 3.2. Here we consider it as a benchmark running with all the available N nodes with model update rule similar to the subroutine in Algorithm 3.2.

• `FedNova` [55]. In each round, each node i updates its local model for τ_i iterations where τ_is vary across the nodes. To mitigate the heterogeneity in τ_is, the server aggregates normalized updates (w.r.t. τ_i) from the clients and updates the global model.

We compare the performance of `FLANP` with such benchmarks in terms of communication rounds and wall-clock time. We examine `FLANP` against the benchmarks under both

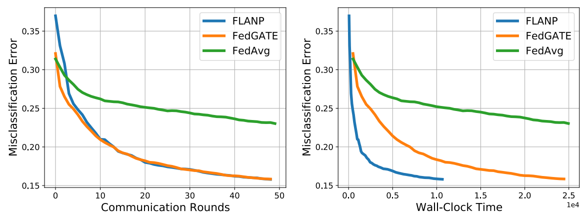


Figure 3.1: Logistic Regression over MNIST

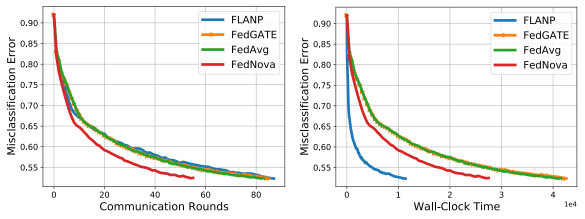


Figure 3.2: NN on CIFAR10

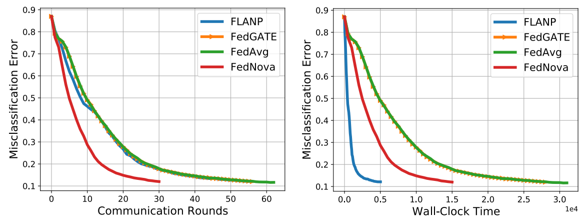


Figure 3.3: NN on MNIST

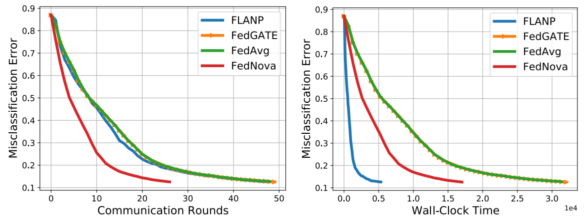


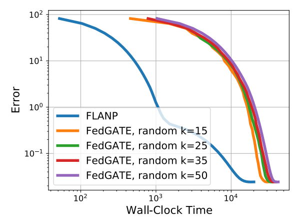
Figure 3.4: NN on MNIST

full and partial node participation scenarios and highlight its practicality. We consider computation speeds that are uniformly distributed and exponentially distributed.

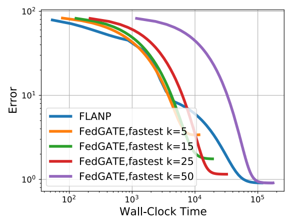
Uniform computation speeds.

Data and Network. We use three main datasets for different problems: MNIST (60,000 training, 10,000 test samples), CIFAR10 (50,000 training, 10,000 test samples) and synthetic (10,000 samples) datasets. To implement our algorithm, we employ a federated network of $N \in \{20, 50, 100\}$ heterogeneous clients and in order to model the device heterogeneity, we realize and then fix the computation speed of each node i, i.e. T_i from the interval $[50, 500]$ uniformly at random.

Logistic Regression. We use the MNIST dataset to learn a multi-class logistic regression model. In a network of $N = 50$ nodes, each client stores $s = 1200$ samples from the MNIST dataset. As demonstrated in Figure 3.1 (left), `FLANP` is slightly outperformed by `FedGATE` at the initial rounds. This is however expected as `FLANP` starts with only a fraction of nodes participating which leads to less accurate models. With respect to wall-clock time however, `FLANP` outperforms both `FedAvg` and `FedGATE` benchmarks due to the fact that the initial participating nodes are indeed the fastest ones. As Figure 3.1

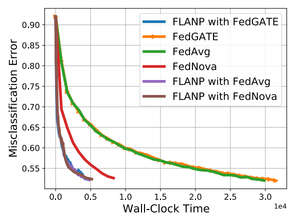


(a) k randomly picked.

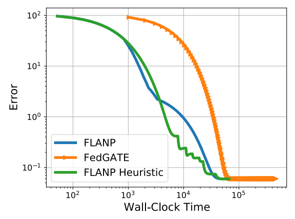


(b) k fastest picked.

Figure 3.5: Partial node participation



(a) Other FL solvers.



(b) Heuristics `FLANP`.

Figure 3.6: Heuristics `FLANP` other solvers.

(right) shows, the adaptive node participation approach leads `FLANP` to speedup gains of up to 2.1× compared to `FedGATE`.

Neural Network Training. We train a fully connected neural network with two hidden layers with 128 and 64 neurons and compare with three other benchmarks including `FedNova` which is stragglers-resilient. We conduct two sets of experiments over CIFAR10 and MNIST on a network of $N = 20$ clients, as demonstratd in Figures 3.2 and 3.3 where `FLANP` accelerates the training by up to 3× compared to `FedNova`.

Random exponential computation speeds. We conduct another set of experiments using the same setup described earlier. However, we here pick the clients' computation speed to be i.i.d. random exponential variables, i.e. consistent with Theorem 3.2. We train a fully connected neural network with two hidden layers with 128 and 64 neurons on MNIST and compare with benchmarks `FedAvg`, `FedGATE` and `FedNova` as demonstrated in Figure 3.4.

Comparison with partial node participation methods. Thus far, we have compared the `FLANP` method with federated benchmarks in which *all* of the available nodes participate in training in every round. To demonstrate the resiliency of `FLANP` to partial node participation methods, we consider two different scenarios. First, we compare the wall-clock time of a neural network training of `FLANP` with partial node participation `FedGATE` in which only k out of $N = 50$ nodes are randomly picked and participate in each round. As demonstrated in Figure 3.5(a), `FLANP` is significantly faster than FedGATE with partial node participation. Second, we consider the case that the k participating nodes are not randomly picked, but are the fastest clients. As shown in Figure 3.5(b), although partial participation methods with k fastest nodes begin to outperform FLANP, towards the end of the training, they suffer from higher training error saturation as the data samples of *only* k nodes contribute in the learned model and hence the final model is significantly inaccurate.

FLANP with other federated solvers. To illustrate the compatibility of FLANP with solvers other than FedGATE, we train the neural network on MNIST and employ FedAvg and FedNova as solvers of FLANP. As shown in Figure 3.6(a), FLANP is able to significantly speedup all three solvers.

Lastly, we note that from the practical point of view, there are several heuristic approaches to estimate the constant parameters μ, c, V_{ns} in Algorithm 3.2. We conducted an experiment to learn a linear regression model with Gaussian synthetic data in which none of the constants are assumed to be known. Rather, we heuristically tune the threshold for each stage transition (i.e. doubling the nodes) by monitoring the norm of the global gradient and successively halving the threshold. As shown in Figure 3.6(b), the performance of such heuristic methods is indeed close to FLANP which highlights its practicality.

3.6 Concluding Remarks

In this chapter, we targeted straggler and system heterogeneity challenge in federated learning frameworks and proposed an adaptive node participation scheme to mitigate slow nodes during the training, namely FLANP. In our proposal, the training begins with only a handful of devices which are the fastest among the total N available nodes in the network. After the trained model on such devices reaches their corresponding statistical accuracy, `FLANP` doubles the number of participating nodes. We rigorously discussed how such doubling procedure enables the trained model at the end of each stage to be a proper warm-up initial model for the next stage. Doubling the participants continues till all the N nodes are incorporated in the training. For strongly convex objectives, we characterized the convergence guarantees of `FLANP` when combined with `FedGATE` as the inner federated learning solver. We also established order-wise speedup gain of the proposed adaptive methods compared to its non-adaptive counter method. Our numerical experiments also demonstrate significant speedups in different convex and non-convex scenarios, where we highlighted the practicality of the proposed method as well.

Chapter 4

Distributionally-Robust Federated Learning

In federated learning settings, the training data is often statistically heterogeneous and manifests various distribution shifts across users, which degrades the performance of the learnt model. The primary goal of this chapter is to develop a robust federated learning algorithm that achieves satisfactory performance against distribution shifts in users' samples. To achieve this goal, we first consider a *structured* affine distribution shift in users' data that captures the device-dependent data heterogeneity in federated settings. This perturbation model is applicable to various federated learning problems such as image classification where the images undergo device-dependent imperfections, e.g. different intensity, contrast, and brightness. To address affine distribution shifts across users, we propose a **F**ederated **L**earning framework **R**obust to **A**ffine distribution shifts (`FLRA`) that is robust against affine distribution shifts to the distribution of observed samples. To solve the `FLRA`'s distributed minimax optimization problem, we propose a fast and efficient optimization method and provide convergence and performance guarantees via a gradient Descent Ascent (GDA) method. We further prove generalization error bounds for the learnt classifier to show proper generalization from empirical distribution of samples to the true underlying distribution. We perform several numerical experiments to empirically support `FLRA`. We show that an affine distribution shift indeed suffices to significantly decrease the performance of the learnt classifier in a new test user, and our proposed algorithm achieves a significant gain in comparison to standard federated learning and adversarial training methods.

4.1 Introduction

A typical federated learning setting consists of a network of hundreds to millions of devices (nodes) which interact with each other through a a parameter server. Communicating messages over such a large-scale network can lead to major slow-downs due to communication bandwidth bottlenecks [3, 25]. In fact, the communication bottleneck is one of the main grounds that distinguishes federated and standard distributed learning paradigms. To reduce communication load in federated learning, one needs to depart from the classical setting of distributed learning in which updated local models are communicated to the central server *at each iteration*, and communicate less frequently.

Another major challenge in federated learning is the statistical heterogeneity of training data [3, 25]. As mentioned above, a federated setting involves many devices, each generating or storing personal data such as images, text messages or emails. Each user's data samples can have a (slightly) different underlying distribution which is another key distinction between federated learning and classical learning problems. Indeed, it has been shown that standard federated methods such as `FedAvg` [30] which are designed for i.i.d. data significantly suffer in statistical accuracy or even diverge if deployed over non-i.i.d. samples [8]. Device-dependency of local data along with privacy concerns in federated tasks does not allow learning the distribution of individual users and necessitates novel algorithmic approaches to learn a classifier robust to distribution shifts across users. Specifically, statistical heterogeneity of training samples in federated learning can be problematic for generalizing to the distribution of a test node unseen in training time. We show through various numerical experiments that even a simple linear filter applied to the test samples will suffice to significantly degrade the performance of a model learned by `FedAvg` in standard image recognition tasks.

To address the aforementioned challenges, we propose a new federated learning scheme called `FLRA`, a **F**ederated **L**earning framework with **R**obustness to **A**ffine distribution shifts. FLRA has a small communication overhead and a low computation complexity. The key insight in FLRA is model the heterogeneity of training data in a device-dependent manner, according to which the samples stored on the ith device $\mathbf{x}^i$ are shifted from a ground distribution by an affine transformation $\mathbf{x}^i \to \Lambda^i \mathbf{x}^i + \delta^i$. To further illustrate this point, consider a federated image classification task where each mobile device maintains a collection of images. The images taken by a camera are similarly distorted depending on the intensity, contrast, blurring, brightness and other characteristics of the camera [9,10], while these features vary across cameras. In addition to camera imperfections, such unseen distributional shifts also originate from changes in the physical environment, e.g. weather conditions [11]. Compared to the existing literature, our model provides more robustness compared to the well-known adversarial training models $\mathbf{x}^i \to \mathbf{x}^i + \delta^i$ with solely additive perturbations [83–85], i.e. $\Lambda^i = I$. Our perturbation model also generalizes the universal adversarial training approach in which *all* the training samples are distorted with an identical perturbation $\mathbf{x}^i \to \mathbf{x}^i + \delta$ [86].

Based on the above model, `FLRA` formulates the robust learning task as a minimax robust optimization problem, which finds a *global* model $\mathbf{w}^*$ that minimizes the total loss induced by the worst-case *local* affine transformations $(\Lambda^{i*}, \delta^{i*})$. One approach to solve this minimax problem is to employ techniques from adversarial training in which for each iteration and a given global model $\mathbf{w}$, each node optimizes its own local adversarial parameters (Λ^i, δ^i) and a new model is obtained. This approach is however undesirable in federated settings since it requires extensive computation resources at each device as they need to fully solve the adversarial optimization problem at each iteration. To tackle this challenge, one may propose to use standard distributed learning frameworks in which each node updates its local adversarial parameters and shares with the server at *each iteration* of the distributed algorithm to obtain the updated global model. This is also in contrast with the availability of limited communication resources in federated settings. The key contribution of our work is to develop a novel method called `FedRobust`, which is a gradient descent ascent (GDA) algorithm to solve the minimax robust optimization problem, can be efficiently implemented in a federated setting, and comes with strong theoretical guarantees. While the `FLRA` minimax problem is in general non-convex non-concave, we show that `FedRobust` which alternates between the perturbation and parameter model variables will converge to a stationary point in the minimax objective that satisfies the Polyak-Łojasiewicz (PL) condition. Our optimization guarantees can also be extended to more general classes of non-convex non-concave distributed minimax optimization problems.

As another major contribution of the chapter, we use the PAC-Bayes framework [87, 88] to prove a generalization error bound for `FLRA`'s learnt classifier. Our generalization bound applies to multi-layer neural network classifiers and is based on the classifier's Lipschitzness and smoothness coefficients. The generalization bound together with our optimization guarantees suggest controlling the neural network classifier's complexity through Lipschitz regularization methods. Regarding `FLRA`'s robustness properties, we connect the minimax problem in `FLRA` to a distributionally robust optimization problem [89, 90] where we use an optimal transport cost to measure the distance between distributions. This connection reveals that the `FLRA`'s minimax objective provides a lower-bound for the objective of a distributionally robust problem. Finally, we discuss the results of several numerical experiments to empirically support the proposed robust federated learning method. Our experiments suggest a significant gain under affine distribution shifts compared to existing adversarial training algorithms. In addition, we show that the trained classifier performs robustly against standard FGSM and PGD adversarial attacks, and outperforms `FedAvg`.

Related work. As a practical on-device learning paradigm, federated learning has recently gained significant attention in machine learning and optimization communities. Since the introduction of `FedAvg` [30] as a communication-efficient federated learning method, many works have developed federated methods under different settings with optimization guarantees for a variety of loss functions [70, 91]. Moreover, another line of work has tackled the communication bottleneck in federated learning via compression and sparsification methods [7,92,93]. [94–97] have focused on designing privacy-preserving federated learning schemes. There have also been several recent works the study local-SGD methods as a subroutine of federated algorithms and provide various convergence results depending on the loss function class [38, 40, 74]. Making federated learning methods robust to non-i.i.d. data has also been the focus of several works [8, 58, 65].

Adversarially robust learning paradigms usually involve solving a minimax problem of the form $\min_{\mathbf{w}} \max_{\boldsymbol{\psi}} f(\mathbf{w}, \boldsymbol{\psi})$. As the theory of adversarially robust learning surges, there has been thriving recent interests in solving the minimax problem for nonconvex cases. Most recently, [18] provides nonasymptotic analysis for nonconvex-concave settings and shows that the iterates of a simple Gradient Descent Ascent (GDA) efficiently find the stationary points of the function $\Phi(\mathbf{w}) := \max_{\boldsymbol{\psi}} f(\mathbf{w}, \boldsymbol{\psi})$. [98] establishes convergence results for the nonconvex-nonconcave setting and under PL condition. This problem has been studied in the context of game theory as well [99].

4.2 Federated Learning Scenario

Consider a federated learning setting with a network of n nodes (devices) connected to a server node. We assume that for every $1 \leq i \leq n$ the ith node has access to m training samples in $S^i = \{(\mathbf{x}^i_j, y^i_j) \in \mathbb{R}^d \times \mathbb{R} : 1 \leq j \leq m\}$. For a given loss function ℓ and function class $\mathcal{F} = \{f_{\mathbf{w}} : \mathbf{w} \in \mathcal{W}\}$, the classical federated learning problem is to fit the best model $\mathbf{w}$ to the nm samples via solving the following empirical risk minimization (ERM) problem:

$$\min_{\mathbf{w} \in \mathcal{W}} \; \frac{1}{nm} \sum_{i=1}^{n} \sum_{j=1}^{m} \ell \left(f_{\mathbf{w}}(\mathbf{x}^i_j), y^i_j \right).$$

As we discussed previously, the training data is statistically heterogeneous across the devices. To capture the non-identically-distributed nature of data in federated learning, we assume that the data points of each node have a local distribution shift from a common distribution. To be more precise, we assume that each sample stored in node i in S^i is distributed according to an affine transformation h^i of a universal underlying distribution $P_{\mathbf{X},Y}$, i.e., transforming the features of a sample $(\mathbf{x}, y) \sim P_{\mathbf{X},Y}$ according to the following affine function $h^i(\mathbf{x}) := \Lambda^i \mathbf{x} + \delta^i$. Here $\Lambda^i \in \mathbb{R}^{d \times d}$ and $\delta^i \in \mathbb{R}^d$, with d being the dimension of input variable $\mathbf{x}$, characterize the affine transformation h^i at node i. According to this model, all samples stored at node i are affected with the same affine transformation while other nodes $j \neq i$ may experience different transformations.

This *structured* model particularly supports the data heterogeneity in federated settings. That is, the data generated and stored in each federated device is exposed to identical yet device-dependent distortions while different devices undergo different distortions. As an applicable example that manifests the proposed perturbation model, consider a federated image classification task over the images taken and maintained by mobile phone devices. Depending on the environment's physical conditions and the camera's imperfections, the pictures taken by a particular camera undergo device-dependent perturbations. According to the proposed model, such distribution shift is captured as an affine transformation $h^i(\mathbf{x}) = \Lambda^i\mathbf{x} + \delta^i$ on the samples maintained by node i. To control the perturbation power, we consider bounded Frobenius and Euclidean norms $\|\Lambda - I_d\|_F \leq \epsilon_1$ and $\|\delta\|_2 \leq \epsilon_2$ enforcing the affine transformation to have a bounded distance from the identity transformation.

Based on the model described above, our goal is to solve the following distributionally robust federated learning problem:

$$\min_{\mathbf{w}\in\mathcal{W}} \; \frac{1}{n}\sum_{i=1}^{n} \max_{\substack{\|\Lambda^i - I\|_F \leq \epsilon_1 \\ \|\delta^i\| \leq \epsilon_2}} \; \frac{1}{m}\sum_{j=1}^{m} \ell\left(f_{\mathbf{w}}(\Lambda^i\mathbf{x}_j^i + \delta^i), y_j^i\right). \tag{4.1}$$

The minimax problem (4.1) can be interpreted as $n+1$ coupled optimization problems. First, in n inner local maximization problems and for a given global model $\mathbf{w}$, each node $1 \leq i \leq n$ seeks a (feasible) affine transformation (Λ^i, δ^i) which results in high losses via solving $\max_{\Lambda^i,\delta^i} \frac{1}{m}\sum_{j=1}^{m} \ell(f_{\mathbf{w}}(\Lambda^i\mathbf{x}_j^i + \delta^i), y_j^i)$ over its m training samples in S^i. Then, the outer minimization problem finds a global model yielding the smallest value of cumulative losses over the n nodes.

Solving the above minimax problem requires collaboration of distributed nodes via the central server. In federated learning paradigms however, such nodes are entitled to limited computation and communication resources. Such challenges particularly prevent us from employing the standard techniques in adversarial training and distributed ERM. More precisely, each iteration of adversarial training requires solving a maximization problem at each local node which incurs extensive computational cost. On the other hand, tackling the minimax problem (4.1) via iterations of standard distributed learning demands frequent message-passing between the nodes and central server *at each iteration*, hence yielding massive communication load on the network. To account for such system challenges, we constitute our goal to solve the robust minimax problem in (4.1) with small computation and communication cost so that it can be feasibly and efficiently implemented in a federated setting.

4.3 The Proposed `FedRobust` Algorithm

To guard against affine distribution shifts, we propose to change the original constrained maximization problem to the following worst-case loss at each node i, given a Lagrange multiplier $\lambda > 0$:

$$\max_{\Lambda^i,\delta^i} \frac{1}{m}\sum_{j=1}^{m} \ell\left(f_{\mathbf{w}}(\Lambda^i \mathbf{x}_j^i + \delta^i), y_j^i\right) - \lambda\|\Lambda^i - I\|_F^2 - \lambda\|\delta^i\|_2^2. \tag{4.2}$$

Here we use a norm-squared penalty requiring a bounded distance between the feasible affine transformations and the identity mapping, and find the worst-case affine transformation that results in the maximum loss for the samples of node i. By averaging such worst-case local losses over all the n nodes and minimizing w.r.t. model $\mathbf{w}$, we reach the following minimax optimization problem:

$$\min_{\mathbf{w}\in\mathcal{W}} \max_{(\Lambda^i,\delta^i)_{i=1}^n} \frac{1}{nm}\sum_{i=1}^{n}\sum_{j=1}^{m} \ell\left(f_{\mathbf{w}}(\Lambda^i \mathbf{x}_j^i + \delta^i), y_j^i\right) - \lambda\|\Lambda^i - I\|_F^2 - \lambda\|\delta^i\|_2^2. \tag{4.3}$$

This formalizes our approach to tackling the robust federated learning problem, which we call **F**ederated **L**earning framework **R**obust to **A**ffine distribution shift or `FLRA` in short.

In order to solve `FLRA` in (4.3), we propose a gradient optimization method that is computationally and communication-wise efficient, called `FedRobust`. The proposed

Algorithm 4.1: `FedRobust`

```
Input: $\{\mathbf{w}_0^i = \mathbf{w}_0, \Lambda_0^i, \delta_0^i\}_{i=1}^n, \eta_1, \eta_2, \tau, T$
for each iteration $t = 0, \cdots, T-1$, node $i$ computes do
    $\Lambda_{t+1}^i = \Lambda_t^i + \eta_2 \widetilde{\nabla}_\Lambda f^i(\mathbf{w}_t^i, \Lambda_t^i, \delta_t^i)$
    $\delta_{t+1}^i = \delta_t^i + \eta_2 \widetilde{\nabla}_\delta f^i(\mathbf{w}_t^i, \Lambda_t^i, \delta_t^i)$
    if $t$ does not divide $\tau$ then
        $\mathbf{w}_{t+1}^i = \mathbf{w}_t^i - \eta_1 \widetilde{\nabla}_\mathbf{w} f^i(\mathbf{w}_t^i, \Lambda_t^i, \delta_t^i)$
    else
        node $i$ uploads to server:
            $\mathbf{w}_t^i - \eta_1 \widetilde{\nabla}_\mathbf{w} f^i(\mathbf{w}_t^i, \Lambda_t^i, \delta_t^i)$
        server sends to all nodes $i$:
            $\mathbf{w}_{t+1}^i = \frac{1}{n} \sum_{j=1}^n \Big[\mathbf{w}_t^j - \eta_1 \widetilde{\nabla}_\mathbf{w} f^j(\mathbf{w}_t^j, \Lambda_t^j, \delta_t^j)\Big]$
    end
end
Output: $\overline{\mathbf{w}}_T = \frac{1}{n}\sum_{i=1}^n \mathbf{w}_T^i$
```

`FedRobust` algorithm is an iterative scheme that applies stochastic gradient descent ascent (SGDA) updates for solving the minimax problem (4.3). As summarized in Algorithm 4.1, in each iteration t of local updates, each node i takes a (stochastic) gradient ascent step and updates its affine transformation parameters $(\Lambda_t^i, \delta_t^i)$. It also updates the local classifier's parameters $\mathbf{w}_t^i$ via a gradient descent step. After τ local iterations, local models $\mathbf{w}_t^i$ are uploaded to the server node where the global model is obtained by averaging the local ones. The averaged model is then sent back to the nodes to begin the next round of local iterations with this fresh initialization. Note that each node updates its perturbation parameters only once in each iteration which yields light computation cost as opposed to standard adversarial training methods. Moreover, periodic communication at every τ iterations, reduces the communication load compared to standard distributed optimization methods by a factor τ.

It is worth noting that the local affine transformation variables Λ^i, δ^i are coupled even though they remain on their corresponding nodes and are not exchanged with the server. This is due to the fact that the fresh model $\mathbf{w}$ is the average of the updated models from *all* the nodes; hence, updating Λ^i, δ^i for node i will affect Λ^j, δ^j for other nodes $j \neq i$ in the following iterations. This is indeed a technical challenge that arises in proving the optimization guarantees of `FedRobust` in Section 4.4.1.

4.4 Theoretical Guarantees: Optimization, Generalization and Robustness

In this section, we establish our main theoretical results. First, we characterize the convergence of `FedRobust` in Algorithm 4.1. Next, we prove that the learned hypothesis will properly generalize from training data to unseen test samples. Lastly, we demonstrate that solving the `FLRA`'s minimax problem (4.3) results in a robust classifier to Wasserstein shifts structured across the nodes.

4.4.1 Optimization guarantees

In this section, we establish our main convergence results and show that `FedRobust` finds saddle points of the minimax problem in (4.2) for two classes of loss functions. We first set a few notations as follows. We let matrix $\boldsymbol{\psi}^i = (\Lambda^i, \delta^i) \in \mathbb{R}^{d \times (d+1)}$ denote the joint transformation variables corresponding to node i. The collection of n such variables corresponding to the n nodes is denoted by the matrix $\Psi = (\boldsymbol{\psi}^1; \cdots; \boldsymbol{\psi}^n)$. We can now rewrite the minimax problem (4.3) as follows:

$$\min_{\mathbf{w}} \max_{\Psi} f(\mathbf{w}, \Psi) := \min_{\mathbf{w}} \max_{\boldsymbol{\psi}^1, \cdots, \boldsymbol{\psi}^n} \frac{1}{n} \sum_{i=1}^{n} f^i(\mathbf{w}, \boldsymbol{\psi}^i), \tag{4.4}$$

where f and f^is denote the penalized global and local losses, respectively; that is, for each node i

$$f^i(\mathbf{w}, \boldsymbol{\psi}^i) := \frac{1}{m} \sum_{j=1}^{m} \ell \left(f_{\mathbf{w}}(\Lambda^i \mathbf{x}_j^i + \delta^i), y_j^i \right) - \lambda \|\Lambda^i - I\|_F^2 - \lambda \|\delta^i\|^2. \tag{4.5}$$

We also define $\Phi(\mathbf{w}) := \max_{\Psi} f(\mathbf{w}, \Psi)$ and $\Phi^* := \min_{\mathbf{w}} \Phi(\mathbf{w})$. Next, we state a few customary assumptions on the data and loss functions. As we mentioned before, we assume that data is heterogeneous (non-iid). There are several notions to quantify the degree of heterogeneity in the data. Here, we use a notion called *non-iid degree* which is defined as the variance of the local gradients with respect to the global gradient [100].

Assumption 4.1 (Bounded non-iid degree) *We assume that when there are no perturbations, the variance of the local gradients with respect to the global gradient is bounded. That is, there exists ρ_f^2 such that for $\boldsymbol{\psi}^i = (I, 0), \Psi = (\boldsymbol{\psi}^1; \cdots; \boldsymbol{\psi}^n)$ and all $\mathbf{w}$,*

$$\frac{1}{n} \sum_{i=1}^{n} \left\| \nabla_{\boldsymbol{w}} f^i(\mathbf{w}, \boldsymbol{\psi}^i) - \nabla_{\boldsymbol{w}} f(\mathbf{w}, \Psi) \right\|^2 \leq \rho_f^2.$$

Assumption 4.2 (Stochastic gradients) *For each node i, the stochastic gradients $\widetilde{\nabla}_{\mathbf{w}} f^i$ and $\widetilde{\nabla}_{\boldsymbol{\psi}} f^i$ are unbiased and have variances bounded by $\sigma_{\mathbf{w}}^2$ and $\sigma_{\boldsymbol{\psi}}^2$, respectively. That is, for all $\mathbf{w}, \boldsymbol{\psi}$,*

$$\mathbb{E} \left\| \widetilde{\nabla}_{\mathbf{w}} f^i(\mathbf{w}, \boldsymbol{\psi}) - \nabla_{\boldsymbol{w}} f^i(\mathbf{w}, \boldsymbol{\psi}) \right\|^2 \leq \sigma_{\mathbf{w}}^2, \quad \mathbb{E} \left\| \widetilde{\nabla}_{\boldsymbol{\psi}} f^i(\mathbf{w}, \boldsymbol{\psi}) - \nabla_{\boldsymbol{\psi}} f^i(\mathbf{w}, \boldsymbol{\psi}) \right\|^2 \leq \sigma_{\boldsymbol{\psi}}^2.$$

Assumption 4.3 (Lipschitz gradients) *All local loss functions have Lipschitz gradi-*

ents. That is, for any node i, there exist constants L_1, L_2, L_{12}, and L_{21} such that for any $\mathbf{w}, \mathbf{w}', \boldsymbol{\psi}, \boldsymbol{\psi}'$ *we have*

$$\begin{aligned}
\left\|\nabla_{\boldsymbol{w}} f^i(\mathbf{w}, \boldsymbol{\psi}) - \nabla_{\boldsymbol{w}} f^i(\mathbf{w}', \boldsymbol{\psi})\right\| &\leq L_1 \left\|\mathbf{w} - \mathbf{w}'\right\|, \\
\left\|\nabla_{\boldsymbol{w}} f^i(\mathbf{w}, \boldsymbol{\psi}) - \nabla_{\boldsymbol{w}} f^i(\mathbf{w}, \boldsymbol{\psi}')\right\| &\leq L_{12} \left\|\boldsymbol{\psi} - \boldsymbol{\psi}'\right\|_F, \\
\left\|\nabla_{\boldsymbol{\psi}} f^i(\mathbf{w}, \boldsymbol{\psi}) - \nabla_{\boldsymbol{\psi}} f^i(\mathbf{w}', \boldsymbol{\psi})\right\|_F &\leq L_{21} \left\|\mathbf{w} - \mathbf{w}'\right\|, \\
\left\|\nabla_{\boldsymbol{\psi}} f^i(\mathbf{w}, \boldsymbol{\psi}) - \nabla_{\boldsymbol{\psi}} f^i(\mathbf{w}, \boldsymbol{\psi}')\right\|_F &\leq L_2 \left\|\boldsymbol{\psi} - \boldsymbol{\psi}'\right\|_F.
\end{aligned}$$

We show the convergence of `FedRobust` for two classes of loss functions: PL-PL and nonconvex-PL. Next, we briefly describe these classes and state the main results. The celebrated work of Polyak [101] introduces a sufficient condition for an unconstrained minimization problem $\min_x g(x)$ under which linear convergence rates can be established using gradient methods. A function $g(x)$ satisfies the Polyak-Łojasiewicz (PL) condition if $g^* = \min_x g(x)$ exits and is bounded, and there exists a constant $\mu > 0$ such that $\|\nabla g(x)\|^2 \geq 2\mu(g(x) - g^*)$, $\forall x$. Similarly, we can define two-sided PL condition for our minimax objective function in (4.4) [98].

Assumption 4.4 (PL condition) *The global function f satisfies the two-sided PL condition, that is, there exist positive constants μ_1 and μ_2 such that*

$$\begin{aligned}
&\text{(i)}\ \frac{1}{2\mu_1}\left\|\nabla_{\boldsymbol{w}} f(\mathbf{w}, \Psi)\right\|^2 \geq f(\mathbf{w}, \Psi) - \min_{\mathbf{w}} f(\mathbf{w}, \Psi), \\
&\text{(ii)}\ \frac{1}{2\mu_2}\left\|\nabla_{\Psi} f(\mathbf{w}, \Psi)\right\|_F^2 \geq \max_{\Psi} f(\mathbf{w}, \Psi) - f(\mathbf{w}, \Psi).
\end{aligned}$$

In other words, Assumptions 4.4 states that the functions $f(\cdot, \Psi)$ and $-f(\mathbf{w}, \cdot)$ satisfy the PL condition with constants, μ_1 and μ_2, respectively. To measure the optimality gap at iteration t, we define the potential function $P_t := a_t + \beta b_t$, where $a_t := \mathbb{E}[\Phi(\overline{\mathbf{w}}_t)] - \Phi^*$

and $b_t \coloneqq \mathbb{E}[\Phi(\overline{\mathbf{w}}_t) - f(\overline{\mathbf{w}}_t, \Psi_t)]$ and β is an arbitrary and positive constant. Note that both a_t and b_t are non-negative and if P_t approaches zero, it implies that $(\overline{\mathbf{w}}_t, \Psi_t)$ is approaching a minimax point.

Theorem 4.1 (PL-PL loss) *Consider the iterates of* `FedRobust` *in Algorithm 4.1 and let Assumptions 4.1, 4.3, and 4.4 hold. Then for any iteration $t \geq 0$, the optimality gap $P_t \coloneqq a_t + \frac{1}{2} b_t$ satisfies the following:*

$$P_t \leq \left(1 - \frac{1}{2}\mu_1\eta_1\right)^t P_0 + 32\eta_1 \frac{\tilde{L}}{\mu_1}(\tau-1)^2\rho^2 + 8\eta_1\frac{\tilde{L}}{\mu_1}(\tau-1)(n+1)\frac{\sigma_{\boldsymbol{w}}^2}{n} + \eta_1\frac{\hat{L}}{\mu_1}\frac{\sigma_{\boldsymbol{w}}^2}{n} + \frac{\eta_2^2}{\eta_1}\frac{L_2}{2\mu_1}\sigma_\psi^2,$$

for maximization step-size η_2 and minimization step-size η_1 that satisfy the following conditions:

$$\eta_2 \leq \frac{1}{L_2}, \quad 32\eta_1^2(\tau-1)^2L_1^2 \leq 1, \quad \frac{\mu_2^2\eta_2 n}{\eta_1 L_1 L_2} \geq 1 + 8\frac{L_{12}^2}{L_1L_2},$$
$$\eta_1\left(\hat{L} + \frac{80\tilde{L}(\tau-1)}{\mu_1\eta_1(1-\frac{1}{2}\mu_1\eta_1)^{\tau-1}}\right) \leq 1.$$

Here, we denote $\rho^2 \coloneqq 3\rho_f^2 + 6L_{12}^2(\epsilon_1^2+\epsilon_2^2)$ where ϵ_1 and ϵ_2 specify the bounds on the affine transformations $h^i(\mathbf{x}) = \Lambda^i\mathbf{x} + \delta^i$. We also use the following notations:

$$L_\Phi = L_1 + \frac{L_{12}L_{21}}{2n\mu_2}, \quad \tilde{L} = \frac{3}{2}\eta_1L_1^2 + \frac{1}{2}\eta_2L_{21}^2, \quad \hat{L} = \frac{3}{2}L_\Phi + \frac{1}{2}L_1 + \frac{L_{21}^2}{L_2}.$$

Proof: We defer the proof to Appendix C.2. ■

Special cases of this convergence result is consistent with similar ones already established in the literature. In the particular case of (non-federated) distributed optimization, i.e. $\tau = 1$, Theorem 4.1 recovers the convergence result in [98]. Moreover, putting

$\epsilon_1, \epsilon_2 \to 0$ reduces the problem to standard (non-robust) federated learning where our result is also consistent with the prior work [70]. We also note that the conditions on the stepsizes can be interpreted as linear conditions on η_1, η_2 and is always feasible. For instance, one can pick $\eta_1 = \mathcal{O}(\ln(T)/T), \eta_2 = \mathcal{O}(\ln(T)/T)$ for running `FedRobust` for T iterations, which yields that $P_T \le \mathcal{O}(\ln(T)/T)$. Next, we relax the PL condition on $f(\cdot, \Psi)$ stated in Assumption 4.4 (i) and show that the iterates of the `FedRobust` method find a stationary point of the minimax problem (4.4) when the objective function $f(\mathbf{w}, \Psi)$ only satisfies the PL condition with respect to Ψ and is nonconvex with respect to $\mathbf{w}$.

Theorem 4.2 (Nonconvex-PL loss) *Consider the iterates of* `FedRobust` *in Algorithm 4.1 and let Assumptions 4.1, 4.3, and 4.4 (ii) hold. Then, the iterates of* `FedRobust` *after T iterations satisfy:*

$$\frac{1}{T}\sum_{t=0}^{T-1}\mathbb{E}\left\|\nabla\Phi(\overline{\mathbf{w}}_t)\right\|^2 \le \frac{4\Delta_\Phi}{\eta_1 T} + \frac{4L_2^2}{\mu_2^2 n^2}\frac{\epsilon^2}{\eta_1 T} + 64\eta_1\tilde{L}(\tau-1)^2\rho^2 + 16\eta_1\tilde{L}(\tau-1)\frac{n+1}{n}\sigma_{\boldsymbol{w}}^2 + 2\eta_1\hat{L}\frac{\sigma_{\boldsymbol{w}}^2}{n} + \frac{\eta_2^2}{\eta_1}L_2\sigma_\psi^2,$$

with $\tilde{L}, \hat{L}, L_\Phi, \rho^2$ defined in Theorem 4.1, $\epsilon^2 := \epsilon_1^2 + \epsilon_2^2$ and $\Delta_\Phi := \Phi(\mathbf{w}_0) - \Phi^*$, if step-sizes η_1, η_2 satisfy

$$\eta_2 \le \frac{1}{L_2}, \quad \frac{\eta_1}{\eta_2} \le \frac{\mu_2^2 n^2}{8L_{12}^2}, \quad 32\eta_1^2(\tau-1)^2L_1^2 \le 1, \quad \eta_1\left(\hat{L} + 40\tilde{L}(\tau-1)^2\right) \le 1.$$

Proof: We defer the proof to Appendix C.3. ■

It is worth noting this theorem also recovers the existing results for distributed minimax optimization, i.e. $\tau = 1$ [18] and standard federated learning for nonconvex objectives, i.e. $\epsilon_1, \epsilon_2 \to 0$ [74, 93].

4.4.2 Generalization guarantees

Following the margin-based generalization bounds developed in [88,102,103], we consider the following margin-based error measure for analyzing the generalization error in `FLRA` with general neural network classifiers:

$$\mathcal{L}_{\gamma}^{\mathrm{adv}}(\mathbf{w}) := \frac{1}{n}\sum_{i=1}^{n} \mathrm{Pr}_i \left(f_{\mathbf{w}}(h_{adv}^{i}(\mathbf{X}))[Y] - \max_{j\neq Y} f_{\mathbf{w}}(h_{adv}^{i}(\mathbf{X}))[j] \leq \gamma \right). \tag{4.6}$$

Here, h_{adv}^{i} denotes the worst-case affine transformation for node i in the maximization problem (4.2); Pr_i denotes the probability measured by the underlying distribution of node i, and $f_{\mathbf{w}}(\mathbf{x})[j]$ denotes the output of the neural network's last softmax layer for label j. Note that for $\gamma = 0$, the above definition reduces to the average misclassfication rate under the distribution shifts, which we simply denote by $\mathcal{L}^{\mathrm{adv}}(\mathbf{w})$. We also use $\hat{\mathcal{L}}_{\gamma}^{\mathrm{adv}}(\mathbf{w})$ to denote the above margin risk for the empirical distribution of samples, where we replace the underlying Pr_i with $\hat{\mathrm{Pr}}_i$ being the empirical probability evaluated for the m samples of node i. The following theorem bounds the difference of the empirical and underlying margin-based error measures in (4.6) for a general deep neural network function.

Theorem 4.3 *Consider an L-layer neural network with d neurons per layer. We assume the activation function of the neural network σ satisfies $\sigma(0) = 0$ and $\max_t\{|\sigma'(t)|, |\sigma''(t)|\} \leq 1$. Suppose the same Lipschitzness and smoothness condition holds for loss ℓ, and $\|\mathbf{X}\|_2 \leq B$. We assume the weights of the neural network are spectrally regularized such that for $M > 0$: $\frac{1}{M} \leq (\prod_{i=1}^{d} \|\mathbf{w}_i\|_{\sigma})^{1/d} \leq M$ with $\|\cdot\|_{\sigma}$ denoting the spectral norm. Also, suppose that for $\eta > 0$, $\mathrm{Lip}(\nabla f_{\mathbf{w}}) := \sum_{i=1}^{d}\prod_{j=1}^{i} \|\mathbf{w}_i\|_{\sigma} \leq \lambda(1-\eta)$ holds where $\mathrm{Lip}(\nabla f_{\mathbf{w}})$ upper-bounds the Lipschitz coefficient of the gradient $\nabla_{\mathbf{x}}\ell(f_{\mathbf{w}}(\mathbf{x}, y))$. Then, for* *every $\xi > 0$ with probability at least $1-\xi$ the following holds for all feasible weights $\mathbf{w}$:*

$$\mathcal{L}^{\mathrm{adv}}(\mathbf{w}) - \hat{\mathcal{L}}^{\mathrm{adv}}_{\gamma}(\mathbf{w})$$
$$\leq \mathcal{O}\left(\sqrt{\frac{B^2L^2d\log(Ld)\lambda^2\big(\prod_{i=1}^{L}\|\mathbf{w}_i\|_\sigma\sum_{i=1}^{L}\frac{\|\mathbf{w}_i\|_F^2}{\|\mathbf{w}_i\|_\sigma^2}\big)^2 + L\log\frac{nmL\log(M)}{\eta\xi}}{m\gamma^2(\lambda-(1+B)\,\mathrm{Lip}(\nabla f_{\mathbf{w}}))^2}}\right).$$

Proof: We defer the proof to Appendix C.5. ■

This theorem gives a non-asymptotic bound on the generalization risk of `FLRA` for spectrally regularized neural nets with their smoothness constant bounded by λ. Thus, we can control the generalization performance by properly regularizing the Lipschitzness and smoothness degrees of the neural net.

4.4.3 Distributional robustness

To analyze `FLRA`'s robustness properties, we draw a connection between `FLRA` and distributionally robust optimization using optimal transport costs. Consider the optimal transport cost $W_c(P,Q)$ for quadratic cost $c(\mathbf{x},\mathbf{x}') = \frac{1}{2}\|\mathbf{x}-\mathbf{x}'\|_2^2$ defined as $W_c(P,Q) := \min_{M\in\Pi(P,Q)} \mathbb{E}[c(\mathbf{X},\mathbf{X}')]$, where $\Pi(P,Q)$ denotes the set of all joint distributions on $(\mathbf{X},\mathbf{X}')$ with marginal distributions P,Q. In other words, $W_c(P,Q)$ measures the minimum expected cost for transporting samples between P and Q. In order to define a distributionally robust federated learning problem against affine distribution shifts, we consider the following minimax problem:

$$\min_{\mathbf{w}} \frac{1}{n}\sum_{i=1}^{n}\max_{\Lambda^i,\delta^i}\big\{\mathbb{E}_{P^i}\big[\ell\big(f_{\mathbf{w}}(\Lambda^i\mathbf{X}+\delta^i),Y\big)\big] - W_c(P^i_{\mathbf{X}},P^i_{\Lambda^i\mathbf{X}+\delta^i})\big\}. \tag{4.7}$$

In this distributionally robust learning problem, we include a penalty term controlling the Wasserstein cost between the original distribution of node i denoted by P^i and its perturbed version under an affine distribution shift, i.e., $P^i_{\Lambda^i\mathbf{X}+\delta^i}$. Note that here we use the averaged Wasserstein cost $\frac{1}{n}\sum_{i=1}^n W_c(P^i_{\mathbf{X}}, P^i_{\Lambda^i\mathbf{X}+\delta^i})$ to measure the distribution shift caused by the affine shifts $(\Lambda^i,\delta^i)_{i=1}^n$. The following theorem shows that this Wasserstein cost can be upper-bounded by a norm-squared function of Λ and δ that appears in the `FLRA`'s minimax problem.

Theorem 4.4 *Consider the Wasserstein cost $W_c(P_{\mathbf{X}}, P_{\Lambda\mathbf{X}+\delta})$ between the distributions of $\mathbf{X}$ and its affine perturbation $\Lambda\mathbf{X}+\delta$. Assuming $\|\mathbb{E}[\mathbf{X}\mathbf{X}^T]\|_\sigma \le \lambda$, we have*

$$W_c(P_{\mathbf{X}}, P_{\Lambda\mathbf{X}+\delta}) \le \max\{\lambda, 1\}\big[\|\Lambda - I\|_F^2 + \|\delta\|_2^2\big]. \tag{4.8}$$

Proof: We defer the proof to Appendix C.6. ■

Substituting the Wasserstein cost in (4.7) with the upper-bound (4.8) results in the `FLRA`'s minimax (4.3). As a result, if $\frac{\lambda}{n}\sum_{i=1}^n[\|\Lambda^i - I\|_F^2 + \|\delta^i\|_2^2] \le \varepsilon^2$ holds for the optimized Λ^i, δ^i's, we will also have the averaged Wasserstein cost bounded by $\frac{1}{n}\sum_{i=1}^n W_c(P^i_{\mathbf{X}}, P^i_{\Lambda^i\mathbf{X}+\delta^i}) \le \varepsilon^2$. Theorem 4.4, therefore, shows the `FLRA`'s minimax approach optimizes a lower-bound on the distributionally robust (4.7).

4.5 Numerical Results

We implemented `FedRobust` in the Tensorflow platform [104] and numerically evaluated the algorithm's robustness performance against affine distribution shifts and adversarial perturbations. We considered the standard MNIST [105] and CIFAR-10 [106] datasets and used three standard neural network architectures in the literature: AlexNet [107], Inception-Net [108], and a mini-ResNet [109].

In the experiments, we simulated a federated learning scenario with $n = 10$ nodes where each node observes $m = 5000$ training samples. We manipulated the training

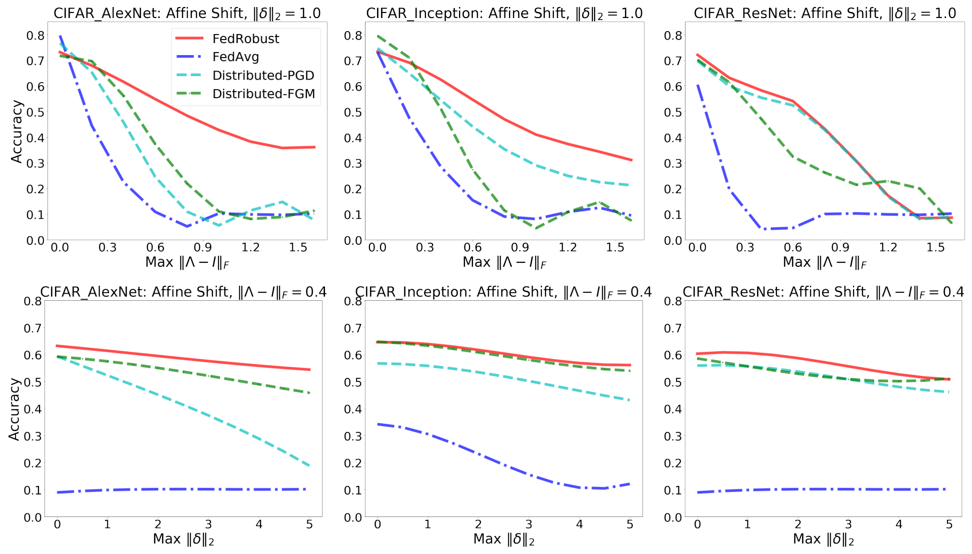


Figure 4.1: Test accuracy under affine distribution shifts over CIFAR-10. Top: constraining $\|\delta\|_2 \leq 1$ and changing maximum allowed $\|\Lambda - I\|_F$. Bottom: constraining $\|\Lambda - I\|_F \leq 0.4$ and changing maximum allowed $\|\delta\|_2$.

samples at each node via an affine distribution shift randomly generated according to a Gaussian distribution. We also used 5000 test samples for which we did not apply any random affine shift and instead considered the following two scenarios: (1) affine distribution shifts by optimizing the inner maximization in (4.1) using projected gradient descent (PGD); (2) ℓ_2-norm bounded adversarial PGD perturbations. We considered three baselines: (1) `FedAvg` where the server node averages the updated parameters of the local nodes after every gradient step; (2) distributed FGM training where the nodes perform fast adversarial training [84] by optimizing a norm-bounded perturbation δ_j^i using one gradient step followed by projection onto an ℓ_2-norm ball; (3) distributed PGD training where each node preforms PGD adversarial training [83] by applying 10 gradient steps where each step is followed by a projection onto an ℓ_2-norm ball.

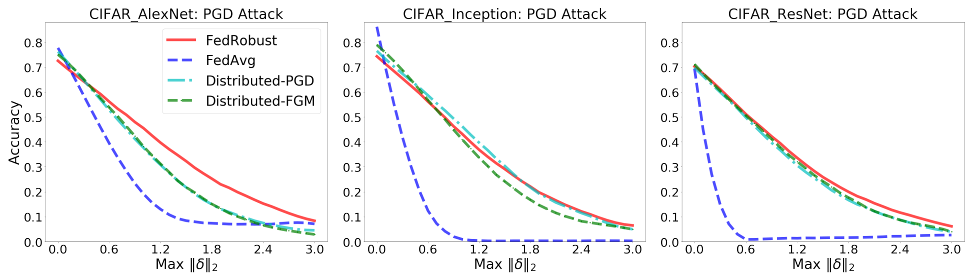


Figure 4.2: Test accuracy under PGD over CIFAR-10. X-axis shows the maximum allowed ℓ_2-norm for PGD.

4.5.1 `FedRobust` vs. `FedAvg` and adversarial training: Affine distribution shifts

We tested the performance of the neural net classifiers trained by `FedRobust`, `FedAvg`, distributed FGM, and distributed PGD under different levels of affine distribution shifts. Figure 4.1 shows the accuracy performance over CIFAR-10 with AlexNet, Inception-Net, and ResNet architectures. As demonstrated, `FedRobust` outperforms the baseline methods in most of the experiments. The improvement over `FedAvg` can be as large as 54%. Moreover, `FedRobust` improved over distributed FGM and PGD adversarial training, which suggests adversarial perturbations may not be able to capture the complexity of affine distribution shifts. `FedRobust` also results in $4\times$ faster training compared to distributed PGD. These improvements motivate `FedRobust` as a robust and efficient federated learning method to protect against affine distribution shifts.

4.5.2 `FedRobust` vs. `FedAvg` and adversarial training: Adversarial perturbations

Figure 4.2 summarizes our numerical results of `FedRobust` and other baselines over CIFAR-10 where the plots show the test accuracy under different levels of ℓ_2-norm perturbations. While we motivated `FedRobust` as a federated learning scheme protecting

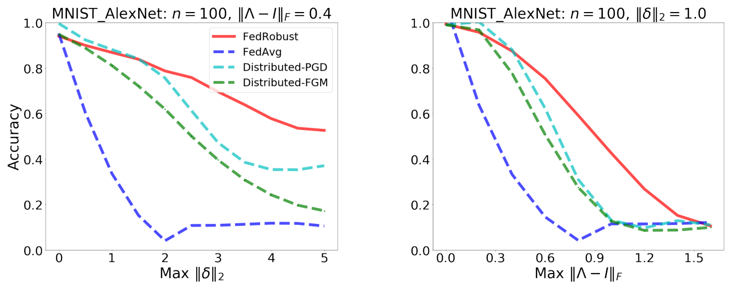


Figure 4.3: Test accuracy under affine perturbations for $n = 100$ nodes over MNIST data. X-axis shows the maximum allowed $\|\Lambda - I\|_F$ (left) and $\|\delta\|_2$ (right) for affine perturbations.

against affine distribution shifts, we empirically observed its robust performance against adversarial perturbations as well. The achieved adversarial robustness in almost all cases matches the robustness offered by distributed FGM and PGD adversarial training. This observation can be explained by analyzing the generalization properties of these algorithms. We note that `FedRobust`'s improved robustness is obtained over the *test* samples. On the other hand, PGD consistently outperformed `FedRobust` on the *training* samples, achieving a near perfect training accuracy. However, `FedRobust` generalized better to the test samples and could overall outperform PGD on the test set. Also, the similar performance of FGM and PGD can be explained via the random Gaussian perturbations used for simulating the heterogeneity across clients and the results of [**?**] indicating FGM initialized at random perturbations performs as well as PGD. These numerical results indicate that affine distribution shifts can cover the distribution changes caused by norm-bounded adversarial perturbations. In summary, our numerical experiments demonstrate the efficiency and robustness of `FedRobust` against PGD adversarial attacks. We defer more details of our experiments and the numerical results on MNIST data to the Appendix.

Finally, we performed additional numerical experiments to analyze the effect of network size n and minimization iteration count τ on the robustness performance. Figure 4.3

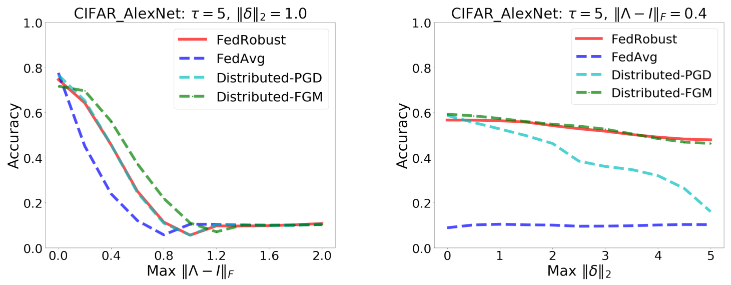


Figure 4.4: Test accuracy under affine perturbations for $\tau = 5$ minimization iteration count over CIFAR-10 data. X-axis shows the maximum allowed $\|\Lambda - I\|_F$ (left) and $\|\delta\|_2$ (right) for affine perturbations.

shows the results of our experiments for a larger network size of $n = 100$ AlexNet neural network classifiers, each trained using $m = 500$ MNIST training data points. As demonstrated in Figure 4.3's plots, `FedRobust` still outperforms the standard and adversarial training baselines over a wide range of affine perturbation parameters. To examine the effect of parameter τ, i.e., minimization step count per training iteration, on our experimental results, we performed the CIFAR-10 experiment with the AlexNet architecture for $\tau = 5$ as demonstrated in Figure 4.4. We observed that after increasing τ to 5, the robustness offered by `FedRobust` slightly decreased and was comparable to the performance of our adversarial training baselines. While `FedRobust` still outperforms `FedAvg` by a clear margin, the numerical results indicate the role of simultaneous min-max optimization and proper selection of hyperparameters in the success of `FedRobust`.

We conclude this section by reiterating the practicality of the considered affine model. As demonstrated in our experiments, the affine model considered in this chapter is particularly practical for image classification tasks in federated learning, where each camera's imperfections affect its pictures [11]. While this model provides significant robustness compared to additive-only perturbation models (i.e. $\Lambda = I$), it lays out potential new directions to study more complicated (non-affine) models such as neural network transformations.

4.5.3 Experimental Setup

In the experiments, we simulated a federated learning scenario with $n = 10$ nodes where each node observes $m = 5000$ training samples. We also divided the extra $10,000$ samples in each dataset to two validation and test sets containing 5000 samples each. For CIFAR-10 samples, we applied the sandard normalization and scaled and linearly mapped the pixel intensity values to interval $[-1, 1]$. We applied batch normalization [110] in order to stabilize training and used the ADAM optimizer [111] with stepsize value 10^{-4} and default beta parameters $\beta_1 = 0.9$ and $\beta_2 = 0.99$ to optimize the neural net's parameters for $T = 100$ epochs (10000 iterations).

We did cross validation to choose $\lambda \in \{0.1, 0.5, 1, 5, 10, 50\}$ and chose the λ-value resulting in the closest additive penalty $\frac{1}{n}\sum_{i=1}^{n}[\|\Lambda^{i^*} - I\|_2^2 + \|\delta^{i^*}\|_2^2]$ to 10 percent of the average sample norm, i.e. $\frac{0.1}{m}\sum_{i=1}^{m}\|\mathbf{x}_i^{\text{val}}\|_2^2$, over the $m = 5000$ validation samples. To perform GDA optimization, we applied two ascent steps per descent step with stepsize $\frac{1}{2\lambda}$. In order to simulate an affine distribution shift, we manipulated each $\tilde{\mathbf{x}}_j^i$ in the original training dataset via an affine transformation chosen randomly at each node:

$$\mathbf{x}_j^i = (I_d + \tilde{\Lambda}^i)\tilde{\mathbf{x}}_j^i + \tilde{\delta}^i. \tag{4.9}$$

Here, each $\tilde{\Lambda}^i$ is a random matrix with i.i.d. Gaussian entries according to $\mathcal{N}(0, \frac{\sigma^2}{d})$, and $\tilde{\delta}^i$ is a random Gaussian vector according to $\mathcal{N}(0, \sigma^2 I_d)$ where we set $\sigma = 0.01$. In test time, we did not apply any random affine transformation to test samples and instead considered the following three scenarios: (1) no perturbation, (2) adversarial affine distribution shift obtained by optimizing the inner maximization in (4.1) using projected gradient descent, 3) adversarial perturbations designed by the projected gradient descent algorithm. We used 100 projected gradient steps with stepsize 0.1.

We considered three baselines in the experiments: (1) `FedAvg` where the server node

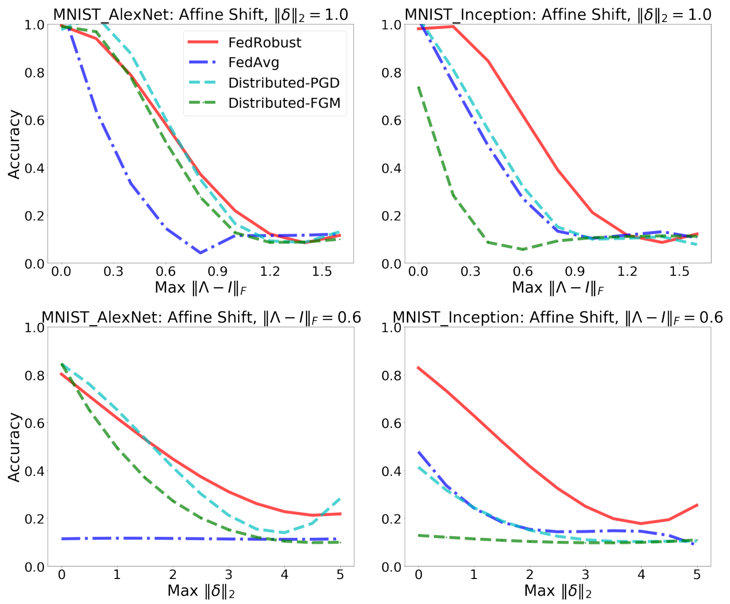


Figure 4.5: Trained networks' test accuracy under affine distribution shifts in the MNIST experiments. Top row: constraining $\|\delta\|_2 \le 1$ and changing maximum allowed $\|\Lambda - I\|_F$, bottom row: constraining $\|\Lambda - I\|_F \le 0.6$ and changing maximum allowed $\|\delta\|_2$.

averages the updated parameters of the local nodes after every gradient step, (2) Distributed FGM training where the nodes perform fast adversarial training [84] by optimizing an ℓ_2-norm bounded perturbation δ_j^i using one gradient step followed by projection onto the ball $\{\delta_j^i : \|\delta_j^i\|_2 \le \epsilon_{\text{fgm}}\}$, and (3) Distributed PGD training where each node preforms PGD adversarial training [83] similar to distributed FGM but uses 10 projected gradient steps, each followed by projection onto $\{\delta_j^i : \|\delta_j^i\|_2 \le \epsilon_{\text{pgd}}\}$. We used the value $\epsilon_{\text{fgm}} = \epsilon_{\text{pgd}} = 0.05\mathbb{E}[\|\mathbf{x}_i\|_2]$ in the experiments. We observed training instability after achieving perfect training accuracy for the baseline `FedAvg` algorithm, and hence performed early stopping to avoid the instability in the `FedAvg` experiments. We did not encounter the instability issue in `FedRobust` experiments.

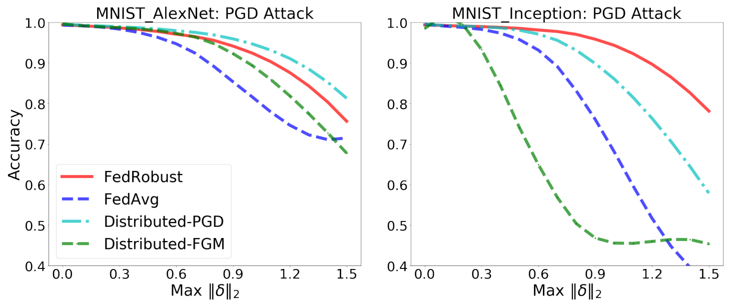


Figure 4.6: Trained networks' test accuracy under PGD perturbations in the MNIST experiments. X-axis shows the maximum allowed ℓ_2-norm for PGD perturbations.

4.5.4 Numerical Results for MNIST data

We repeated the CIFAR experiments in Figures 4.1 and 4.2 for the MNIST dataset. Figure 4.5 shows the numerical results under affine distribution shifts. The figure's top row includes the plots for fixed maximum delta norm $\|\delta\|_2 \leq 1$ and different levels of maximum allowed $\|\Lambda - I\|_F$, while in the bottom row we fix the maximum allowed linear shift $\|\Lambda - I\|_F \leq 0.6$ and evaluate the test accuracy under different levels of $\|\delta\|_2$. As shown in the plots, `FedRobust` results in the best performance in most of the evaluations, which indicates the superior performance of `FedRobust` against affine distribution shifts. Figure 4.6 shows the test accuracy of the trained networks under different levels of adversarial PGD perturbations. The figure's experiments again shows that `FedRobust` can effectively shield against PGD adversarial attacks and achieve a comparable performance to PGD and FGM adversarial training.

4.6 Concluding Remarks

We tackled data heterogeneity challenge in federated learning paradigms in this chapter, and proposed `FLRA`, a federated learning scheme that is robust to affine distribution shifts across the devices. Our proposal makes the trained model robust to worst-case affine transformations on the data samples by formulating the model training via a minimax optimization problem. Moreover, we proposed a GDA optimization routine, named as `FedRobust` to solve the minimax problem while imposing light computation and communication overheads. We established optimization and generalization characteristics of the proposed design and demonstrated its practical implications and improvements over existing federated and adversarial training methods.

Part II

Algorithms for Distributed Computing

Chapter 5

Coded Computation over Heterogeneous Clusters

In large-scale distributed computing clusters, such as Amazon EC2, there are several types of "system noise" that can result in major degradation of performance: system failures, bottlenecks due to limited communication bandwidth, latency due to straggler nodes, etc. There have been recent results that demonstrate the impact of coding for efficient utilization of computation and storage redundancy to alleviate the effect of stragglers and communication bottlenecks in *homogeneous* clusters. In this chapter, we focus on general *heterogeneous* distributed computing clusters consisting of a variety of computing machines with different capabilities. We propose a coding framework for speeding up distributed computing in heterogeneous clusters by trading redundancy for reducing the latency of computation. In particular, we propose **H**eterogeneous **C**oded **M**atrix **M**ultiplication (`HCMM`) algorithm for performing distributed matrix multiplication over heterogeneous clusters that is provably asymptotically optimal for a broad class of processing time distributions. Moreover, we show that `HCMM` is unboundedly faster than *any uncoded* scheme that partitions the total work load among the workers. To demonstrate how the proposed HCMM scheme can be applied in practice, we provide results from numerical studies and Amazon EC2 experiments comparing HCMM with three benchmark load allocation schemes – Uniform Uncoded, Load-balanced Uncoded, and Uniform Coded. In particular, in our numerical studies, HCMM achieves speedups of up to 73%, 56% and 42% respectively over the three benchmark schemes mentioned above. Furthermore, we carry out experiments over Amazon EC2 clusters and demonstrate how HCMM can be combined with rateless codes with nearly linear decoding complexity. In particular, we show that HCMM combined with the Luby transform (LT) codes can significantly reduce the overall execution time. HCMM is found to be up to 61%, 46% and 36% faster than the aforementioned three benchmark schemes, respectively. Additionally, we provide a generalization to the problem of optimal load allocation in heterogeneous settings, where we take into account the monetary costs associated with distributed computing clusters. We argue that HCMM is asymptotically optimal for budget-constrained scenarios as well. In particular, we characterize the minimum possible expected cost associated with a computation task over a given cluster of machines. Furthermore, we develop a heuristic for HCMM load allocation for the distributed implementation of budget-limited computation tasks.

5.1 Introduction

General distributed computing frameworks, such as MapReduce [12] and Spark [13], along with the availability of large-scale commodity servers, such as Amazon EC2, have made it possible to carry out large-scale data analytics at the production level. These "virtualized data centers" enjoy an abundance of storage space and computing power, and are cheaper to rent by the hour than maintaining dedicated data centers round the year. However, these systems suffer from various forms of "system noise" which reduce their efficiency: system failures, limited communication bandwidth, straggler nodes, etc.

The current state-of-the-art approaches to mitigate the impact of system noise in cloud computing environments involve creation of some form of "computation redundancy". For example, *replicating* the straggling task on another available node is a common approach to deal with stragglers [14], while partial data replication is also used to reduce the communication load in distributed computing [15]. However, there have been recent results demonstrating that *coding* can play a transformational role for creating and exploiting computation redundancy to effectively alleviate the impact of system noise. In particular, there have been two coding concepts proposed to deal with the communication and straggler bottlenecks in distributed computing.

The first coding concept introduced in [112–114] enables an inverse-linear tradeoff between computation load and communication load in distributed computing. This result implies that increasing the computation load by a factor of r (i.e. evaluating each computation at r carefully chosen nodes) can create novel coding opportunities that reduce the required communication load for computing by the same factor r. Hence, these codes can be utilized to pool the underutilized computing resources at network edge to slash the communication load of Fog computing [115]. Other related works tackling the communication bottleneck in distributed computation include [116–120].

In the second coding concept introduced in [116], an inverse-linear tradeoff between computation load and computation latency (i.e. the overall job response time) is established for distributed matrix multiplication in homogeneous computing environments. More specifically, this approach utilizes coding to effectively inject redundant computations to alleviate the effects of stragglers and speed up the computations. Hence, by utilizing more computation resources, this can significantly speed up distributed computing applications. A number of related works have been proposed recently to mitigate stragglers in distributed computation. In [121], the authors propose the use of redundant short dot products to speed up distributed computation of linear transforms. The work in [122] proposes coding schemes for mitigating stragglers in distributed batch gradient computation. Coding schemes for high-dimensional matrix-matrix multiplication have been developed in [123–127]. Techniques for efficient straggler mitigation for matrix-vector computation in distributed wireless settings have been developed in [128]. In [129], the potential of the multicore nature of computing machines is studied. In [130], the authors propose an anytime approach to distributed computing, developing an approximate matrix multiplication scheme. The authors in [131] propose a novel encoding scheme for achieving large sparsity in the encoded matrix. Work in [132] develops a coding strategy for mitigating straggling decoders in cloud radio access network. Speeding up the computation of linear transformations with unreliable components is studied in [133]. Straggler mitigation through data encoding in distributed optimization is proposed in [134]. A coded scheme based on LT codes is proposed in [135] for multiplying a matrix by a set of vectors in a distributed computing environment. Addressing stragglers has attracted a lot of attention in the queuing-based frameworks for large-scale computation as well [136, 137]. These works utilize the technique of dynamically replicating the tasks in a careful manner to minimize run-time.

We extend the problem of distributed matrix multiplication in homogeneous clusters in [116] to heterogeneous environments. As discussed in [14], the computing environments in virtualized data centers are heterogeneous and algorithms based on homogeneous assumptions can result in significant performance reduction. In this paper, we focus on general *heterogeneous* distributed computing clusters consisting of a variety of computing machines with different capabilities. Specifically, we propose a coding framework for speeding up distributed matrix multiplication in heterogeneous clusters with straggling servers, named **H**eterogeneous **C**oded **M**atrix **M**ultiplication (`HCMM`). Matrix multiplication is a crucial computation module in many engineering and scientific disciplines. In particular, it is a fundamental component of many popular machine learning algorithms such as logistic regression, reinforcement learning and gradient descent-based algorithms. Implementations that speed up matrix multiplication would naturally speed up the execution of a wide variety of popular algorithms. Therefore, we envision HCMM to play a fundamental role in speeding up big data analytics in virtualized data centers by leveraging the wide range of computing capabilities provided by these heterogeneous environments.

We now describe the main ideas behind HCMM, which results in asymptotically optimal performance. In a coded implementation of distributed matrix-vector multiplication, each worker node is assigned the task of computing inner products of the assigned coded rows with the input vector, where the assigned coded rows are random linear combinations of the rows of the original matrix. Computation time at each worker is a random variable, which is first assumed to have shifted exponential distribution, and we later generalize it to shifted Weibull distribution. The master node receives the results from the worker nodes and aggregates them until it receives a decodable set of inner products and recovers the matrix-vector multiplication. We are interested in finding the optimal load allocation that minimizes the expected waiting time to complete this computation. However, due to heterogeneity, finding the exact solution to the optimization problem seems intractable.

As the main contribution of the paper, we propose an alternative optimization that focuses on maximizing the expected number of returned computation results from the workers. Apart from being computationally tractable, the alternative optimization asymptotically approximates the problem of finding the optimal computation load allocation. Specifically, we develop the HCMM algorithm that is derived as a solution to the alternative formulation, and prove it is asymptotically optimal. Furthermore, we prove that given a heterogeneous cluster of n workers, HCMM is $\Theta(\log n)$ times faster than uncoded schemes under the shifted exponential distribution for run-time. We further generalize the proposed HCMM algorithm to shifted Weibull model and provide similar unbounded gains over uncoded scenarios.

In addition to proving the asymptotic optimality of `HCMM`, we carry out numerical studies and experiments over Amazon EC2 clusters to demonstrate how `HCMM` can be used in practice. We compare `HCMM` with three benchmark schemes – Uniform Uncoded, Load-balanced Uncoded, and Uniform Coded. In our numerical analysis, `HCMM` results in significant speedups of up to 73%, 56% and 42% over the three aforementioned benchmark schemes, respectively. In experiments using Amazon EC2 clusters, we use the Luby transform (LT) codes for coding and demonstrate that `HCMM` combined with LT codes significantly reduces the overall execution time in comparison to uncoded and coded schemes. In particular, `HCMM` achieves gains of up to 61%, 46% and 36%, respectively over Uniform Uncoded, Load-balanced Uncoded and Uniform Coded. Furthermore, the overall computation load of `HCMM` is less than the one of Uniform Coded. Our results demonstrate that `HCMM` combines the benefits of both Load-balanced Uncoded and Uniform Coded schemes by achieving efficient load balancing along with minimal number of redundant computations.

Furthermore, we consider the problem of load allocation under budget constraints, considering an intuitive and convincing pricing model. In particular, we show that `HCMM` is the (asymptotically) optimal load allocation in feasible budget-constrained scenarios as well, and determine whether a budget-constrained computation task is feasible given a cluster of machines. We then develop a heuristic algorithm to find the (sub)optimal load allocations using the proposed `HCMM` scheme. The heuristic is based on the observation that given a computation task and a set of machines, decreasing the number of fastest machines participating in `HCMM` results in smaller average cost.

Notation. We denote by $[n]$ the set $\{1, \cdots, n\}$ for any $n \in \mathbb{N}$. For non-negative sequences $g(n)$ and $h(n)$, we denote $g(n) = \mathcal{O}\big(h(n)\big)$ if there exist constants $c > 0$ and $n_0 \in \mathbb{N}$ such that $g(n) \leq c \cdot h(n)$ for all $n > n_0$; and $g(n) = \Theta\big(h(n)\big)$ if $g(n) = \mathcal{O}\big(h(n)\big)$

and $h(n) = \mathcal{O}\big(g(n)\big)$. Moreover, we write $g(n) = o\big(h(n)\big)$ if $\lim_{n\to\infty} g(n)/h(n) = 0$.

5.2 Problem Formulation and Main Results

In this section, we describe our computation model, the network model and the precise problem formulation. We then conclude with four theorems highlighting the main contributions of the chapter.

5.2.1 Computation Model

We consider the problem of matrix-vector multiplication, in which given a matrix $\mathbf{A} \in \mathbb{R}^{r\times m}$ for some positive integers r and m, we want to compute the output $\mathbf{y} = \mathbf{A}\mathbf{x}$ for an input vector $\mathbf{x} \in \mathbb{R}^m$. Due to limited computing power, the computation cannot be carried out at a single server and a distributed implementation is required. As an example, consider a matrix $\mathbf{A}$ with an even number of rows and two computing nodes. The matrix can be divided into two equally tall matrices $\mathbf{A}_1$ and $\mathbf{A}_2$, and each will be stored in a different worker node. The master node receives the input $\mathbf{x}$ and broadcasts it to the two worker nodes. These nodes will then compute $\mathbf{y}_1 = \mathbf{A}_1\mathbf{x}$ and $\mathbf{y}_2 = \mathbf{A}_2\mathbf{x}$ locally and return their results to the master node, which combines them to obtain the intended outcome $\mathbf{y} = [\mathbf{y}_1; \mathbf{y}_2] = \mathbf{A}\mathbf{x}$. This example also illustrates an uncoded implementation of distributed computing, in which results from all the worker nodes are required to recover the final result.

We now present the formal definition of *Coded Distributed Computation.*

Definition 5.1 ***(Coded Distributed Computation)*** *The coded distributed implementation of a computation task $f_{\mathbf{A}}(\cdot)$ is specified by:*

- *local data blocks $\langle \mathbf{A}_i \rangle_{i=1}^n$ and local computation tasks $\left\langle f^i_{\mathbf{A}_i}(\cdot) \right\rangle_{i=1}^n$;*

- *a decoding function that outputs $f_{\mathbf{A}}(\cdot)$ given the results from a decodable set of local computations.*

For matrix-vector multiplication tasks in particular, local data blocks $\mathbf{A}_i \in \mathbb{R}^{\ell_i \times m}$ are matrices consisting of *coded* combinations of the rows in $\mathbf{A}$, for non-negative integers ℓ_i. To assign the computation tasks to each worker, we use random linear combinations of the r rows of the matrix $\mathbf{A}$, such that the master node can recover the result $\mathbf{Ax}$ from any r inner products received from the worker nodes with probability 1. As an example, if worker i is assigned a matrix-vector multiplication with matrix size $\ell_i \times m$, it will compute ℓ_i inner products of the assigned coded rows of $\mathbf{A}$ with $\mathbf{x}$. The master node shall wait for the first r inner products and will use them to decode the required output. In order to ensure the recovery of the output from any r inner products received from the workers, we pick the computation matrix assigned to worker i as $\mathbf{A}_i = \mathbf{S}_i\mathbf{A}$, where $\mathbf{S}_i \in \mathbb{R}^{\ell_i \times r}$ is the coding matrix with i.i.d. $\mathcal{N}(0,1)$ entries. Worker i computes $\mathbf{A}_i\mathbf{x}$ and returns the result to the master node. Upon receiving r inner products, the aggregated results at the master will be in the form of $\mathbf{z} = \mathbf{S}_{(r)}\mathbf{Ax}$, where $\mathbf{S}_{(r)} \in \mathbb{R}^{r \times r}$ is the aggregated coding matrix, and it is full-rank with probability 1 [138]. Therefore, the master node can recover $\mathbf{Ax} = \mathbf{S}_{(r)}^{-1}\mathbf{z}$ with probability 1.[1,2]

[1]Although we consider random linear coding in our theoretical analysis, other codes such as Maximum-Distance Separable (MDS) codes and Luby transform (LT) codes are compatible with HCMM as well, given a decodable set of results at the master. For example, in the MDS case, the entries in the coding matrix $\{\mathbf{S}_i\}_{i=1}^n$ are drawn from a finite field. Specifically, one can encode the rows of $\mathbf{A}$ using an $(\sum_{i=1}^n \ell_i, r)$ MDS code and assign ℓ_i coded rows to the worker node i. The output $\mathbf{Ax}$ can be recovered from the inner products of any r coded rows with the input vector $\mathbf{x}$. Furthermore, to implement the ideas developed in this work, we use LT codes in our experiments over Amazon EC2 clusters.

[2]Instead of i.i.d. Gaussian, we could use any continuous distribution for the random entries, since Schwartz-Zippel lemma ensures that such random matrix is full-rank with high probability

5.2.2 Network Model

The network model is based on a master-worker setup illustrated in Fig. 5.1. The master node receives an input $\mathbf{x}$ and broadcasts it to all the workers. Each worker computes its assigned set of computations and unicasts the result to the master node. The master node aggregates the results from the worker nodes until it receives a decodable set of computations and recovers the output $\mathbf{Ax}$.

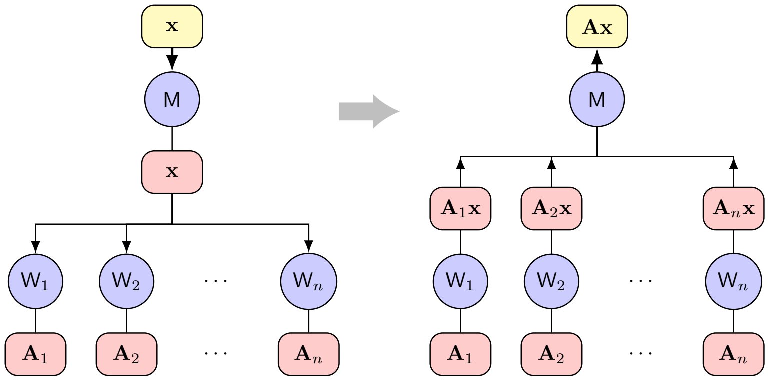


Figure 5.1: Master-worker setup of the computing clusters: The master node receives the input vector $\mathbf{x}$ and broadcasts it to all the worker nodes. Upon receiving the input, worker node i starts computing the inner products of the input vector with the locally assigned rows, i.e., $\mathbf{y}_i = \mathbf{A}_i\mathbf{x}$, and unicasts the output vector $\mathbf{y}_i$ to the master node upon completing the computation. The results are aggregated at the master node until r inner products are received and the desired output $\mathbf{Ax}$ is recovered.

We denote by T_i the random variable representing the task run-time at node i and assume that the run-times $T_1, \cdots, T_n$ are mutually independent. We consider the distribution of run-time random variables to be exponential, and later generalize it to Weibull distribution. More specifically, we consider a 2-parameter shifted exponential distribution for the execution time of each worker, i.e., the CDF of execution time of worker node i, T_i, loaded with ℓ_i row vectors is as follows:

$$\Pr[T_i \le t] = 1 - e^{-\frac{\mu_i}{\ell_i}(t - a_i\ell_i)}, \tag{5.1}$$

for $t \geq a_i \ell_i$ and $i \in [n]$, where $a_i > 0$ is the shift parameter and $\mu_i > 0$ denotes the straggling parameter associated with worker node i. The shifted exponential model for computation time, which is the sum of a constant (deterministic) term and a variable (stochastic) term, is motivated by the distribution model proposed by authors in [139] for latency in querying data files from cloud storage systems. As demonstrated in [116] as well as by our own experiments, exponential model provides a good fit for the distribution of computation times over cloud computing environments such as Amazon EC2 clusters. Moreover, these experiments confirm the assumption that as a first order approximation, both shift and mean parameters of the shifted exponential distributions linearly scale with the load size.

We further generalize the analysis to shifted Weibull distribution in Section 5.4, where we consider a 3-parameter shifted Weibull distribution for the execution time of each worker. That is, the CDF of task run-time at worker node i, loaded with ℓ_i row vectors is as follows:

$$\Pr[T_i \leq t] = 1 - e^{-\left(\frac{\mu_i}{\ell_i}(t - a_i \ell_i)\right)^{\alpha_i}}, \tag{5.2}$$

for $t \geq a_i \ell_i$ and $i \in [n]$, where $a_i > 0$ denotes the shift parameter, $\mu_i > 0$ is the straggling parameter and $\alpha_i > 0$ represents the shape parameter associated with worker i. A similar model has been considered in [140] as well.

5.2.3 Problem Formulation

We consider the problem of using a cluster of n worker nodes for distributedly computing the matrix-vector multiplication $\mathbf{Ax}$, where $\mathbf{A}$ is a size $r \times m$ matrix for positive integers r and m. Let $\boldsymbol{\ell} = (\ell_1, \cdots, \ell_n)$ be the load allocation vector where ℓ_i denotes the number of rows assigned to worker node i. Let T_{CMP} be the random variable denoting the waiting time for receiving a decodable set of results, i.e. at least r inner products.

We aim at finding the optimal load allocation vector that minimizes the average waiting time by solving the following optimization problem:

$$\mathcal{P}_{\text{main}}: \quad \underset{\boldsymbol{\ell}}{\text{minimize}} \quad \mathbb{E}[T_{\textsf{CMP}}]. \tag{5.3}$$

For a homogeneous cluster, to achieve a coded solution, one can divide $\mathbf{A}$ into k equal size submatrices, and apply an (n, k) MDS code to these submatrices. The master node can then obtain the final result from any k responses. In [116], the authors find the optimal k for minimizing the average running time for the shifted exponential run-time model.

For heterogeneous clusters, however, assigning equal loads to servers is clearly not optimal. Moreover, directly finding the optimal solution to $\mathcal{P}_{\text{main}}$ is hard. In homogeneous clusters, the problem of finding a sufficient number of inner products can be mapped to the problem of finding the waiting time for a set of fastest responses, and thus closed form expressions for the expected computation time can be found using order statistics of i.i.d. run-times. However, this is not straight-forward in heterogeneous clusters, where the load allocation is non-uniform. In Section 5.3, we present an alternative formulation to $\mathcal{P}_{\text{main}}$ in (5.3), and show that the solution to the alternative formulation – which we shall name `HCMM` – is tractable and provably asymptotically optimal.

Assumptions. From now onward, we consider the practically relevant regime where the size of the problem scales linearly with the size of the network, while the computing power and the storage capacity of each worker node remain constant. Specifically, we assume $r = \Theta(n)$, $a_i = \Theta(1)$, $\mu_i = \Theta(1)$ and $\alpha_i = \Theta(1)$ for each worker i.

5.2.4 Main Results

Having set the model and formulation of the problem, we now present the main contributions of this chapter. The following theorem characterizes the asymptotic optimality of `HCMM` for the shifted exponential run-time model.

Theorem 5.1 *Let* $T_{\texttt{HCMM}}$ *be the random variable denoting the finish time of the* `HCMM` algorithm and T_{OPT} be the random variable representing the finish time of the optimum algorithm obtained by solving $\mathcal{P}_{\text{main}}$. Then, for shifted exponential run-times in (5.1) with constant parameters $a_i = \Theta(1)$ and $\mu_i = \Theta(1)$ for each worker $i \in [n]$ and $r = \Theta(n)$, we have $\lim_{n\to\infty} \mathbb{E}[T_{\texttt{HCMM}}] = \lim_{n\to\infty} \mathbb{E}[T_{\mathsf{OPT}}]$.

Proof: We defer the proof to Appendix D.1. ■

Remark 5.1 *Theorem 5.1 demonstrates that our proposed* `HCMM` *algorithm is asymptotically optimal as the number of workers n approaches infinity. In other words, the optimal computation load allocation problem* $\mathcal{P}_{\text{main}}$ *in (5.3) can be optimally solved using the proposed* `HCMM` *algorithm as n gets large.*

Remark 5.2 *We note that* $\mathcal{P}_{\text{main}}$ *in (5.3) is a hard combinatorial optimization problem since it will require checking all load combinations to minimize the overall expected execution time. The key idea in Theorem 5.1 is to consider an alternative formulation to (5.3) focusing on maximizing the expected number of returned computation results from the workers, i.e. maximizing the aggregate return. As we describe in Section 5.3, the alternative optimization problem not only can be solved efficiently in a tractable way giving rise to* `HCMM` *algorithm, it also asymptotically approximates* $\mathcal{P}_{\text{main}}$ *and allows us to establish Theorem 5.1.*

Remark 5.3 *While Theorem 5.1 theoretically characterizes the optimality of our proposed scheme* `HCMM`*, we also demonstrate gains that one can get in practice. In particular,* *we carry out numerical studies and experiments over Amazon EC2 clusters that demonstrate that* `HCMM` *can provide significant gains in a wide variety of computing scenarios. In particular, we compare* `HCMM`*'s performance with three benchmark load allocation policies – Uniform Uncoded, Load-balanced Uncoded, and Uniform Coded. In numerical studies,* `HCMM` *achieves speedups of up to* 71% *over Uniform Uncoded, up to* 53% *over Load-balanced Uncoded, and up to* 39% *over Uniform Coded. In EC2 experiments,* `HCMM` *combined with the Luby transform (LT) codes provides speedups of up to* 61%, 46% *and* 36% *over Uniform Uncoded, Load-balanced Uncoded and Uniform Coded, respectively.*

Theorem 5.2 *Let T_{UC} denote the completion time of the uncoded distributed matrix multiplication algorithm. Then, for the shifted exponential run-times with constant parameters and $r = \Theta(n)$,*

$$\frac{\mathbb{E}[T_{\mathsf{UC}}]}{\mathbb{E}[T_{\mathtt{HCMM}}]} = \Theta\big(\log n\big).$$

Proof: We defer the proof to Appendix D.2. ■

Remark 5.4 *As Theorem 5.2 shows, our proposed* `HCMM` *guarantees an improvement of $\Theta\big(\log n\big)$ in expected execution time over* any *uncoded scheme, including the one that optimally allocates the workers' loads. This result illustrates that by leveraging coded computing, one achieves the same order-wise gain over heterogeneous clusters as over homogeneous clusters [116].*

Although Theorems 5.1 and 5.2 are based on the shifted exponential model (5.1) for run-time random variables for the workers, our analyses are general and can be extended to other models. The following two theorems generalize the results when the execution time of each worker follows the Weibull distribution as described in (5.2).

Theorem 5.3 *For the shifted Weibull distribution of run-times with constant parameters $a_i = \Theta(1)$, $\mu_i = \Theta(1)$ and $\alpha_i = \Theta(1)$ for each worker $i \in [n]$ and $r = \Theta(n)$, the proposed* `HCMM` *algorithm is asymptotically optimal, i.e.,* $\lim_{n\to\infty} \mathbb{E}[T_{\mathtt{HCMM}}] = \lim_{n\to\infty} \mathbb{E}[T_{\mathsf{OPT}}]$.

Theorem 5.4 *Under the Weibull distribution for run-times with constant parameters and $r = \Theta(n)$, the proposed* HCMM *scheme unboundedly outperforms the uncoded scheme, i.e.,*

$$\frac{\mathbb{E}[T_{\mathsf{UC}}]}{\mathbb{E}[T_{\texttt{HCMM}}]} \geq \Theta\big((\log n)^{1/\tilde{\alpha}}\big),$$

where $\tilde{\alpha} = \max_{i\in[n]} \alpha_i$ is the largest shape parameter among the workers.

Remark 5.5 *As stated in Theorem 5.4,* HCMM *provides an unbounded gain over* any *uncoded scheme – including the optimal uncoded load allocation – under the Weibull distribution for workers' run-times. Furthermore, our numerical simulations demonstrate speedups of up to* 73%, 56% *and* 42% *over Uniform Uncoded, Load-balanced Uncoded and Uniform Coded, respectively.*

In the following section, we describe our alternative formulation based on aggregate return and describe our proposed HCMM algorithm that solves the alternative optimization.

5.3 The Proposed HCMM Scheme

In this section, we prove Theorems 5.1 and 5.2 for the exponential model (5.1). In particular, we start by describing the HCMM algorithm and show that it asymptotically achieves the optimal performance, as stated in Theorem 5.1, and lastly conclude the section by characterizing the gain of HCMM over uncoded scheme.

To derive HCMM, we start by reformulating $\mathcal{P}_{\text{main}}$ defined in (5.3) and show that the alternative formulation can be efficiently solved, as opposed to solving $\mathcal{P}_{\text{main}}$ that needs an exhaustive search over all possible load allocations. The solution to the alternative problem gives rise to HCMM. We will further prove the optimality of HCMM and compare its average run-time to uncoded schemes.

5.3.1 Alternative Formulation of $\mathcal{P}_{\text{main}}$ via Maximal Aggregate Return

Consider an n-tuple load allocation $\boldsymbol{\ell} = (\ell_1, \cdots, \ell_n)$ and let t be a feasible time for computation, i.e., $t \geq \max_i \{a_i \, \ell_i\}$. The number of equations received from worker $i \in [n]$ at the master node till time t is a random variable, $X_i(t) = \ell_i \, \mathbb{1}_{\{T_i \leq t\}}$, where T_i is the random execution time for machine i that is assigned the load ℓ_i and $\mathbb{1}_{\{.\}}$ denotes the indicator function. Then, the *aggregate return* at the master node at time t is:

$$X(t) = \sum_{i=1}^{n} X_i(t).$$

We propose the following two-step alternative formulation for $\mathcal{P}_{\text{main}}$ defined in (5.3). First, for a fixed feasible time t, we maximize the aggregate return over different load allocations, i.e., we solve

$$\mathcal{P}_{\text{alt}}^{(1)} : \boldsymbol{\ell}^*(t) = \arg\max_{\boldsymbol{\ell}} \mathbb{E}\big[X(t)\big]. \tag{5.4}$$

Then, given the load allocation $\boldsymbol{\ell}^*(t) = \big(\ell_1^*(t), \cdots, \ell_n^*(t)\big)$ obtained from $\mathcal{P}_{\text{alt}}^{(1)}$, we find the smallest time t such that with high probability, there is enough aggregate return by time t at the master node, i.e., we solve

$$\begin{aligned} \mathcal{P}_{\text{alt}}^{(2)} : \quad & \text{minimize} \quad t \\ & \text{subject to} \quad \Pr\big[X^*(t) < r\big] = o\left(\frac{1}{n}\right), \end{aligned}$$

where $X^*(t)$ is the aggregate return at time t for load allocation obtained from $\mathcal{P}_{\text{alt}}^{(1)}$, that is

$$X^*(t) = \sum_{i=1}^{n} X_i^*(t) = \sum_{i=1}^{n} \ell_i^*(t) \, \mathbb{1}_{\{T_i \leq t\}}.$$

From now onward, we denote the solution to $\mathcal{P}_{\text{alt}}^{(2)}$ by t^* and hence $\boldsymbol{\ell}^*(t^*)$ denotes the solution to the two-step alternative formulation in (5.4) and (A.19) which gives rise to our proposed `HCMM` scheme described next.

5.3.2 Solving the Alternative Formulation

Considering the exponential distribution for workers' run-times, we first proceed to solve $\mathcal{P}_{\text{alt}}^{(1)}$ in (5.4). The expected number of equations aggregated at the master node at time t is:

$$\mathbb{E}\big[X(t)\big] = \sum_{i=1}^{n} \mathbb{E}\big[X_i(t)\big] = \sum_{i=1}^{n} \ell_i \left(1 - e^{-\frac{\mu_i}{\ell_i}(t - a_i\,\ell_i)}\right).$$

Since there is no constraint on load allocations, $\mathcal{P}_{\text{alt}}^{(1)}$ can be decomposed to n decoupled optimization problems, i.e.,

$$\ell_i^*(t) = \arg\max_{\ell_i} \mathbb{E}\big[X_i(t)\big], \tag{5.5}$$

for all workers $i \in [n]$. The solution to (5.5) satisfies the following optimality condition:

$$\frac{\partial}{\partial \ell_i}\mathbb{E}\left[X_i(t)\right] = 1 - e^{-\frac{\mu_i}{\ell_i}(t - a_i\ell_i)}\left(\frac{\mu_i t}{\ell_i} + 1\right) = 0,$$

which yields

$$\ell_i^*(t) = \frac{t}{\lambda_i}, \tag{5.6}$$

where $\lambda_i = \Theta(1)$ is a constant independent of t and is the positive solution to the following equation:

$$e^{\mu_i\lambda_i} = e^{a_i\mu_i}(\mu_i\lambda_i + 1).$$

```
Algorithm 5.1: Heterogeneous Coded Matrix Multiplication (HCMM)
  Input: computation time parameters $(a_i, \mu_i)$ for each worker $i$ [3]
  Output: computation load assigned to each worker $i$
  Procedure HCMM
      solve $\mathcal{P}_{\text{alt}}^{(1)}$ for any feasible $t$ obtain $\ell_i^*(t) = \frac{t}{\lambda_i}$
      solve $\mathcal{P}_{\text{alt}}^{(2)}$ and obtain $t^*$
      return $\ell_i^*(t^*) = \frac{t^*}{\lambda_i}$ row vector computations for worker $i$
  end
```

One can easily check that the condition $t \geq a_i \ell_i^*(t)$ holds for all i as well. Moreover, we denote by t^* the solution to $\mathcal{P}_{\text{alt}}^{(2)}$. Now, we define the HCMM load allocation as

$$\ell_i^*(t^*) = \frac{t^*}{\lambda_i}, \tag{5.7}$$

for all workers i. In the following, we formally define the HCMM algorithm which is basically the solution to $\mathcal{P}_{\text{alt}}$.

Remark 5.6 *We note that in order to implement any load allocation scheme, each worker supposedly admits an integer number of rows as its associated computation load. However, the load allocation $\ell_i^*(t^*)$ given by* HCMM *scheme in Algorithm 5.1 is a real number for any worker i and therefore one needs to round the result before proceeding with experiments. In practical scenarios, $\ell_i^*(t^*)$ is fairly large, e.g. in the order of 100 row vectors. Therefore, the effect of rounding the load allocations shall be insignificant.*

We now provide an approximation to t^* and show it asymptotically converges to t^*. The expected aggregate return at time t for optimal loads obtained in (5.6) is

$$\mathbb{E}\left[X^*(t)\right] = \sum_{i=1}^{n} \ell_i^*(t) \left(1 - e^{-\frac{\mu_i}{\ell_i^*(t)}\left(t - a_i \ell_i^*(t)\right)}\right) = \sum_{i=1}^{n} \frac{t}{\lambda_i} \left(1 - e^{-\frac{\mu_i}{t/\lambda_i}\left(t - \frac{a_i t}{\lambda_i}\right)}\right) = ts, \tag{5.8}$$

where

$$s = \sum_{i=1}^{n} \frac{1}{\lambda_i}\left(1 - e^{-\mu_i\lambda_i(1-\frac{a_i}{\lambda_i})}\right) = \sum_{i=1}^{n} \frac{\mu_i}{1+\mu_i\lambda_i} = \Theta(n),$$

since $\mu_i = \Theta(1)$ and $\lambda_i = \Theta(1)$. Let τ^* be the solution to the following equation when solved for t:

$$\mathbb{E}\big[X^*(t)\big] = \sum_{i=1}^{n} \ell_i^*(t)\left(1 - e^{-\frac{\mu_i}{\ell_i^*(t)}(t-a_i\ell_i^*(t))}\right) = r. \tag{5.9}$$

In other words, τ^* is the time for which there are exactly r inner products – on average – aggregated at the master node, when the workers are loaded according to the loading obtained in (5.6). Using (5.6), (A.6) and (A.34), we find that

$$\begin{aligned} \tau^* &= \frac{r}{s} = \Theta(1), \\ \ell_i^*(\tau^*) &= \frac{\tau^*}{\lambda_i} = \frac{r}{s\lambda_i} = \Theta(1). \end{aligned} \tag{5.10}$$

We now present the following lemma, showing that τ^* converges to t^* for large n.

Lemma 5.1 *Let t^* be the solution to the alternative formulation $\mathcal{P}_{\text{alt}}$ in (5.4-A.19) and τ^* be the solution to (A.34). Then,*

$$\tau^* \leq t^* \leq \tau^* + o(1).$$

Proof: We defer the proof to Appendix D.3. ■

5.4 Generalization to the Shifted Weibull Model

In this section, we consider the shifted Weibull distribution for the workers' execution times, which captures a broader class of run-time models than the exponential

[3]For the shifted Weibull distribution, parameters (a_i, μ_i, α_i) are taken as inputs.

distribution. We particularly generalize our proposed HCMM algorithm to the class of shifted Weibull distributed run-times and prove Theorems 5.3 and 5.4. More specifically, we argue that asymptotic optimality of HCMM is derived similar to the shifted exponential case and further show that HCMM provides unbounded gain over uncoded schemes, asymptotically.

A random variable T has Weibull distribution with shape parameter $\alpha > 0$ and scale parameter $\mu > 0$, denoted by $T \sim \mathcal{W}(\alpha, \mu)$, if the CDF of T is of the following form:

$$\Pr[T \leq t] = 1 - e^{-(\mu t)^{\alpha}}, \quad t \geq 0.$$

The expected value of the Weibull distribution is known to be $\mathbb{E}[T] = \frac{1}{\mu}\Gamma(1+1/\alpha)$, where $\Gamma(\cdot)$ denotes the Gamma function.

As stated in Section 5.2.2, we consider a 3-parameter shifted Weibull distribution for workers' run-times defined in (5.2). The mean value of the worker i's run-times is then $\mathbb{E}[T_i] = a_i\ell_i + \frac{\ell_i}{\mu_i}\Gamma(1 + 1/\alpha_i)$. Clearly, shifted exponential distribution is a special case of the shifted Weibull model when $\alpha_i = 1$. By slight reparameterizations, this model can be similarly applied to the HCMM algorithm proposed in Algorithm 5.1, meaning that the main and alternative optimization problems defined in (5.3), (5.4) and (A.19) can be similarly analyzed under the shifted Weibull model.

As in the exponential case, we begin by maximizing the expected aggregate return at the master node ($\mathcal{P}^{(1)}_{\text{alt}}$) under the shifted Weibull distribution, which is given by

$$\mathbb{E}\left[X(t)\right] = \sum_{i=1}^{n} \mathbb{E}\left[X_i(t)\right] = \sum_{i=1}^{n} \ell_i \left(1 - e^{-\left(\frac{\mu_i}{\ell_i}(t-a_i\ell_i)\right)^{\alpha_i}}\right).$$

The optimal load allocation that maximizes the individual expected aggregate returns at each worker (and thus the total aggregate return) can be found by solving the following equation:

$$\frac{\partial}{\partial \ell_i}\mathbb{E}\left[X_i(t)\right] = 1 - e^{-\left(\frac{\mu_i}{\ell_i}(t-a_i\ell_i)\right)^{\alpha_i}}\left(1+\frac{{\mu_i}^{\alpha_i}\alpha_i t}{\ell_i}\left(\frac{t}{\ell_i}-a_i\right)^{\alpha_i-1}\right) = 0. \tag{5.11}$$

Solving (5.11) for ℓ_i yields $\ell_i^*(t) = \frac{t}{\lambda_i}$ where the constant $\lambda_i > a_i$ is the positive solution to

$$e^{{\mu_i}^{\alpha_i}(\lambda_i-a_i)^{\alpha_i}} = 1 + \alpha_i{\mu_i}^{\alpha_i}\lambda_i(\lambda_i-a_i)^{\alpha_i-1}.$$

Similar to Section 5.3, we can define s as follows,

$$\begin{aligned} s &= \frac{\mathbb{E}\left[X^*(t)\right]}{t} \\ &= \frac{1}{t}\sum_{i=1}^{n}\ell_i^*(t)\left(1-e^{-\left(\frac{\mu_i}{\ell_i^*(t)}\left(t-a_i\ell_i^*(t)\right)\right)^{\alpha_i}}\right) \\ &= \sum_{i=1}^{n}\frac{1}{\lambda_i}\left(1-e^{-\left(\mu_i\lambda_i\left(1-\frac{a_i}{\lambda_i}\right)\right)^{\alpha_i}}\right) \\ &= \sum_{i=1}^{n}\frac{\alpha_i{\mu_i}^{\alpha_i}(\lambda_i-a_i)^{\alpha_i-1}}{1+\alpha_i{\mu_i}^{\alpha_i}\lambda_i(\lambda_i-a_i)^{\alpha_i-1}} \\ &= \Theta(n). \end{aligned}$$

The last equality uses the fact that all the distribution parameters are constants. The expected aggregate return with optimal loads, $\mathbb{E}\left[X^*(t)\right]$, equals to r at time $t = \tau^*$. Thus, $\tau^* = \frac{r}{s} = \Theta(1)$ and $\ell_i^*(\tau^*) = \frac{\tau^*}{\lambda_i} = \frac{r}{s\lambda_i} = \Theta(1)$.

Proof: [Proof of Theorem 5.3] With the aforementioned reparametrizations of λ_i, s and τ^*, the `HCMM` algorithm defined in Algorithm 5.1 is identically applicable to the Weibull model. Proof of the asymptotic optimality of `HCMM` under the Weibull distribution follows the similar steps as in the proof for the exponential case in Appendix D.1 (unless specifically justified, e.g. (D.5)). We avoid rewriting these steps for the purpose of readability of the paper, but we note that the concentration inequalities used to establish the proof of Theorem 5.1 can be applied to a wide class of distributions including the Weibull distribution. ■

As an implication of Theorem 5.3, the induced expected execution time by `HCMM` algorithm is asymptotically constant, that is $\mathbb{E}[T_{\mathsf{HCMM}}] = \Theta(1)$; which was also the case for shifted exponential distribution. To compare with the uncoded scenario, we start by the following lemma which characterizes the extreme value of a sequence of Weibull random variables.

Lemma 5.2 *Let $\{T_i\}_{i=1}^{\infty}$ be a sequence of i.i.d. $\mathcal{W}(\alpha,\mu)$ random variables and $T_n^* = \max_{i\in[n]} T_i$ denote the maximum of the first n variables. Then,*

$$\mathbb{E}\left[T_n^*\right] \geq \Theta\left((\log n)^{1/\alpha}\right).$$

Proof: Consider the sequence of maximums $\{T_i^*\}_{i=1}^{\infty}$. From Markov's inequality, we have $\frac{\mathbb{E}[T_n^*]}{t_n} \geq \Pr[T_n^* \geq t_n]$, for any $t_n > 0$ and $n \in \mathbb{N}$. Pick $t_n = \frac{1}{\mu}\big(\log n\big)^{1/\alpha}$. Therefore,

$$\begin{aligned}
\frac{\mathbb{E}[T_n^*]}{\frac{1}{\mu}\big(\log n\big)^{1/\alpha}} &\geq \Pr\left[T_n^* \geq \frac{1}{\mu}\big(\log n\big)^{1/\alpha}\right] \\
&= 1 - \Pr\left[T_n^* < \frac{1}{\mu}\big(\log n\big)^{1/\alpha}\right] \\
&= 1 - \prod_{i=1}^{n} \Pr\left[T_i < \frac{1}{\mu}\big(\log n\big)^{1/\alpha}\right] \\
&= 1 - \left(1 - e^{-\log n}\right)^n \\
&= 1 - \left(1 - \frac{1}{n}\right)^n.
\end{aligned}$$

Therefore,

$$\lim_{n\to\infty} \frac{\mathbb{E}[T_n^*]}{\frac{1}{\mu}\big(\log n\big)^{1/\alpha}} \geq \lim_{n\to\infty} 1 - \left(1 - \frac{1}{n}\right)^n = 1 - \frac{1}{e} > 0.63,$$

which implies $\mathbb{E}[T_n^*] \geq \Theta\left((\log n)^{1/\alpha}\right)$. ■

Now we complete the proof of Theorem 5.4.

Proof: [Proof of Theorem 5.4] Recall that T_{UC} denotes the completion time of the optimum uncoded distributed matrix multiplication algorithm across n workers parametrized by tuples $\{(a_i, \mu_i, \alpha_i,)\}_{i=1}^{n}$. To bound the mean of T_{UC}, assume that every machine is replaced with a stochastically faster machine with parameters $(\tilde{a}, \tilde{\mu}, \tilde{\alpha})$ where $\tilde{a} = \min_i a_i$, $\tilde{\mu} = \max_i \mu_i$ and $\tilde{\alpha} = \max_i \alpha_i$, i.e., the expected run-time of the latter scenario is no greater than that of the former one. For the new set of n identical machines, the optimal loading is uniform, i.e., $\widetilde{\ell}_i^* = \frac{r}{n}$. Let $\{\widetilde{T}_i\}_{i=1}^{n}$ denote the i.i.d. shifted Weibull run times for new set of machines which have CDFs of the form

$$\Pr[\widetilde{T}_i \leq t] = 1 - e^{-\left(\frac{\tilde{\mu}}{\widetilde{\ell}_i^*}\left(t - \tilde{a}\widetilde{\ell}_i^*\right)\right)^{\tilde{\alpha}}} = 1 - e^{-\left(\tilde{\mu}\frac{n}{r}\left(t - \tilde{a}\frac{r}{n}\right)\right)^{\tilde{\alpha}}},$$

for $t \geq \frac{\tilde{a}r}{n}$. The mean of computation time for the new set of machines is

$$\mathbb{E}[\widetilde{T}_{\mathsf{UC}}] = \mathbb{E}[\max_{i\in[n]} \widetilde{T}_i] = \frac{\tilde{a}r}{n} + \mathbb{E}[\max_{i\in[n]} \widetilde{\widetilde{T}}_i],$$

where $\widetilde{\widetilde{T}}_i = \widetilde{T}_i - \frac{\tilde{a}r}{n}$ are i.i.d. $\mathcal{W}(\tilde{\alpha}, \tilde{\mu}\frac{n}{r})$ for all workers $i \in [n]$. Using Lemma 5.2, we can write

$$\mathbb{E}[T_{\mathsf{UC}}] \geq \mathbb{E}[\widetilde{T}_{\mathsf{UC}}] \geq \frac{\tilde{a}r}{n} + \Theta\left((\log n)^{1/\tilde{\alpha}}\right) = \Theta\left((\log n)^{1/\tilde{\alpha}}\right).$$

Comparing the best uncoded scheme with the proposed coded algorithm demonstrates that `HCMM` outperforms the best uncoded scheme by a factor of at least $\Theta\left((\log n)^{1/\tilde{\alpha}}\right)$, i.e.,

$$\frac{\mathbb{E}[T_{\mathsf{UC}}]}{\mathbb{E}[T_{\mathtt{HCMM}}]} \geq \Theta\left((\log n)^{1/\tilde{\alpha}}\right).$$

■

5.5 Numerical Results

In this section, we present our results both from simulations as well as from experiments over Amazon EC2 clusters. These results demonstrate how `HCMM` can provide significant speedups in comparison to state-of-the-art load allocation schemes.

5.5.1 Numerical Analysis

We now present numerical results evaluating the performance of `HCMM`. We consider both the shifted exponential model in (5.1) and the shifted Weibull model in (5.2) for run-time distributions in our simulations, assuming the unit seconds per row (s/row) for a and $1/\mu$. The underlying computation task is to compute $r = 10000$ inner products using a heterogeneous cluster of $n = 100$ workers, where different scenarios for heterogeneity are considered. For each scenario under consideration, we implement the following load allocation schemes[4]:

1. **Uniform Uncoded**: Each worker is assigned an equal number of rows, i.e., $\ell_i = r/n$ for all workers i.

2. **Load-balanced Uncoded**: Each worker is assigned a load which is inversely proportional to its expected time for computing one inner product, i.e., for the shifted exponential model, $\ell_i \propto \mu_i/(a_i\mu_i + 1)$, while for the shifted Weibull model, $\ell_i \propto \mu_i/(a_i\mu_i + \Gamma(1 + 1/\alpha_i))$ for all workers i. Furthermore, we set $\sum_{i=1}^{n} \ell_i = r$.

3. **Uniform Coded**: Equal number of coded rows are assigned to each worker. Redundancy is numerically optimized for minimizing the average computation time for receiving results of at least r inner products at the master node.

[4]For each scheme, the load number for each worker is approximated to the nearest larger integer using the `ceil()` function. For the practical large load regime considered in simulations, this rounding step has negligible impact on load allocation and on the overall results.

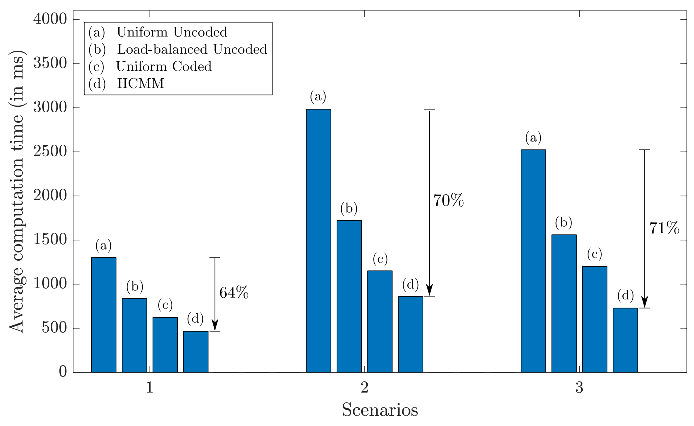


Figure 5.2: Illustration of the performance gain of `HCMM` over the three benchmark schemes for the exponential run-time model. Among the three scenarios, `HCMM` achieves a performance improvement of up to 71% over Uniform Uncoded, up to 53% over Load-balanced Uncoded, and up to 39% over Uniform Coded. Furthermore, the coding redundancy $\sum_{i=1}^{n} \ell_i/r$ for the three scenarios is in the range of $1.41 - 1.46$ for `HCMM` and in the range of $2.3 - 2.8$ for Uniform Coded. This demonstrates the efficient utilization of resources by `HCMM`.

4. `HCMM`: Each worker is assigned the asymptotically optimal load allocation derived in Section 5.3.2, i.e., $\ell_i = \tau^*/\lambda_i$ for each worker i according to (5.7) and (5.10).

For simulations under the shifted exponential model, we consider the following three scenarios:

- **Scenario 1 (2-mode heterogeneity)**: $(a_i, \mu_i) = (1, 1)$ for 50 workers, and $(a_i, \mu_i) = (4, 0.5)$ for the other 50 workers.
- **Scenario 2 (3-mode heterogeneity)**: $(a_i, \mu_i) = (1, 0.5)$ for 25 workers, $(a_i, \mu_i) = (4, 2)$ for 25 workers, and $(a_i, \mu_i) = (12, 0.25)$ for the remaining 50 workers.
- **Scenario 3 (Random heterogeneity)**: For each worker i, parameters a_i and μ_i are sampled from the sets $\{1, 4, 12\}$, $\{0.5, 2, 0.25\}$, respectively and all uniformly at random.

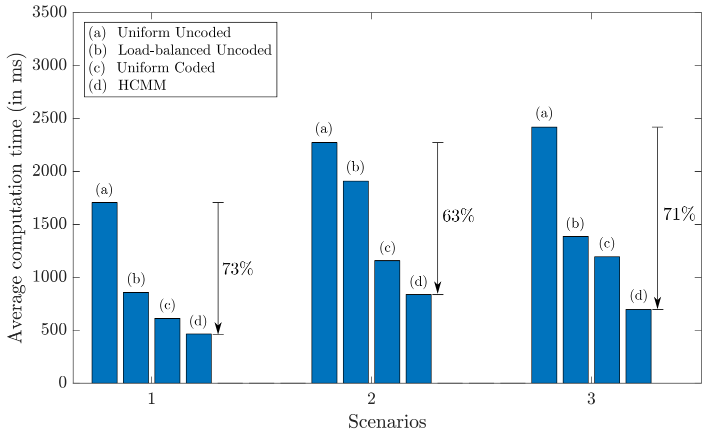


Figure 5.3: Illustration of the performance gain of HCMM over the three benchmark schemes for Weibull model for run-time. Among the three scenarios, HCMM achieves a performance improvement of up to 73% over Uniform Uncoded, up to 56% over Load-balanced Uncoded, and up to 42% over Uniform Coded. Furthermore, the coding redundancy $\sum_{i=1}^{n} \ell_i/r$ for the three scenarios is in the range of $1.30 - 1.42$ for HCMM and in the range of $2.0 - 2.5$ for Uniform Coded. This demonstrates the efficient utilization of resources by HCMM.

The following three scenarios are considered for simulations under the shifted Weibull distribution for run-times:

- **Scenario 1 (2-mode heterogeneity)**: $(a_i, \mu_i, \alpha_i) = (1, 1, 1.2)$ for 50 workers, and $(a_i, \mu_i, \alpha_i) = (4, 0.5, 0.8)$ for the other 50 workers.
- **Scenario 2 (3-mode heterogeneity)**: $(a_i, \mu_i, \alpha_i) = (1, 0.5, 0.9)$ for 25 workers, $(a_i, \mu_i, \alpha_i) = (4, 2, 1.2)$ for 25 workers, and $(a_i, \mu_i, \alpha_i) = (12, 0.25, 1.5)$ for the remaining 50 workers.
- **Scenario 3 (Random heterogeneity)**: For each worker i, parameters a_i, μ_i and α_i are sampled from the sets $\{1, 4, 12\}$, $\{0.5, 2, 0.25\}$ and $\{0.9, 1.2, 1.5\}$, respectively and all uniformly at random.

Figure 5.2 and 5.3 illustrate the performance comparison of the four schemes for the two run-time models. We make the following conclusions from the results.

- `HCMM` significantly outperforms the benchmark load allocation schemes. In particular, for the shifted exponential model, `HCMM` provides speedups of up to 71% over Uniform Uncoded, up to 53% over Load-balanced Uncoded, and up to 39% over Uniform Coded, among the three scenarios. When the machine run-time is assumed to have a shifted Weibull distribution, among the three scenarios `HCMM` results in gains of up to 73%, 56% and 42% over Uniform Uncoded, Load-balanced Uncoded, and Uniform Coded respectively.

- The coding redundancy $\sum_{i=1}^{n} \ell_i/r$ for Uniform Coded is higher in comparison to the one for `HCMM`. In particular, for simulations under the shifted exponential model, the coding redundancy for the three scenarios is in the range of $2.3 - 2.8$ for Uniform Coded and in the range of $1.41 - 1.46$ for `HCMM`. For simulations under the shifted Weibull distribution, the coding redundancy is in the range of $2.0 - 2.5$ for Uniform Coded, while for `HCMM`, it is in the range $1.30 - 1.42$. This demonstrates that `HCMM` leads to a better utilization of computing resources.

- Both Load-balanced Uncoded and Uniform Coded improve upon the performance of Uniform Uncoded. In Load-balanced Uncoded scheme, assigning larger loads to faster machines leads to better performance, while for Uniform Coded, repeated computations lead to better performance as the master does not need to wait for all the results. `HCMM` provides the best expected execution time among the four schemes as it combines the gains of Load-balanced Uncoded and Uniform Coded by employing efficient load balancing along with minimal number of redundant computations.

Next, we present the results from our experiments over Amazon EC2 clusters. These results show agreement with our numerical studies.

5.5.2 Experiments using Amazon EC2 machines

We use Python with `mpi4py` package [141] to implement our developed `HCMM` scheme over Amazon EC2 clusters. To emulate the straggler effects in large-scale systems [142], we inject artificial delays.[5] This is achieved by selecting some workers to be stragglers at the beginning of experiments and slowing down each such worker by making it wait for 3 times the amount of time it spends in computation before it sends its results to the master. This is done using the `sleep()` function in `time` package. For each scenario, the choice of stragglers is made by drawing a sample from the Bernoulli(0.5) distribution for each worker, i.e., each worker is chosen to be a straggler with probability 0.5.

In line with our simulation studies, we compare the performance of `HCMM` with the three benchmark load allocation schemes. For Load-balanced Uncoded, the number of uncoded rows ℓ_i assigned to a worker i is proportional to the number of virtual CPUs, and the loads are normalized to have a sum equal to r. For the encoding and the decoding steps for Uniform Coded as well as `HCMM`, we utilize the Luby transform (LT) codes with peeling decoder which provides nearly linear decoding complexity [144]. Utilization of LT codes for distributed computing is proposed in [145] as well. However, they perform a homogeneous load allocation by assigning an equal number of rows of the encoded data matrix to each worker and hence do not capture the heterogeneity of the computing cluster in distributing the encoded data matrix. Towards this end, we relax our goal of recovering all the inner products from any r of the coded inner products to recovering all the inner products from any $r' = r(1+\epsilon)$ coded inner products with high probability. Ideally, we would like to have $\epsilon > 0$ to be as small as possible. In our experiments, we keep $r = 10000$, and based on the results in [145], we use the robust Soliton degree distribution

[5]Artificial delays are injected since stragglers are rarely observed in small clusters in Amazon EC2. Though other emerging platforms such as federated learning, computation with deadline, mobile edge computing, fog computing, etc., still suffer from stragglers where our ideas can be employed [143].

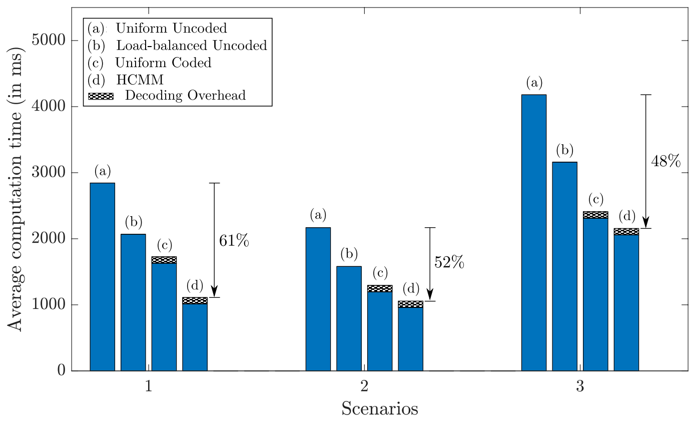


Figure 5.4: Illustration of the performance gain of HCMM over the three benchmark schemes. Among the three scenarios, HCMM achieves a performance improvement of up to 61% over Uniform Uncoded, up to 46% over Load-balanced Uncoded, and up to 36% over Uniform Coded. Furthermore, the coding redundancy $\sum_{i=1}^{n} \ell_i/r$ for the three scenarios is approximately 1.4 for HCMM and in the range of $2.12-2.26$ for Uniform Coded. Therefore, HCMM gives the best overall execution time among the four scenarios with minimal coding overhead.

with $(c, \delta) = (0.03, 0.1)$ and select $\epsilon = 0.13$, where c is a tuning parameter and δ is a bound on the probability of failure of decoding from a certain number of received coded inner products (see [145] for details). Therefore, for both HCMM and Uniform Coded, we design the load allocation such that the master needs to wait only for $r' = 11300$ coded inner products. The total computation time is equal to the waiting time for $r' = 11300$ results plus the average time for decoding the $r = 10000$ inner products from the received $r' = 11300$ coded inner products.[6] For HCMM, we use the shifted exponential distribution for estimating the computation model for each worker.

For performance comparison of the four schemes, we consider the following three computing scenarios:

- **Scenario 1**: Each row has 500000 elements. We use a heterogeneous cluster of 11 machines – one master of instance type **m4.xlarge**, four workers of instance type

[6]The average time for decoding $r = 10000$ inner products from any $r(1 + \epsilon)$ coded inner products is obtained using a **m4.xlarge** instance.

r4.2xlarge, and six workers of instance type **r4.xlarge**.

- **Scenario 2**: Each row has 500000 elements. We use a heterogeneous cluster of 16 machines – one master of instance type **m4.xlarge**, six workers of instance type **r4.2xlarge**, and nine workers of instance type **r4.xlarge**.

- **Scenario 3**: Each row has 1000000 elements. We use the same heterogeneous cluster as in the previous scenario.

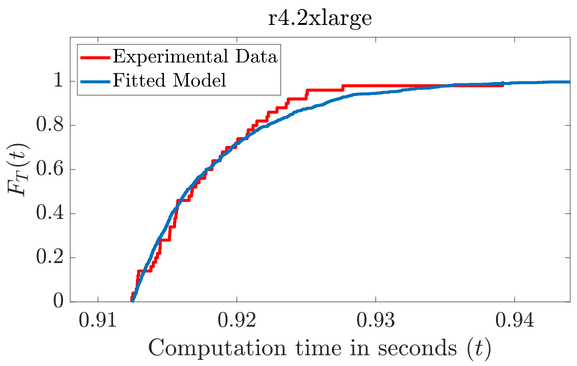
Figure 5.4 provides a performance comparison of `HCMM` with the benchmark load allocation schemes for the three scenarios, where the decoding time is taken into account as well. Figure 5.5 presents the typical cumulative distribution functions for the instances used in the experiments. We make the following conclusions from the results:

- As demonstrated in Figure 5.5, the shifted exponential model is a good first order fit for the run-times of the workers.

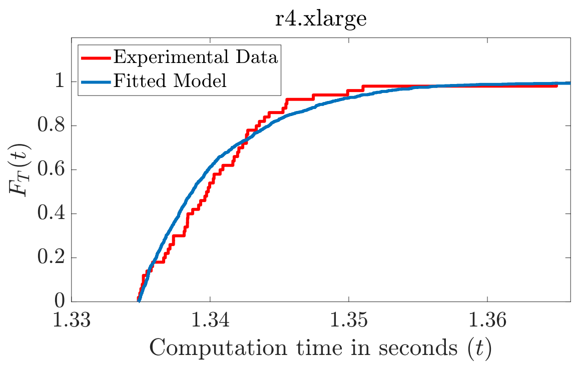
- `HCMM` achieves significant speedups over the benchmark load allocation policies. In particular, `HCMM` combined with LT codes provides gains in the overall execution time of up to 61% over Uniform Uncoded, up to 46% over Load-balanced Uncoded, and up to 36% over Uniform Coded.

- As presented in Table 5.1, `HCMM` has significantly lower total computation load compared to Uniform Coded. Hence, `HCMM` leads to efficient utilization of the computing resources, combining the benefits of both Load-balanced Uncoded and Uniform Coded schemes.

These results demonstrate that `HCMM` can provide significant speedups in large-scale computing environments.



(a) $(a, 1/\mu) = (1.37\times10^{-3}, 8.25\times10^{-6})$(s/row)



(b) $(a, 1/\mu) = (2.00\times10^{-3}, 8.72\times10^{-6})$(s/row)

Figure 5.5: Typical empirical cumulative distribution functions for two instances used in Scenario 3 of our experiments. The measurements were taken in the absence of any manual delay. As demonstrated here, shifted exponential distribution is a good model for the task execution time in EC2 machines.

Table 5.1: Total computation load ($\sum_{i=1}^{n} \ell_i$) of HCMM and Uniform Coded

Scenario	n	HCMM	Uniform Coded
1	10	11397	22600
2	15	11402	21201
3	15	11403	21201

5.6 Generalization to Computing Scenarios under Budget Constraints

In this section, we consider the optimization problem in (5.3) under the shifted exponential distribution with a monetary constraint for carrying out the overall computation. Running computation tasks on a commodity server costs depending on several factors including CPU, memory, ECU, storage, bandwidth, etc. Different cloud computing platforms employ different pricing policies, and these need to be taken into account for developing efficient task allocation and execution algorithms [146–150]. For example, Table 5.2 summarizes the cost per hour of using Amazon EC2 clusters with different parameters (at the time of writing this manuscript) [151]. In this section, we take into account

Table 5.2: Amazon EC2 Pricing for Linux

machine	vCPU	ECU	Memory (GiB)	Instance Storage (GB)	price (/Hour)
m3.medium	1	3	3.75	1×4 SSD	\$0.077
m3.large	2	6.5	7.5	1×32 SSD	\$0.154
m3.xlarge	4	13	15	2×40 SSD	\$0.308
m3.2xlarge	8	26	30	2×80 SSD	\$0.616

the monetary constraint in the optimization problem in (5.3) and provide a heuristic algorithm towards finding the optimal load allocation under cost budget constraint.

We now present the precise problem formulation we are interested in. For a computation task and a given set of N machines, the goal is to minimize the expected run-time while satisfying the budget constraint C, that is

$$\begin{aligned} \mathcal{P}_{\text{main-constrained}}: \quad & \underset{\ell}{\text{minimize}} \quad \mathbb{E}[T_{\mathsf{CMP}}] \\ & \text{subject to} \quad \sum_{i=1}^{N} c_i \mathbb{1}_{\{\ell_i > 0\}} \mathbb{E}[T_{\mathsf{CMP}}] \leq C, \end{aligned}$$

where c_i represents the cost per time unit of using machine $i \in [N]$. According to the pricing polices provided by AWS, e.g. Table 5.2, a linear model for cost (versus performance parameters) is intuitive and convincing. Considering the last two rows of Table II for instance, doubling the parameters results in doubled cost. To be general, we model the computation cost of a single machine as $c = \kappa\mu^{\gamma}$ per unit of time, which captures a convex dependency of the speed parameter μ for constants $\gamma \geq 1$ and $\kappa > 0$.

We assume that there are K types of machines parameterized with $\{(a_k, \mu_k)\}_{k=1}^{K}$, and $N_k, k \in [K]$ of each type is available to run a distributed computation task, where $N = \sum_{k=1}^{K} N_k$ is the total number of available machines. We also assume that $\mu_1 \leq \cdots \leq \mu_K$

and $a_1\mu_1 = \cdots = a_K\mu_K = \xi$ for a constant ξ.[7] As we showed in Theorem 5.1, `HCMM` is asymptotically optimal (i.e. optimal within a vanishing deviation) regarding the average run-time. In this section, we also consider the asymptotic regime, i.e. for large enough number of machines and hence `HCMM` attains the optimality per $\mathcal{P}_{\text{main}}$ in (5.3).

The following lemma states a useful observation regarding the solutions to the constrained problem $\mathcal{P}_{\text{main-constrained}}$ and the minimum possible cost for carrying out a computation task.

Lemma 5.3 `HCMM` *is the (asymptotic) solution to the feasible* $\mathcal{P}_{\text{main-constrained}}$. *Moreover, given a computation task and a set of machines, decreasing the number of fastest (slowest) machines in* `HCMM`*, results in smaller (greater) expected cost. And, the minimum (maximum) cost of* `HCMM` *is induced by running the task only on any number of the slowest (fastest) machines.*

Proof: We defer the proof to Appendix D.4. ■

Lemma 5.3 implies that if the available budget C is less than C_{min} defined in (D.15), then $\mathcal{P}_{\text{main-constrained}}$ is infeasible and it is impossible to run the task on the given set of machines while satisfying the budget constraint. Moreover, reducing one machine from the available set of fastest machines along with `HCMM` results in a lower expected cost; and reducing the number of participating slowest machines results in a larger expected cost.

Now that `HCMM` asymptotically solves the feasible budget-constrained problem in (5.12), i.e. for $C \geq C_{\text{min}}$, finding the optimal number of machines of each type to use in `HCMM` requires combinatorial search over all possible allocations. However, as Lemma 5.3 suggests, using faster machines induces a larger cost. Further, the computation time

[7]The latter assumption can be intuitively justified as follows. If a machine is c times more powerful than another machine, as the first order estimation, one can assume that both the shift (a_k) and the straggling parameter (μ_k) of the computation are c times stronger.

increases if we decrease the number of machines. This is the motivation behind our heuristic algorithm for an efficient search to find the number of machines of each type to include in HCMM, which we describe next.

Algorithm 5.2: Heuristic Search

```
Procedure Heuristic Search
    $(n_1, \cdots, n_K) \leftarrow (N_1, \cdots, N_K)$
    top
        Run HCMM with $(n_1, \cdots, n_K)$
        if cost$\big($HCMM$(n_1, \cdots, n_K)\big) > C$ then
            $n_j \leftarrow n_j - 1$ where $j = \max\{k : n_k > 0\}$
            goto top
        else
            return $(n_1, \cdots, n_K)$
        end
    end
end
```

First, Algorithm 5.2 runs HCMM algorithm using all machines, i.e., $n_k = N_k$ for each $k \in [K]$. Then, it calculates the corresponding cost according to (D.13). If the cost is larger than C, it starts to decrease the number of available fastest machines, i.e. $n_K \leftarrow n_K - 1$, and runs HCMM again. While the cost is larger than C, the algorithm keeps decreasing the number of used fast machines till $n_K = 0$. Then, the algorithm sets $n_K = 0$ and starts decreasing n_{K-1} and so on, until a feasible cost is achieved. Thus, the algorithm returns $(N_1, \cdots, N_j, n_{j+1}, 0, \cdots, 0)$ which is the first tuple that satisfies the cost constraint. Therefore, the search space complexity of the heuristic is $\mathcal{O}(N_1 + \cdots + N_K) = \mathcal{O}(N)$ which is more efficient than the exhaustive search where the complexity is $\mathcal{O}(N_1 \cdots N_K)$. The pseudo-code in Algorithm 5.2 summarizes the heuristic.

Example 5.1 *In this example, we consider two different scenarios to demonstrate the application of the proposed heuristic search algorithm. For the cost model, we assume* $\gamma = 2$ *and* $\kappa = 1$*, i.e.* $c = \mu^2$*. Further, we consider the task of computing* $r = 100$

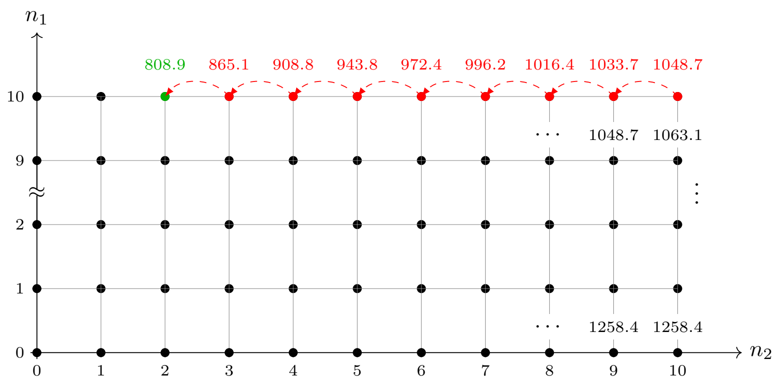


Figure 5.6: Total cost associated with every pair of (n_1, n_2); $0 \leq n_1, n_2 \leq 10$.

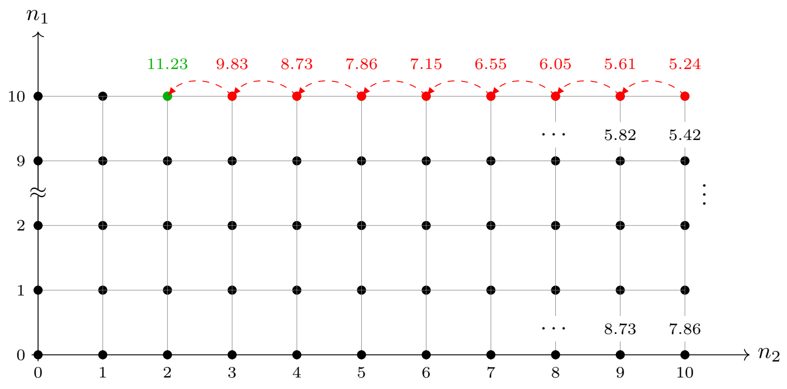


Figure 5.7: Expected time associated with every pair of (n_1, n_2); $0 \leq n_1, n_2 \leq 10$.

equations.

- ***Scenario 1:*** *Two types of machines are available parameterized by* $(a_1, \mu_1) = (0.5, 2)$ *and* $(a_2, \mu_2) = (0.25, 4)$, *assuming* 10 *machines available of each type. Further, the available budget is* $C = 860$. *Using Lemma 5.3, the minimum and maximum induced costs are* $C_{min} = 629.2$ *and* $C_{max} = 1258.4$. *As* $C \geq C_{min}$, *there exists an* HCMM *load allocation which is asymptotically optimal per (5.12). Applying the proposed heuristic search, it takes* 9 *iterations (see Fig. 5.6 and 5.7) to arrive at the load allocation* $(n_1, n_2) = (10, 2)$ *which corresponds to the expected cost* 808.9 *and average execution time* $\mathbb{E}[T_{\texttt{HCMM}}] = 11.23$.

- ***Scenario 2:*** *Three types of machines are available which are parameterized by*

$(a_1, \mu_1) = (1, 1)$, $(a_2, \mu_2) = (0.5, 2)$ and $(a_3, \mu_3) = (0.125, 8)$, assuming 10 machines of each type available. Further, the available budget is $C = 475$. Using Lemma 5.3, the minimum and maximum induced costs for the task of computing $r = 100$ equations are $C_{min} = 314.6$ and $C_{max} = 2516.8$ respectively. It takes 15 iterations for the proposed heuristic search algorithm to arrive at the tuple $(n_1, n_2, n_3) = (10, 6, 0)$. This corresponds to the expected cost 486.2 and the average time $\mathbb{E}[T_{\texttt{HCMM}}] = 14.3$.

5.7 Concluding Remarks

In this chapter, we proposed a coding framework for distributed matrix-vector multiplication in heterogeneous cloud computing environments. In particular, we considered two distributions for machines' run-times, i.e. shifted exponential and shifted Weibull and tackled the intractable problem of minimizing the average run-time of a computation task over all possible load allocations by proposing a tractable alternative formulation. The solution to the alternative problem established our proposed HCMM load allocation scheme which we proved to be asymptotically optimal. We also demonstrated the speedup of HCMM over three benchmark load allocation schemes and presented both the numerical and the experimental results. Experiments over Amazon EC2 clusters demonstrate that HCMM combined with LT codes and peeling decoders can provide significant gains in the average overall execution time. Moreover, we argued that HCMM is the asymptotically optimal allocation in budget-constrained scenarios as well, which led to providing a heuristic search in order to find a (sub)optimal load-machine assignment for a given set of machines while satisfying a pre-defined budget constraint.

Chapter 6

Robust and Efficient Gradient Aggregation in Distributed Learning

We focus on the commonly used synchronous Gradient Descent paradigm for large-scale distributed learning, for which there has been a growing interest to develop efficient and robust gradient aggregation strategies that overcome two key system bottlenecks: communication bandwidth and stragglers' delays. In particular, Ring-AllReduce (`RAR`) design has been proposed to avoid bandwidth bottleneck at any particular node by allowing each worker to only communicate with its neighbors that are arranged in a logical ring. On the other hand, Gradient Coding (`GC`) has been recently proposed to mitigate stragglers in a master-worker topology by allowing carefully designed redundant allocation of the data set to the workers. We propose a joint communication topology design and data set allocation strategy, named `CodedReduce` (`CR`), that combines the best of both `RAR` and `GC`. That is, it parallelizes the communications over a tree topology leading to efficient bandwidth utilization, and carefully designs a redundant data set allocation and coding strategy at the nodes to make the proposed gradient aggregation scheme robust to stragglers. In particular, we quantify the communication parallelization gain and resiliency of the proposed CR scheme, and prove its optimality when the communication topology is a regular tree. Moreover, we characterize the expected run-time of CR and show order-wise speedups compared to the benchmark schemes. Finally, we empirically evaluate the performance of our proposed CR design over Amazon EC2 and demonstrate that it achieves speedups of up to 27.2× and 7.0×, respectively over the benchmarks GC and RAR.

6.1 Introduction

Modern machine learning algorithms are now used in a wide variety of domains. However, training a large-scale model over a massive data set is an extremely computation and storage intensive task, e.g. training ResNet with more than 150 layers and hundreds of millions of parameters over the data set ImageNet with more than 14 million images. As a result, there has been significant interest in developing distributed learning strategies that speed up the training of learning models (e.g., [104, 152–157]).

In the commonly used Gradient Descent (GD) paradigm for learning, parallelization can be achieved by arranging the machines in a master-worker setup. Through a series of iterations, the master is responsible for updating the underlying model from the results received from the workers, where they compute the partial gradients using their local data batches and upload to the master at each iteration. For the master-worker setup, both synchronous and asynchronous methods have been developed [152–157]. In synchronous settings, all the workers wait for each other to complete the gradient computations, while in asynchronous methods, the workers continue the training process after their local gradient is computed. While synchronous approaches provide better generalization behaviors than the asynchronous ones [154, 158], they face major system bottlenecks due to (1) bandwidth congestion at the master due to concurrent communications from the workers to the master [159]; and (2) the delays caused by slow workers or stragglers that significantly increase the run-time [155].

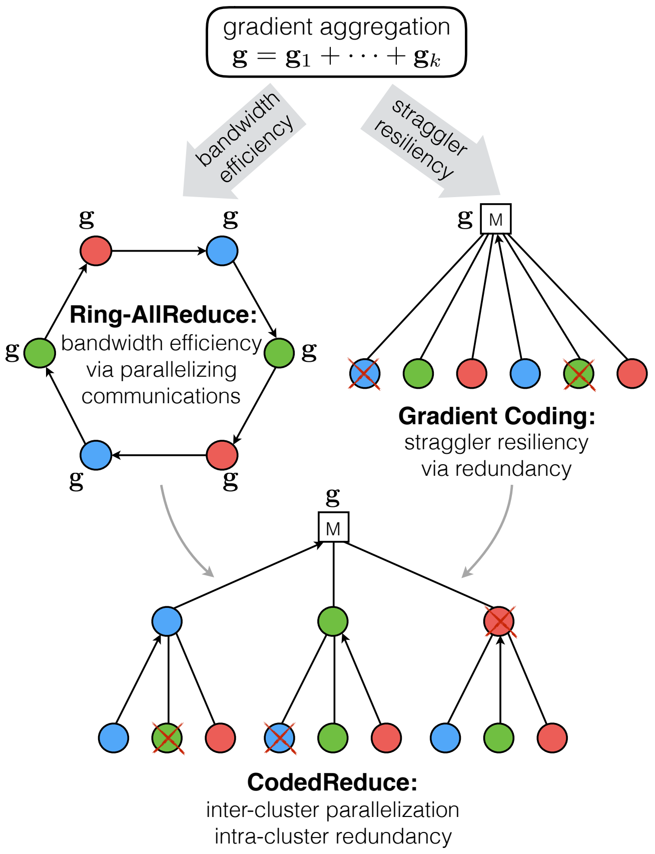


Figure 6.1: Illustration of RAR, GC and CR: In RAR, workers communicate only with their neighbors on a ring, which results in high bandwidth utilization; however, RAR is prone to stragglers. GC is robust to stragglers by doing redundant computations at workers; however, GC imposes bandwidth bottleneck at the master. CR achieves the benefits of both worlds, providing high bandwidth efficiency along with straggler resiliency.

To alleviate the communication bottleneck in distributed learning, various bandwidth efficient strategies have been proposed [160–162]. Particularly, Ring-AllReduce (RAR) [159] strategy has been proposed by allowing each worker to only communicate with its neighbors that are arranged in a logical ring. More precisely, the data set, $\mathcal{D}$, is uniformly distributed among N workers and each node combines and passes its partial gradient along the ring such that at the end of the collective operation, each worker has a copy of the full gradient $\mathbf{g}$ (Figure 6.1). Due to the master-less topology of RAR, it avoids bandwidth bottleneck at any particular node. Furthermore, as shown in [160], RAR is provably bandwidth optimal and induces $\mathcal{O}(1)$ communication overhead that does not depend on the number of distributed workers. As a result, RAR has recently become a central component in distributed deep learning for model updating [163–165]. More recent approaches to mitigate bandwidth bottleneck in distributed gradient aggregation include compression and quantization of the gradients [166–168].

Despite being bandwidth efficient, AllReduce-type algorithms are inherently sensitive to stragglers, which makes them prone to significant performance degradation and even complete failure if *any* of the workers slows down. Straggler bottleneck becomes even more significant as the cluster size increases [142, 169].

One approach to mitigate stragglers in distributed computation is to introduce computational redundancy via replication. [14] proposes to replicate the straggling task on other available nodes. In [15], the authors propose a partial data replication for robustness. Other relevant replication based strategies have been proposed in [170–172]. Recently, coding theoretic approaches have also been proposed for straggler mitigation [121, 123, 128, 134, 173–175]. Specifically, Gradient Coding (GC) [176] has been proposed to alleviate stragglers in distributed gradient aggregation in a master-worker topology (Figure 6.1). In GC, the data set $\mathcal{D}$ is carefully and redundantly distributed among the N workers where each worker computes a *coded* gradient from its local batch. The master node waits for the results of *any* $N - S$ workers and recovers the total gradient $\mathbf{g}$, where the design parameter S denotes the maximum number of stragglers that can be tolerated. Therefore, GC prevents the master from waiting for *all* the workers to finish their computations, and it was shown to achieve significant speedups over the classical uncoded master-worker setup [176].

However, as the cluser size gets large, GC suffers from significant network congestion at the master. In particular, the communication overhead increases to $\mathcal{O}(N)$, as the master needs to receive messages from $\mathcal{O}(N)$ workers. Thus, it is essential to design distributed learning strategies that alleviate stragglers while imposing low communication overhead across the cluster. Consequently, our goal in this chapter is to answer the following fundamental question:

Can we achieve the communication parallelization of `RAR` *and the straggler toleration of* `GC` *simultaneously in distributed gradient aggregation?*

We answer this question in the affirmative. As the main contribution of this chapter, we propose a joint design of data allocation and communication strategy that is robust to stragglers, alongside being bandwidth efficient. Specifically, we propose a scalable and robust scheme for synchronous distributed gradient aggregation, called `CodedReduce` (`CR`).

There are two key ideas behind `CR`. Firstly, we use a logical tree topology for communication consisting of a master node, L layers of workers, where each *parent* node has n *children* nodes (Figure 6.1). In the proposed configuration, each node communicates only with its parent node for *downloading* the updated model and *uploading* partial gradients. As in the classical master-worker setup, the root node (master) recovers the full gradient and updates the model. Except for the leaf nodes, each node receives *enough* number of *coded* partial gradients from its children, combines them with its local and partial gradient and uploads the result to its parent. This distributed communication strategy alleviates the communication bottleneck at the nodes, as multiple parents can concurrently receive from their children. Secondly, the coding strategy utilized in `CR` provides robustness to stragglers. Towards this end, we exploit ideas from `GC` and propose a data allocation and communication strategy such that *each* node needs to only wait for *any* $n - s$ of its children to return their results.

The theoretical guarantees of the proposed `CR` scheme are two-fold. First, we char-

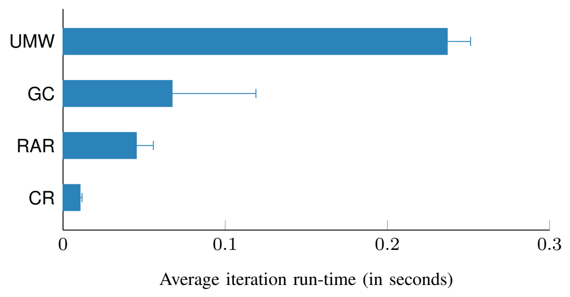


Figure 6.2: Average iteration time for gradient aggregation in different schemes `CR`, `RAR`, `GC` and `UMW`: Training a linear model is implemented on a cluster of $N = 84$ `t2.micro` instances.

acterize the computation load introduced by the proposed `CR` and prove that for a fixed straggler resiliency, `CR` achieves the optimal *computation load* (relative size of the assigned local data set to the total data set) among all the robust gradient aggregation schemes over a fixed tree topology. Moreover, `CR` significantly improves upon `GC` in the computation load of the workers. More precisely, to be robust to straggling/failure of α fraction of the children, `GC` loads each worker with $\approx \alpha$ fraction of the total data set, while `CR` assigns only $\approx \alpha^L$ fraction of the total data set, which is a major improvement. Secondly, we model the workers' computation times as shifted exponential random variables and asymptotically characterize the average latency of `CR`, that is the expected time to aggregate the gradient at the master node as the number of workers tends to infinity. This analysis further demonstrates how `CR` alleviates the bandwidth efficiency and speeds up the training process by parallelizing the communications via a tree.

In addition to provable theoretical guarantees, the proposed `CR` scheme offers substantial improvements in practice. As a representative case, Figure 6.2 provides the gradient aggregation time averaged over many gradient descent iterations implemented over Amazon EC2 clusters. Compared to three benchmarks – classical Uncoded Master-Worker (`UMW`), `GC`, `RAR` – the proposed `CR` scheme attains speedups of 22.5×, 6.4× and 4.3×, respectively.

6.2 Problem Setup and Background

In this section, we provide the problem setup followed by a brief background on `RAR` and `GC` and their corresponding straggler resiliency and communication parallelization.

6.2.1 Problem Setting

Many machine learning tasks involve fitting a model over a training data set by minimizing a loss function. For a given labeled data set $\mathcal{D} = \{\mathbf{x}_j \in \mathbb{R}^{p+1} : j = 1, \cdots, d\}$, the goal is to solve the following optimization problem:

$$\mathbf{w}^* = \underset{\mathbf{w}\in\mathbb{R}^p}{\arg\min} \sum_{\mathbf{x}\in\mathcal{D}} \ell\left(\mathbf{w};\mathbf{x}\right) + \lambda R(\mathbf{w}), \tag{6.1}$$

where $\ell(\cdot)$ and $R(\cdot)$ respectively denote the loss and regularization functions, and the optimization problem is parameterized by λ. One of the most popular ways of solving (6.1) in distributed learning is to use the Gradient Descent (GD) algorithm. More specifically, under standard convexity assumptions, the following sequence of model updates $\{\mathbf{w}^{(t)}\}_{t=0}^{\infty}$ converges to the optimal solution $\mathbf{w}^*$:

$$\mathbf{w}^{(t+1)} = h_R\left(\mathbf{w}^{(t)}, \mathbf{g}\right), \tag{6.2}$$

where $h_R(\cdot)$ is a gradient-based optimizer depending on the regularizer $R(\cdot)$ and

$$\mathbf{g} = \sum_{\mathbf{x}\in\mathcal{D}} \nabla\ell\left(\mathbf{w}^{(t)};\mathbf{x}\right), \tag{6.3}$$

denotes the gradient of the loss function evaluated at the model at iteration t over the data set $\mathcal{D}$. Under certain assumptions, the iterations in (6.2) converge to a local optimum in the non-convex case, as well. For instance, if all the saddle points of a smooth non-convex objective are strict-saddle, then the iterations in (6.2) converge to a local minimum [177]. The core component of the iterations defined in (6.2) is the computation of the gradient vector $\mathbf{g}$ at each iteration. At scale, due to limited storage and computation capacity of the computing nodes, gradient aggregation task (6.3) has to be carried out over distributed nodes. This parallelization, as we discussed earlier, introduces two major bottlenecks: stragglers and bandwidth contention. The goal of the distributed gradient aggregation scheme is to provide straggler resiliency as well as communication parallelization. At a high level, straggler resiliency, α, refers to the fraction of the straggling workers that the distributed aggregation scheme is robust to, and communication parallelization gain, β, quantifies the number of simultaneous communications in the network by distributed nodes compared to only one simultaneous communication in a single-node (master-worker) aggregation scheme.

Next, we discuss the data allocation and communication strategy of two synchronous gradient aggregation schemes in distributed learning and their corresponding straggler resiliency and communication parallelization gain.

6.2.2 Ring-AllReduce

In AllReduce-type aggregation schemes, the data set is uniformly distributed over N worker nodes $\{W_1, \cdots, W_N\}$ which coordinate among themselves in a master-less setting to aggregate their partial gradients and compute the aggregate gradient $\mathbf{g}$ at each worker. Particularly in `RAR`, each worker W_i partitions its local partial gradient into N segments $\mathbf{v}_{1,i}, \cdots, \mathbf{v}_{N,i}$. In the first round, W_i transmits $\mathbf{v}_{i,i}$ to W_{i+1}. Each worker then adds up the received segment to the corresponding segment of its local gradient, i.e., W_i obtains $\mathbf{v}_{i-1,i-1} + \mathbf{v}_{i-1,i}$. In the second round, the reduced segment is forwarded to the neighbor and added up to the corresponding segment. Proceeding similarly, at the end of $N-1$ rounds, each worker has a unique segment of the full gradient, i.e., W_i has $\mathbf{v}_{i+1,1} + \ldots + \mathbf{v}_{i+1,N}$. After the reduce-scatter phase, the workers execute the collective operation of AllGather where the full gradient $\mathbf{g}$ becomes available at each node. The `RAR` operation for a cluster of three workers is illustrated in Figure 6.3.

It is clear that `RAR` cannot tolerate *any* straggling nodes since the communications are carried out over a ring and each node requires its neighbor's result to proceed in the ring, i.e., the straggler resiliency for `RAR` is $\alpha_{\texttt{RAR}} = 0$. However, the ring communication design in `RAR` alleviates the communication congestion at busy nodes, and achieves communication parallelization gain $\beta_{\texttt{RAR}} = \Theta(N)$ which is optimal [159].

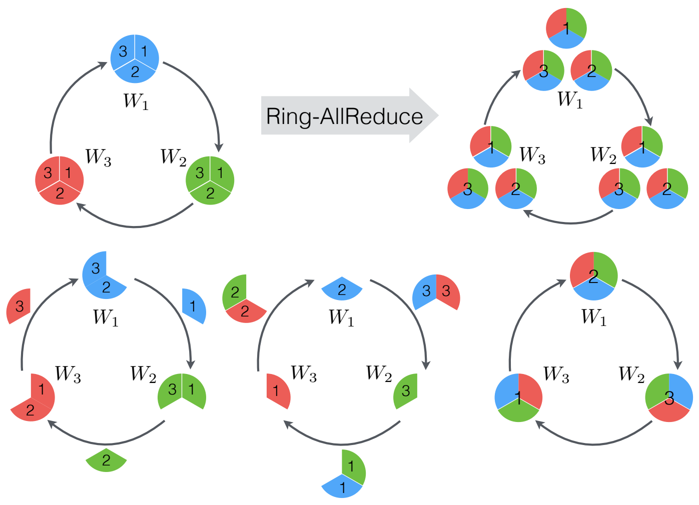


Figure 6.3: Illustration of communication strategy in `RAR` for $N = 3$ workers.

6.2.3 Gradient Coding

Gradient Coding (`GC`) [176] was recently proposed to provide straggler resiliency in a master-worker topology with one master node and N distributed worker nodes $\{W_1, \cdots, W_N\}$ as depicted in Figure 6.1. We start the description of `GC` with an illustrative example.

Example 6.1 (Gradient Coding) *To make gradient aggregation over $N = 3$ workers robust to any $S = 1$ straggler,* `GC` *partitions the data set to $\{\mathcal{D}_1, \mathcal{D}_2, \mathcal{D}_3\}$ and assigns 2 partitions to each worker as depicted in Figure 6.4. Full gradient $\mathbf{g} = \mathbf{g}_1 + \mathbf{g}_2 + \mathbf{g}_3$ can be recovered from any $N - S = 2$ workers, e.g., the master recovers $\mathbf{g}$ from W_1 and W_2 by combining their results as $\mathbf{g} = 2\left(\frac{1}{2}\mathbf{g}_1 + \mathbf{g}_2\right) - (\mathbf{g}_2 - \mathbf{g}_3)$.*

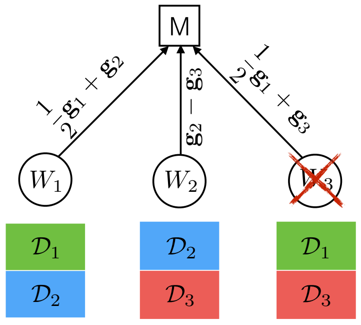

M
$\frac{1}{2}\mathbf{g}_1 + \mathbf{g}_2$
$\mathbf{g}_2 - \mathbf{g}_3$
$\frac{1}{2}\mathbf{g}_1 + \mathbf{g}_3$
W_1
W_2
W_3
$\mathcal{D}_1$
$\mathcal{D}_2$
$\mathcal{D}_2$
$\mathcal{D}_3$
$\mathcal{D}_1$
$\mathcal{D}_3$

Figure 6.4: Illustration of data allocation and communication strategy in `GC` for $N = 3$ workers.

In general, to be robust to *any* $S \in [N] = \{1, \cdots, N\}$ stragglers, `GC` uniformly partitions the data set $\mathcal{D}$ to $\{\mathcal{D}_1, \cdots, \mathcal{D}_k\}$ (e.g. $k = N$) with corresponding partial gradients $\mathbf{g}_1, \cdots, \mathbf{g}_k$ and distributes them redundantly among the workers such that each partition is placed in $S + 1$ workers, thus achieving a computation load of $r_{\texttt{GC}} = \frac{S+1}{N}$. Let matrix $\mathbf{G} = [\mathbf{g}_1, \cdots, \mathbf{g}_k]^\top \in \mathbb{R}^{k \times p}$ denote the collection of partial gradients. Each worker W_i then computes its local partial gradients and sends $\mathbf{b}_i\mathbf{G}$ to the master, where $\mathbf{B} = [\mathbf{b}_1; \cdots; \mathbf{b}_N] \in \mathbb{R}^{N \times k}$ denotes the encoding matrix, i.e. non-zero elements in $\mathbf{b}_i$ specifies the partitions stored in worker W_i. Upon receiving the results of any $N - S$ workers, the master recovers the total gradient $\mathbf{g}$ by linearly combining the received results, that is $\mathbf{g} = \mathbf{a}_f\mathbf{B}\mathbf{G}$ where the row vector $\mathbf{a}_f \in \mathbb{R}^{1 \times N}$ corresponds to a particular set of S stragglers and $\mathbf{A} = [\mathbf{a}_1; \cdots; \mathbf{a}_F]$ denotes the decoding matrix with $F = \binom{N}{S}$ distinct straggling scenarios. The `GC` algorithm designs encoding and decoding matrices $(\mathbf{B}, \mathbf{A})$ such that, in the worst case, the full gradient $\mathbf{g}$ is recoverable from the results of

Table 6.1: Communication parallelization gain and straggler resiliency of three designs RAR, GC, and CR in a system with N nodes with computation load r, where CR has a tree communication topology of L layers.

Scheme	Straggler Resiliency (α)	Communication Parallelization Gain (β)
RAR	0	$\Theta(N)$
GC	r	$\Theta(1)$
CR	$r^{1/L}$	$\Theta\left(N^{1-1/L}\right)$

any $N-S$ out of N workers, i.e. straggler resiliency $\alpha_{\texttt{GC}} = S/N$ is attained. Although GC prevents the master to wait for *all* the workers to finish their computations, it requires simultaneous communications from the workers that will cause congestion at the master node, and lead to parallelization gain $\beta_{\texttt{GC}} = \Theta(1)$ for a constant resiliency.

Having reviewed RAR and GC strategies and their resiliency and parallelization properties, we now informally provide the guarantees of our proposed CR scheme in the following remark.

Remark 6.1 CR *arranges the available N workers via a tree configuration with L layers of nodes and each parent having n children, i.e. $N = n + \cdots + n^L$. The proposed data allocation and communication strategy in* CR *results in communication parallelization gain $\beta_{\texttt{CR}} = \Theta(N^{1-1/L})$ which approaches $\beta_{\texttt{RAR}} = \Theta(N)$ for large L. Moreover, given a computation load $0 \le r \le 1$,* CR *is robust to straggling of $\alpha_{\texttt{CR}} \approx r^{1/L}$ fraction of the children per any parent in the tree, while* GC *is robust to only $\alpha_{\texttt{GC}} \approx r$ fraction of nodes and* RAR *has no straggler resiliency. Therefore,* CR *achieves the best of* RAR *and* GC*, simultaneously. Table 6.1 summarizes these results and Theorems 6.1 and 6.2 formally characterize such guarantees.*

6.3 Proposed CodedReduce Scheme

In this section, we first present our proposed CodedReduce (`CR`) scheme by describing data set allocation and communication strategy at the nodes followed by an illustrative example. Then, we provide theoretical guarantees of `CR` and conclude the section with optimality of `CR`.

6.3.1 Description of `CR` Scheme

Let us start with the proposed network configuration. `CR` arranges the communication pattern among the nodes via a *regular* tree structure as defined below. An (n, L)–regular tree graph T consists of a master node and L layers of worker nodes. At any layer (except for the lowest), each *parent* node is connected to n *children* nodes in the lower layer, i.e. there is a total of $N = n + \cdots + n^L$ nodes (See Figure 6.5). Each node of the tree is identified with a pair (l, i), where $l \in [L]$ and $i \in [n^l]$ denote the corresponding layer and the node's index in that layer, respectively. Furthermore, $T(l, i)$ denotes the sub-tree with the root node (l, i).

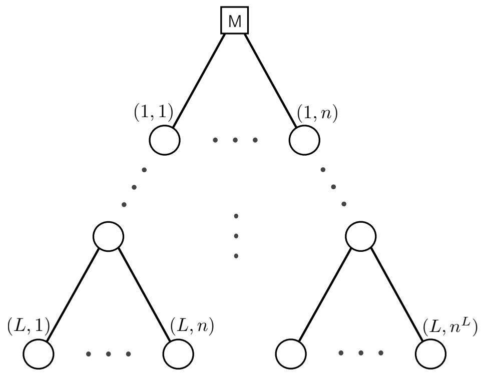


Figure 6.5: (n, L)–regular tree topology.

We next introduce a notation that eases the algorithm description. We associate a real scalar b to all the data points in a generic data set $\mathcal{D}$, denoting it by $b\mathcal{D}$, and define the gradient over $b\mathcal{D}$ as $\mathbf{g}_{b\mathcal{D}} = b\mathbf{g}_{\mathcal{D}} = b\sum_{\mathbf{x}\in\mathcal{D}} \nabla\ell(\mathbf{w}^{(t)};\mathbf{x})$. As a building block of `CR`, we define the sub-routine `CompAlloc` in which given a generic data set $\mathcal{D}$, n workers are carefully assigned with data partitions and combining coefficients such that the full gradient over $\mathcal{D}$ is retrievable from the computation results of any $n-s$ workers (Pseudo-code in Appendix E.1).

`CompAlloc`: For specified n and s, `GC` (Algorithm 2 in [176]) constructs the encoding matrix $\mathbf{B} = [\mathbf{b}_1; \cdots; \mathbf{b}_n] = [b_{i\kappa}]$. In `CompAlloc`, the input data set $\mathcal{D}$ is partitioned to $\mathcal{D} = \cup_{\kappa=1}^{k}\mathcal{D}_\kappa$ and distributed among the n workers along with the corresponding coefficients. That is, each worker $i \in [n]$ is assigned with $\mathcal{D}(i) = \cup_{\kappa=1}^{k} b_{i\kappa}\mathcal{D}_\kappa$ which specifies its local data set and corresponding combining coefficients. The parent of the n workers is then able to recover the gradient over $\mathcal{D}$, i.e. $\mathbf{g}_{\mathcal{D}}$ upon receiving the partial coded gradients of any $n-s$ workers and using the decoding matrix $\mathbf{A}$ designed by `GC` (Algorithm 1 in [176]).

`CodedReduce`: `CR` is implemented in two phases. It first allocates each worker with its local computation task via `CR.Allocate` procedure. This specifies each worker with its local data set and combining coefficients. Then, the communication strategy is determined by `CR.Execute`.

`CR.Allocate`:

(1) Starting from the master, data set $\mathcal{D}^{T(1,i)}$ is assigned to sub-tree $T(1,i)$ for $i \in [n]$ via the allocation module `CompAlloc` (Figure 6.6).

(2) In layer $l = 1$, each worker $(1,i)$, $i \in [n]$, picks $r_{\texttt{CR}}d$ data points from the corresponding sub-tree's data set $\mathcal{D}^{T(1,i)}$ as its local data set $\mathcal{D}(1,i)$ and passes the rest $\mathcal{D}_{T(1,i)} = \mathcal{D}^{T(1,i)} \setminus \mathcal{D}(1,i)$ to its children and their sub-trees (Figure 6.6).

(3) Step (1) is repeated by using the module `CompAlloc` and treating $\mathcal{D}_{T(1,i)}$ as the input data set to distribute it among the children of node $(1, i)$.

(4) Same procedure is applied till reaching the bottom layer (Figure 6.6). By doing so, the data set $\mathcal{D}$ is redundantly distributed across the tree while all the workers are equally loaded with $r_{\texttt{CR}}d$ data points, where in Theorem 6.1 we will show that $r_{\texttt{CR}}$ is a self-derived pick for `CR` given in (6.5).

`CR.Execute`:

(1) All the N nodes start their local partial *coded* gradient computations on the current model $\mathbf{w}^{(t)}$, i.e. $\mathbf{g}_{\mathcal{D}(l,i)}$ for all nodes (l, i). Note that $\mathbf{g}_{\mathcal{D}(l,i)}$ is a coded gradient (i.e. a linear combination of partial gradients) since $\mathcal{D}(l, i)$ carries combining coefficients along with its data points.

(2) Starting from the leaf nodes, they send their partial coded gradient computation results (messages) $\mathbf{m}_{(L,i)} = \mathbf{g}_{\mathcal{D}(L,i)}$ up to their parents.

(3) Upon receiving enough results from their children (any $n-s$ of them), workers in layer $L-1$ recover a linear combination of their children's messages via proper row in the decoding matrix $\mathbf{A}$, e.g., parent node $(L-1, 1)$ recovers from its children's messages $[\mathbf{m}_{(L,1)}; \cdots ; \mathbf{m}_{(L,n)}]$ via the proper decoding row $\mathbf{a}_{f(L-1,1)}$.

(4) Recovered partial gradient is added to the local partial coded gradient and is uploaded to the parent, e.g. node $(L-1, 1)$ uploads $\mathbf{m}_{(L-1,1)}$ to its parent, where

$$\mathbf{m}_{(L-1,1)} = \mathbf{a}_{f(L-1,1)}[\mathbf{m}_{(L,1)}; \cdots ; \mathbf{m}_{(L,n)}] + \mathbf{g}_{\mathcal{D}(L-1,1)}.$$

(5) The same procedure is repeated till reaching the master node which is able to aggregate the total gradient $\mathbf{g}_{\mathcal{D}}$.

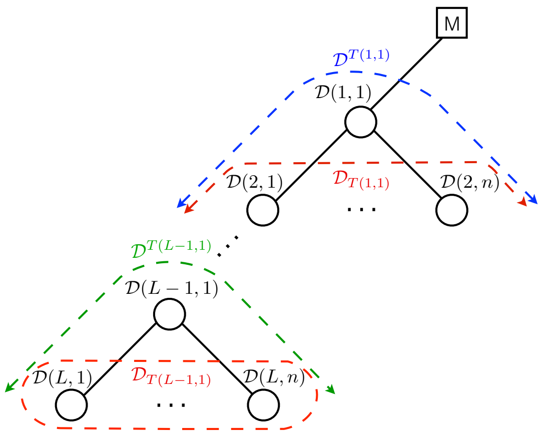


Figure 6.6: Illustration of task allocation in CR.

The pseudo-code for CR is available in Appendix E.2.

6.3.2 An Example for CR

In this section, we provide a simple example to better illustrate the proposed CR scheme.

Example 6.2 (CodedReduce) *Consider a $(3,2)$–regular tree with $N = 12$ nodes and $s = 1$ straggler per parent. From* GC*, we have the decoding and encoding matrices*

$$\mathbf{A} = \begin{pmatrix} 0 & 1 & 2 \\ 1 & 0 & 1 \\ 2 & -1 & 0 \end{pmatrix}, \quad \mathbf{B} = \begin{pmatrix} 1/2 & 1 & 0 \\ 0 & 1 & -1 \\ 1/2 & 0 & 1 \end{pmatrix}. \tag{6.4}$$

Following CR*'s description, we partition the data set of size d as $\mathcal{D} = \{\mathcal{D}_1, \mathcal{D}_2, \mathcal{D}_3\}$ and assign $\mathcal{D}^{T(1,1)} = \frac{1}{2}\mathcal{D}_1 \cup \mathcal{D}_2$ to sub-tree $T(1,1)$. Node $(1,1)$ then picks $r_{\texttt{CR}}d = \frac{4}{15}d$ data points from $\mathcal{D}^{T(1,1)}$ as $\mathcal{D}(1,1)$. To do so, $\mathcal{D}^{T(1,1)}$ is partitioned to 5 sub-sets as $\mathcal{D}^{T(1,1)} = \mathcal{D}_1^{T(1,1)} \cup \cdots \cup \mathcal{D}_5^{T(1,1)}$ and node $(1,1)$ picks the first two sub-sets, i.e. $\mathcal{D}(1,1) = \mathcal{D}_1^{T(1,1)} \cup \mathcal{D}_2^{T(1,1)}$ and the rest $\mathcal{D}_{T(1,1)} = \mathcal{D}_3^{T(1,1)} \cup \mathcal{D}_4^{T(1,1)} \cup \mathcal{D}_5^{T(1,1)}$ is passed to layer 2.*

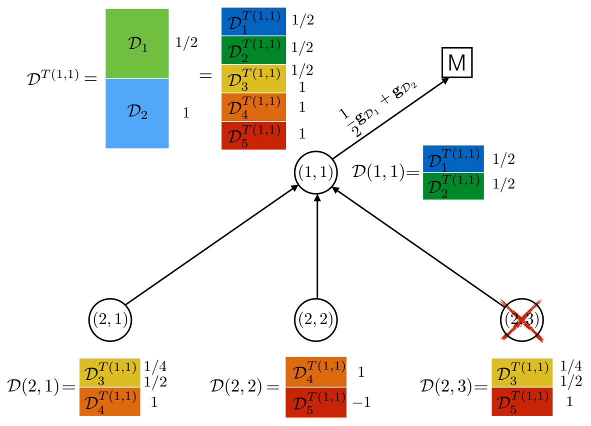

$\mathcal{D}^{T(1,1)} =$
$\mathcal{D}_1$ 1/2
$\mathcal{D}_2$ 1
$=$
$\mathcal{D}_1^{T(1,1)}$ 1/2
$\mathcal{D}_2^{T(1,1)}$ 1/2
$\mathcal{D}_3^{T(1,1)}$ 1/2
$\mathcal{D}_4^{T(1,1)}$ 1
$\mathcal{D}_5^{T(1,1)}$ 1
M
$\frac{1}{2}\mathbf{g}_{\mathcal{D}_1} + \mathbf{g}_{\mathcal{D}_2}$
(1,1)
$\mathcal{D}(1,1)=$ $\mathcal{D}_1^{T(1,1)}$ 1/2 $\mathcal{D}_2^{T(1,1)}$ 1/2
(2,1)
(2,2)
(2,3)
$\mathcal{D}(2,1)=$ $\mathcal{D}_3^{T(1,1)}$ 1/4 1/2 $\mathcal{D}_4^{T(1,1)}$ 1
$\mathcal{D}(2,2)=$ $\mathcal{D}_4^{T(1,1)}$ 1 $\mathcal{D}_5^{T(1,1)}$ −1
$\mathcal{D}(2,3)=$ $\mathcal{D}_3^{T(1,1)}$ 1/4 1/2 $\mathcal{D}_5^{T(1,1)}$ 1

Figure 6.7: Illustration of data allocation and communication strategy in CR for a $(3,2)$–regular tree.

Note that data points in $\mathcal{D}(1,1)$ carry on the linear combination coefficients associated with $\mathcal{D}^{T(1,1)} = \frac{1}{2}\mathcal{D}_1 \cup \mathcal{D}_2$. Figure 6.7 demonstrates each node in sub-tree $T(1,1)$ with its allocated data set along with the encoding coefficients. Moving to layer 2, $\mathcal{D}_{T(1,1)}$ is partitioned to 3 subsets and according to $\mathbf{B}$ in (6.4), the allocations to nodes $(2,1)$, $(2,2)$ and $(2,3)$ are as follows:

$$\mathcal{D}(2,1) = \frac{1}{2}\mathcal{D}_3^{T(1,1)} \cup \mathcal{D}_4^{T(1,1)},$$
$$\mathcal{D}(2,2) = \mathcal{D}_4^{T(1,1)} \cup (-1)\mathcal{D}_5^{T(1,1)},$$
$$\mathcal{D}(2,3) = \frac{1}{2}\mathcal{D}_3^{T(1,1)} \cup \mathcal{D}_5^{T(1,1)}.$$

Similarly for other sub-trees, each node now is allocated with a data set for which each data point is associated with a scalar. For instance, node $(2,1)$ uploads $\mathbf{m}_{(2,1)} = \mathbf{g}_{\mathcal{D}(2,1)} = \frac{1}{2}\mathbf{g}_{\mathcal{D}_3^{T(1,1)}} + \mathbf{g}_{\mathcal{D}_4^{T(1,1)}}$ to its parent $(1,1)$. Node $(1,1)$ can recover from any 2 surviving children, e.g. from $(2,1)$ and $(2,1)$ and using the first row in $\mathbf{A}$, it uploads

$$\mathbf{m}_{(1,1)} = [2,-1,0][\mathbf{m}_{(2,1)}; \mathbf{m}_{(2,2)}; \mathbf{m}_{(2,3)}] + \mathbf{g}_{\mathcal{D}(1,1)}$$

$$\begin{aligned} &= 2\mathbf{m}_{(2,1)} - \mathbf{m}_{(2,2)} + \mathbf{g}_{\mathcal{D}(1,1)} \\ &= \frac{1}{2}\mathbf{g}_{\mathcal{D}_1} + \mathbf{g}_{\mathcal{D}_2} \end{aligned}$$

to the master. Similarly for other nodes, the master can recover the full gradient from any two children, e.g. using the second row of decoding matrix $\mathbf{A}$ *and surviving children* $(1,1)$ *and* $(1,3)$*:*

$$\begin{aligned} [1,0,1][\mathbf{m}_{(1,1)};\mathbf{m}_{(1,2)};\mathbf{m}_{(1,3)}] &= \mathbf{m}_{(1,1)} + \mathbf{m}_{(1,3)} \\ &= \left(\frac{1}{2}\mathbf{g}_{\mathcal{D}_1} + \mathbf{g}_{\mathcal{D}_2}\right) + \left(\frac{1}{2}\mathbf{g}_{\mathcal{D}_1} + \mathbf{g}_{\mathcal{D}_3}\right) \\ &= \mathbf{g}_{\mathcal{D}}. \end{aligned}$$

6.3.3 Theoretical Guarantees of CR

In this section, we formally present the theoretical guarantees of CR. We first characterize the computation load induced by CR and demonstrate its significant improvement over GC. Then, we consider the commonly-used shifted exponential run-time computation distribution and a single-port communication model for workers and asymptotically characterize the expected run-time of CR and conclude with a discussion on its communication parallelization gain.

Computation Load Optimality: We show that for a fixed tree topology, the proposed CR is optimal in the sense that it achieves the minimum per-node computation load for a target resiliency. This optimality is established in two steps per Theorem 6.1: (i) we first show the achievability by characterizing the computation load of CR; and (ii) we establish a converse showing that CR's computation load is as small as possible.

Theorem 6.1 *For a fixed* (n, L)*–regular tree, any gradient aggregation scheme robust to* *any s stragglers per any parent requires computation load r where*

$$r \geq r_{\texttt{CR}} = \frac{1}{\left(\frac{n}{s+1}\right) + \cdots + \left(\frac{n}{s+1}\right)^{L}}. \tag{6.5}$$

Proof: We the proof to Appendix E.3. ■

Remark 6.2 *While* CR *is α-resilient, i.e. robust to* any *$s = \alpha n$ stragglers per* any *parent node, it significantly improves the per-node computation (and storage) load compared to an equivalent* GC *scheme with the same resiliency. In particular,* GC *loads each worker with $r_{\texttt{GC}} = \frac{S+1}{N} = \frac{\alpha N+1}{N} \approx \alpha$ fraction of the data set, while* CR *considerably reduces it to $r_{\texttt{CR}} = 1/\sum_{l=1}^{L}\left(\frac{n}{\alpha n+1}\right)^{l} \approx \alpha^{L}$. For $\alpha = 0.5$ as an instance,* CR *reduces the computation load $7\times$ by rearranging the nodes from 1 layer to 3 layers.*

Remark 6.3 CR *makes the distributed GD strategy α-resilient, that is any $s = \alpha n$ stragglers per any parent node which sums up to a total of $S = \alpha N$ stragglers – the same as the worst case number of stragglers in* GC*. It is clear than if the stragglers are picked adversarially, for instance all the nodes in layer 1, then* CR *fails to recover the total gradient at the master. However, our experiments over Amazon EC2 confirm that stragglers are randomly distributed over the tree and not adversarially picked, which is aligned with the random stragglers pattern considered in this paper.*

Total Gradient Computation Complexity: To better characterize the advantages of CR, we characterize its *total* gradient computation complexity in order to reach the final parameter model with predefined accuracy. More precisely, we focus on learning problems with strongly convex losses and let $T_{\texttt{CR}}$ denote the total number of iterations to reach a final model $\mathbf{w}$ such that $\|\mathbf{w}-\mathbf{w}^*\|^2 \leq \epsilon$. Since in each iteration of CR the *exact* gradient on all the d data samples is computed (same as in GD), therefore $T_{\texttt{CR}} = \mathcal{O}(\log(1/\epsilon))$. In each iteration, each of the N worker nodes compute $\alpha_{\texttt{CR}} \cdot d$ gradients, where according to Theorem 6.1, we have $\alpha_{\texttt{CR}} \approx \alpha^L$. All in all, in order to reach an ϵ-accurate model, the CR method requires $\mathcal{O}(\alpha^L \cdot N \cdot \log(1/\epsilon) \cdot d)$ gradient computations in total.

One simple and yet naive approach to mitigate stragglers is to update the model using the gradient computation results of *only* a fraction (α) of worker node (non-stragglers). This approach can be treated as standard Stochastic Gradient Descent (SGD) which requires $T_{\text{SGD}} = \mathcal{O}(1/\epsilon)$ iterations in total to reach an ϵ-accurate model. Since each of the N worker nodes store d/N samples (i.e. no redundant data allocation), therefore in each iteration, each node computes $\alpha d/N$ gradients. Putting all together, in order to reach ϵ-accurate model, SGD requires $\mathcal{O}(\alpha \cdot 1/\epsilon \cdot d)$ gradient computations in total. Comparing the two gradient computation complexities of CR and SGD, we observe that although SGD slashes the complexity by a *linear* factor N, however, it suffer from two *exponential* factors, that are growing α^L to α and $\log(1/\epsilon)$ to $1/\epsilon$ which significantly increase the total gradient computation complexities, as $\alpha^L \ll \alpha$ and $\log(1/\epsilon) \ll 1/\epsilon$.

Latency Performance: While we have derived the straggler resiliency of CR, the ultimate goal of a distributed gradient aggregation scheme is to have small latency which is partly attained by establishing higher communication parallelization.

Computation Time Model: We consider random computation time model for workers with shifted exponential distribution which is used in several prior works [139, 178, 179]. More precisely, for a worker W_i with assigned data set of size d_i, we model the computation time as a random variable with a shifted exponential distribution as follows:

$$\Pr[T_i \leq t] = 1 - e^{-\frac{\mu}{d_i}(t - ad_i)}, \text{ for } t \geq ad_i, \tag{6.6}$$

where system parameters $a = \Theta(1)$ and $\mu = \Theta(1)$ respectively denote the shift and the exponential rate. We assume that T_i's are independent.

Communication Time Model: To model the communication time and bandwidth bottleneck, we assume that each node is able to receive messages from only one other node at a time, and the total available bandwidth is dedicated to the communicating node. We also assume that communicating a partial gradient vector (of size p) from a child to its parent takes a constant time t_c.

The following theorem asymptotically characterizes the expected run-time of $\mathtt{CR}$ which we denote by $T_{\mathtt{CR}}$. More precisely, we consider the regime of interest where the data set size d and the number of layers L in the tree are fixed, while the number of children per parent, i.e. n is approaching infinity with a constant straggler ratio $\alpha = s/n = \Theta(1)$.

Theorem 6.2 *Considering the computation time model in (6.6) for workers, the expected run-time of* $\mathtt{CR}$ *on an* (n, L)*–regular tree with resiliency* $\alpha = \Theta(1)$ *satisfies the followings:*

$$\begin{aligned}
\mathbb{E}\left[T_{\mathtt{CR}}\right] &\geq \frac{r_{\mathtt{CR}} d}{\mu} \log\left(\frac{1}{\alpha}\right) + a r_{\mathtt{CR}} d + \big(n(1-\alpha) - o(n) + L - 1\big)\big(1 - o(1)\big)\, t_c + o(1),\\
\mathbb{E}\left[T_{\mathtt{CR}}\right] &\leq \frac{r_{\mathtt{CR}} d}{\mu} \log\left(\frac{1}{\alpha}\right) + a r_{\mathtt{CR}} d + n\big(1 - o(1)\big) L t_c + o(1)
\end{aligned} \tag{6.7}$$

Proof: We the proof to Appendix E.4. ■

Remark 6.4 *Theorem 6.2 implies that the expected run-time of the proposed* $\mathtt{CR}$ *algorithm breaks down into two terms:* $\mathbb{E}\left[T_{\mathtt{CR}}\right] = \Theta(1) + \Theta(n)$*, where the two terms* $\Theta(1)$ *and* $\Theta(n)$ *correspond to computation and communication times, respectively. As a special case, it also implies that the average run-time for* $\mathtt{GC}$ *is* $\mathbb{E}\left[T_{\mathtt{GC}}\right] = \Theta(1) + \Theta(N)$*. This clearly demonstrates that* $\mathtt{CR}$ *is indeed alleviating the bandwidth bottleneck and it improves the communication parallelization gain from* $\beta_{\mathtt{GC}} = \Theta(1)$ *to* $\beta_{\mathtt{CR}} = \Theta(N/n) = \Theta(N^{1-1/L})$ *by parallelizing the communications over an* L*-layer tree structure.*

6.4 Numerical Results

In this section, we provide the results of our experiments conducted over Amazon EC2, for which we used `Python` with `mpi4py` package. Our results demonstrate significant speedups of `CR` over baseline approaches. We consider two sets of machine learning experiments, one with a real data set, and another with an artificial data set. For each machine learning setting, we consider two cluster configurations, one with $N = 84$ workers, and another with $N = 156$ workers, using `t2.micro` instance for master and all workers. Furthermore, each experiment is run for 300 rounds. Next, we describe the experiments in detail and provide the results.

6.4.1 Convex Optimization

Real Data Set

We consider the machine learning problem of logistic regression via gradient descent (GD) over the real data set GISETTE [180]. The problem is to separate the often confused digits '9' and '4'. We use $d = 6552$ training samples, with model size $p = 5001$. The following relative error rate is considered for model estimation:

$$\text{Relative Error Rate } = \frac{\|\mathbf{w}^{(t)} - \mathbf{w}^{(t-1)}\|^2}{\|\mathbf{w}^{(t-1)}\|^2}, \tag{6.8}$$

where $\mathbf{w}^{(t)}$ denotes the estimated model at iteration t. The following schemes are considered for data allocation and gradient aggregation:

(1) Uncoded Master-worker (`UMW`): This is the naive scheme in which the data set is uniformly partitioned among the workers, and the master waits for results from all the workers to aggregate the gradient.

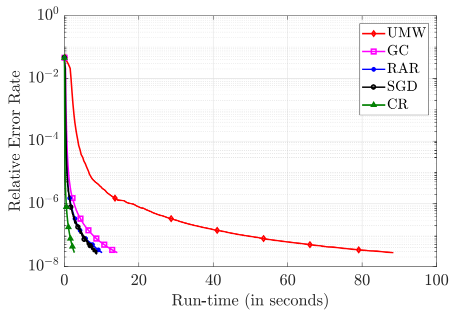


Figure 6.8: Convergence curves for relative error rate vs wall-clock time for logistic regression over $N = 84$ workers. The straggler resiliency is $\alpha = 1/4$. `CR` achieves a speedup of up to 32.8×, 5.3×, 3.8× and 3.2× respectively over `UMW`, `GC`, `RAR` and `SGD`.

(2) Gradient Coding (`GC`): We implement `GC` as described in Section 6.2.3, with the straggler parameter $S = \alpha N$.

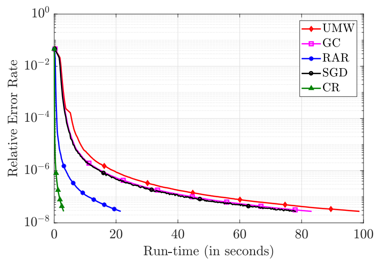
(3) Ring-AllReduce (`RAR`): The data set is uniformly partitioned over the workers and the MPI function `MPI_Allreduce()` is used for gradient aggregation.

(4) Stochastic Gradient Descent (`SGD`): The data allocation is the same as `UMW`. However, the master updates the model using the partial gradient obtained via aggregating the results from results of *only* the first $N - S$ children. Furthermore, as is typical in SGD experiments, we used a learning rate of $c_1/(t + c_2)$ where c_1 and c_2 were numerically optimized.

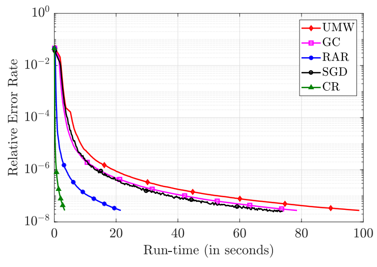
(5) `CodedReduce` (`CR`): We implement our proposed scheme as presented in Section 6.3 on a tree with $(n, L) = (12, 2)$, while the straggler parameter $s = \alpha n$.

Next, we plot the relative error rate defined in (6.8) as a function of wall-clock time for our logistic regression experiments with $N = 84$ workers and $N = 156$ workers respectively in Fig. 6.8 and Fig. 6.9. For $N = 84$, we consider a straggler-resiliency of $\alpha = 1/4$, while for $N = 156$, we consider three different values of $\alpha : 1/12, 2/12$ and $3/12$.

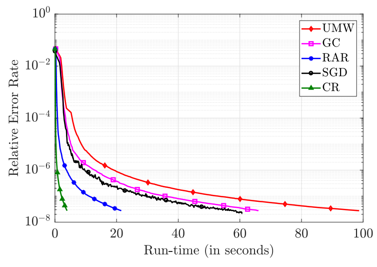
We make the following observations from the plots:



(a) $\alpha = 1/12$



(b) $\alpha = 2/12$



(c) $\alpha = 3/12$

Figure 6.9: Convergence results for relative error rate vs wall-clock time for logistic regression over $N = 156$ workers with different straggler resiliency α. (a) `CR` achieves a speed up of up to 32.3×, 27.2×, 7.0× and 25.4× respectively over `UMW`, `GC`, `RAR` and `SGD`. (b) `CR` achieves a speed up of up to 29.3×, 23.3×, 6.4× and 21.9× respectively over `UMW`, `GC`, `RAR` and `SGD`. (c) `CR` achieves a speed up of up to 25.0×, 16.8×, 5.4× and 15.4× respectively over `UMW`, `GC`, `RAR` and `SGD`.

- As demonstrated by Fig. 6.8 and 6.9, `CR` achieves significant speedups over the baseline approaches. Specifically, for $(N, \alpha) = (84, 1/4)$, `CR` is faster than `UMW`, `GC`, `RAR` and `SGD` by 32.8×, 5.3×, 3.8× and 3.2× respectively. For $(N, \alpha) = (156, 1/12)$, `CR` achieves speedups of 32.3×, 27.2×, 7.0× and 25.4× respectively over `UMW`, `GC`, `RAR` and `SGD`. Similar speedups are obtained with $(N, \alpha) = (156, 2/12)$ and $(N, \alpha) = (156, 3/12)$, as demonstrated by Fig. 6.9b and Fig. 6.9c respectively.

- Although `GC` gains over `UMW` by avoiding stragglers, its performance is still bottlenecked by bandwidth congestion, and the increase in computation load at each worker by a factor of $(S + 1)$ in comparison to `UMW`. The bottlenecks are reflected in comparison with `SGD`, which has similar or better performance in comparison to `GC` due to much less computation load per worker.

- `RAR` significantly outperforms `UMW` as well as `GC` for $N = 84$ as well as $N = 156$ worker settings. Although `RAR` achieves similar performance in comparison to `SGD` for $N = 84$ workers scenario, it ultimately beats all the schemes with the generic master-worker topology when the cluster size is increased to $N = 156$. Our proposed `CR`

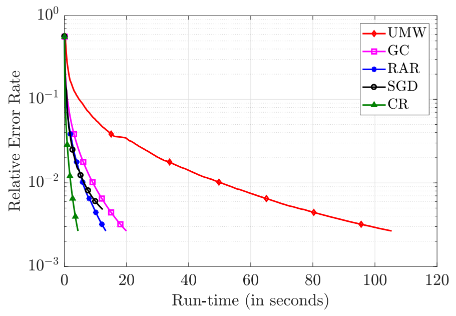


Figure 6.10: Convergence curves for normalized error rate vs wall-clock time for linear regression over $N = 84$ workers. The straggler resiliency is $\alpha = 1/4$. CR achieves a speedup of up to 24.1×, 4.6×, 3.0× and 2.8× respectively over UMW, GC, RAR and SGD.

algorithm combines the best of GC and RAR by providing straggler robustness via coding and alleviating bandwidth bottleneck via a tree topology.

Artificial Data Set

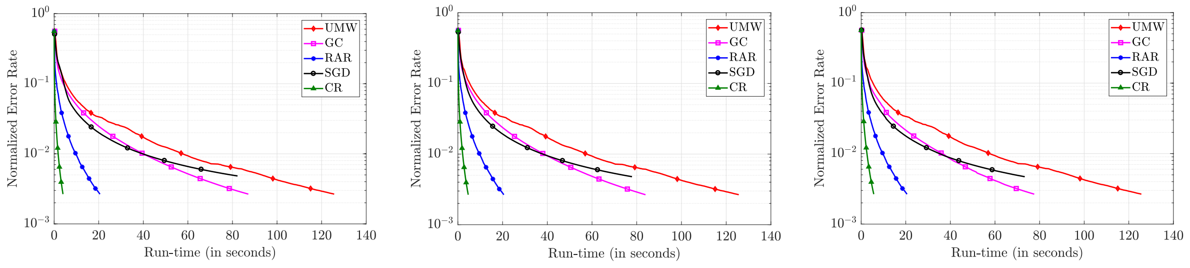
Next we solve a linear regression problem via GD over a synthetic data set with parameters $(d, p) = (7644, 6500)$. We generate the data set using the following model:

$$\mathbf{x}_j(p+1) = \mathbf{x}_j(1:p)^\top \mathbf{w}_* + z_j, \qquad \text{for } j \in [d], \tag{6.9}$$

where the true model $\mathbf{w}_*$ and features $\mathbf{x}_j(1:p) = [\mathbf{x}_j(1); \cdots; \mathbf{x}_j(p)]$ are drawn randomly from $\mathcal{N}(0, I_p)$ distribution and z_j is a standard Gaussian noise. We consider the following normalized error rate:

$$\text{Normalized Error Rate} = \frac{\|\mathbf{w}^{(t)} - \mathbf{w}_*\|^2}{\|\mathbf{w}_*\|^2}. \tag{6.10}$$

In Fig. 6.10 and 6.11, we plot the normalized error rate defined in (6.10) as a function of wall-clock time for $N = 84$ and $N = 156$ respectively. We consider similar configuration and schemes as for the experiments with real data set. The following observations are



(a) $\alpha = 1/12$ (b) $\alpha = 2/12$ (c) $\alpha = 3/12$

Figure 6.11: Convergence results for normalized error rate vs wall-clock time for linear regression over $N = 156$ workers with different straggler resiliency α. (a) `CR` achieves a speed up of up to 31.7×, 22.0×, 5.2× and 20.7× respectively over `UMW`, `GC`, `RAR` and `SGD`. (b) `CR` achieves a speed up of up to 27.1×, 18.1×, 4.4× and 16.8× respectively over `UMW`, `GC`, `RAR` and `SGD`. (c) `CR` achieves a speed up of up to 22.2×, 13.7×, 3.6× and 13.0× respectively over `UMW`, `GC`, `RAR` and `SGD`.

made with regard to the experiments:

- As in the previous case of logistic regression with real data set, `CR` achieves significant speedups over baseline approaches for linear regression as well. Particularly, for $(N, \alpha) = (84, 1/4)$, `CR` achieves speedups of 24.1×, 4.6×, 3.0× and 2.8× over `UMW`, `GC`, `RAR` and `SGD` respectively. When $(N, \alpha) = (156, 1/12)$, `CR` achieves speedups of 31.7×, 22.0×, 5.2× and 20.7× in comparison to `UMW`, `GC`, `RAR` and `SGD` respectively. Similar speedups are obtained for $(N, \alpha) = (156, 2/12)$ and $(N, \alpha) = (156, 3/12)$.

- `GC` performs better than `UMW` by avoiding stragglers. However, its performance is still bottlenecked by bandwidth congestion and the increase in computation load at each worker by a factor of $(S + 1)$ in comparison to `UMW`.

- `SGD` achieves a gain in per iteration time over `UMW` and `GC`. However, it has higher normalized error with respect to the true model.

- Combined with the results of logistic regression, our experiments complement the theoretical gains of `CR` that have been established earlier. As demonstrated by the

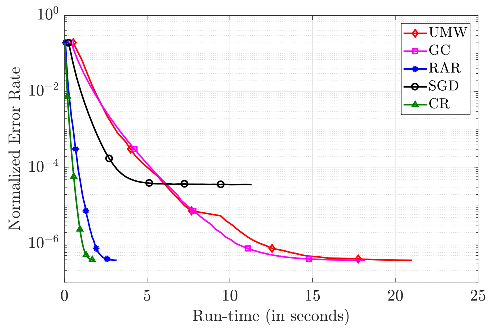


Figure 6.12: Convergence curves for normalized error rate vs wall-clock time for linear regression over $N = 156$ workers and $(d, p) = (32760, 5000)$. The straggler resiliency is $\alpha = 1/4$ and the number of rounds is 50. `CR` achieves a speedup of up to $11.3\times$, $9.7\times$, $1.69\times$ and $6.1\times$ respectively over `UMW`, `GC`, `RAR` and `SGD`.

results, a tree-based topology is well-suited for bandwidth bottleneck alleviation in large-scale commodity clusters. Furthermore, the data allocation and coding strategy provide resiliency to stragglers.

Remark 6.5 *Till now, we have considered small-scale datasets in our experiments, which is motivated by the fact that in edge based devices with non-dedicated resources, the amount of memory available for computation shall be low. Nevertheless, our proposed scheme* `CR` *can speedup general machine learning in cloud environments. To illustrate this point, we have carried out another experiment with a larger dataset* $(d, p) = (32760, 500)$, *with* $(N, \alpha) = (156, 1/4)$. *As illustrated by Fig. 6.12,* `CR` *outperforms the baseline approaches by considerable margins. Specifically,* `CR` *achieves a speedup of* $11.3\times, 9.7\times, 1.69\times$ *and* $6.1\times$ *over* `UMW`, `GC`, `RAR` *and* `SGD` *respectively.*

6.4.2 Neural Networks

We carry out simulations for evaluating the benefits of `CR` in distributed training of neural networks with cross-entropy loss, which essentially involves non-convex and non-smooth loss functions due to variety of non-linearities such as ReLUs. For this,

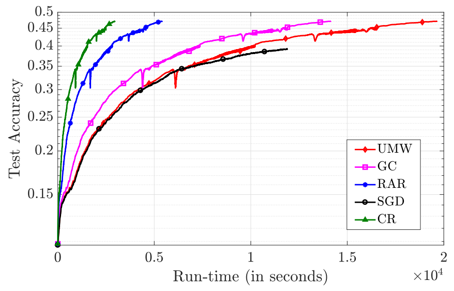


Figure 6.13: Convergence curves for test accuracy vs wall-clock time for neural network training over $N = 156$ workers. The neural network model has $p \approx 120,000$ parameters. The straggler resiliency is $\alpha = 5/12$ and the number of rounds is 2500. `CR` achieves a speedup of up to 6.6×, 4.8×, 1.8× and 4.0× respectively over `UMW`, `GC`, `RAR` and `SGD`.

we consider the CIFAR10 dataset [181], which has 10 different categories of images. CIFAR10 has 50000 images while the test dataset has 10000 images. We provide the details of the neural network in Table 6.2. We use an initial step size of 0.02, and a step decay of 0.7 at iterations 1300 and 2100. We use Glorot uniform initializer for initializing the convolutional layer weights, while for fully connected layers, we use the default initializer.

We consider a cluster of $N = 156$ servers, a resiliency of $5/12$, and $n = 12$ children per node for `CR`. We use a random subset of $d = 49920$ training images for training. Accuracy is reported on test dataset. We use the Pytorch library for neural network training. Furthermore, we use the computation and communication model as described earlier, where we assume $t_c = 0.05$ seconds, $a = 5\times10^{-5}$ seconds/data, and assume $a\mu = 1$.

In Fig. 6.13, we plot the accuracy vs wall-clock time curves for the different approaches, where training is carried out for a total of 2500 iterations. Clearly, `CR` outperforms other approaches by significant margins. Particularly, `CR` achieves a speedup of up to 6.6×, 4.8×, 1.8× and 4.0× respectively over `UMW`, `GC`, `RAR` and `SGD`.

Table 6.2: Details of the neural network architecture used in the simulations.

Sl. No.	**Parameter**	**Shape**	**Hyperparameters**
1	Conv2d	3×16×3×3	stride= 1, padding= $(1, 1)$
2	Conv2d	16×64×4×4	stride= 1, padding= $(0, 0)$
3	Linear	64×384	-
4	Linear	384×192	-
5	Linear	192×10	-

6.5 Concluding Remarks

To conclude, we discussed two critical bottlenecks in scaling up Gradient Descent-based distributed learning frameworks: communication efficiency and stragglers' delays. We proposed `CodedReduce` (`CR`), that is a joint communication topology design and data set allocation strategy. `CR` combines the best of two existing approaches–Ring-AllReduce (`RAR`) and Gradient Coding (`GC`)–by leveraging communication parallelization of `RAR` and straggler resiliency of `GC`. Theoretically, we characterized the computation load and straggler resiliency of `CR` and its asymptotic expected run-time. Lastly, we empirically demonstrated that our proposed `CR` design achieves speedups of up to 27.2× and 7.0×, respectively over the `GC` and `RAR`.

Lastly, the tree structure proposed in this paper opens up new interesting directions in order to further improve the resiliency of distributed gradient aggregation schemes. For instance, given a fix set of available worker nodes, how can one find the optimal tree (i.e. optimal depth and width) in order to minimize the expected run-time.

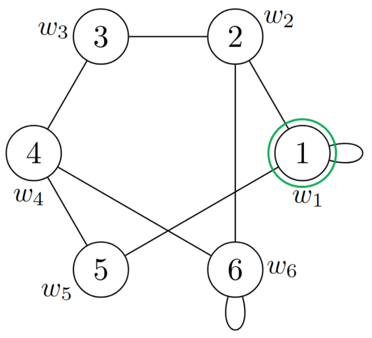
Chapter 7

Coded Computing for Distributed Graph Analytics

Many distributed computing systems have been developed recently for implementing graph based algorithms such as PageRank over large-scale graph-structured datasets such as social networks. Performance of these systems significantly suffers from *communication bottleneck* as a large number of messages are exchanged among servers at each step of the computation. Motivated by graph based MapReduce, we propose a coded computing framework that leverages computation redundancy to alleviate the communication bottleneck in distributed graph processing. As a key contribution of this work, we develop a novel *coding* scheme that systematically injects structured redundancy in the computation phase to enable *coded* multicasting opportunities during message exchange between servers, reducing the communication load substantially in large-scale graph processing. For theoretical analysis, we consider random graph models, and focus on schemes in which subgraph allocation and Reduce allocation are only dependent on vertex ID while the Shuffle design varies with graph connectivity. Specifically, we prove that our proposed scheme enables an (asymptotically) inverse-linear trade-off between *computation load* and *average communication load* for two popular random graph models – Erdös-Rényi model, and power law model. Particularly, for a given computation load r, (i.e. when each graph vertex is carefully stored at r servers), the proposed scheme slashes the average communication load by (nearly) a multiplicative factor of r. Furthermore, for the Erdös-Rényi model, we prove that our proposed scheme is optimal asymptotically as the graph size increases by providing an information-theoretic converse. To illustrate the benefits of our scheme in practice, we implement PageRank over Amazon EC2, using artificial as well as real-world datasets, demonstrating gains of up to 50.8% in comparison to the conventional PageRank implementation. Additionally, we specialize our coded scheme and extend our theoretical results to two other random graph models – random bi-partite model, and stochastic block model. Our specialized schemes asymptotically enable inverse-linear trade-offs between computation and communication loads in distributed graph processing for these popular random graph models as well. We complement the achievability results with converse bounds for both of these models.

7.1 Introduction

Graphs are widely used to identify and incorporate the relationship patterns and anomalies inherent in real-life datasets. Their growing scale and importance have prompted the development of various large-scale distributed graph processing frameworks, such as Pregel [19], PowerGraph [20] and GraphLab [21]. The underlying theme in these systems is the *think like a vertex* approach [22] where the computation at each vertex requires only the data available in the neighborhood of the vertex (see Fig. 7.1 for an illustrative example). This approach significantly improves performance in comparison to general-purpose distributed data processing systems (e.g., Dryad [182]), which do not leverage the underlying structure of graphs.



File w_1 : $\{\underbrace{\Pi_1^{\text{curr}}}_{\text{State}}, \underbrace{\mathbb{P}(1 \to 1), \mathbb{P}(1 \to 2), \mathbb{P}(1 \to 5)}_{\text{Neighborhood Parameters}}\}$

PageRank Computation :

$$\Pi_1^{\text{new}} = \sum_{j \in \mathcal{N}(j)} \underbrace{\mathbb{P}(j \to 1)\Pi_j^{\text{curr}}}_{\text{Intermediate Values}}$$

Figure 7.1: Illustrating the *think like a vertex* paradigm prevalent in common parallel graph computing frameworks. The computation associated with a vertex only depends on its neighbors. In this example, we consider the PageRank computation over a graph with six vertices. Using vertex 1 for representation, we illustrate the file and PageRank update at each vertex. File w_1 contains the state (current PageRank Π_1^{curr}) and the neighborhood parameters (probabilities of transitioning to neighbors $\{\mathbb{P}(1 \to 1), \mathbb{P}(1 \to 2), \mathbb{P}(1 \to 5)\}$). The PageRank update associated with vertex 1 is a function of only the neighborhood files (specifically, of the PageRanks of neighboring vertices and the transition probabilities from neighbors to vertex 1).

In these distributed graph processing systems, different subgraphs are stored at different servers, where a subgraph refers to the set of files associated with a subset of graph vertices. As a result of the distributed subgraph allocation, for carrying out the graph computation for a given vertex at a particular server, the intermediate values corresponding to the neighboring vertices whose files are not available at the server have to be communicated from other servers. These distributed graph processing systems, therefore, require many messages to be exchanged among servers during job execution. This results in communication bottleneck in parallel computations over graphs [23], accounting for more than 50% of the overall execution time in representative cases [24].

To alleviate the communication bottleneck in distributed graph processing, we develop a new framework that leverages *computation redundancy* by computing the intermediate values at multiple servers via *redundant subgraph allocation*. The redundancy in computation of intermediate values at multiple servers allows coded multicasting opportunities during exchange of messages between servers, thus reducing the communication load.

Our proposed framework comprises of a mathematical model for MapReduce decomposition [12] of the graph computation task. The Map computation for a vertex corresponds to computing the intermediate values for the vertices in its neighborhood, while the Reduce computation for a vertex corresponds to combining the intermediate values from the neighboring vertices to obtain the final result of graph computation. Referring to the example in Fig. 7.1, the Map and Reduce computations associated with vertex 1 are as follows:

$$\text{Map: } \Pi_1^{\text{curr}} \rightarrow \{v_{1,1}, v_{2,1}, v_{5,1}\},$$
$$\text{Reduce: } \Pi_1^{\text{new}} = v_{1,1} + v_{1,2} + v_{1,5},$$

where $v_{j,i} = \mathbb{P}(i \rightarrow j)\Pi_i^{\text{curr}}$ is the intermediate value obtained from the Map computation of vertex $i \in \mathcal{N}(j)$.

In distributed graph based MapReduce, each server is allocated a subgraph for Map computations and Reduce tasks for a subset of graph vertices, and the overall execution takes place in three phases – Map, Shuffle, and Reduce. During Map phase, each server computes the intermediate values associated with the files in the allocated subgraph. During Shuffle phase, servers communicate with each other to exchange missing intermediate values that are needed for executing the allocated Reduce tasks. Finally, each server carries out the Reduce computations allocated to it to obtain the final results, using the intermediate values obtained locally during the Map phase and the missing intermediate values obtained from other servers during the Shuffle phase. Using our mathematical model of graph based MapReduce, our framework proposes to trade redundant computations in the Map phase with communication load during the Shuffle phase. The key idea is to leverage the graph structure and create coded messages during the Shuffle phase that simultaneously satisfy the data demand of multiple computing servers in the Reduce phase.

Our work is rooted in the recent development of a coding framework that establishes an inverse-linear trade-off between computation and communication for general MapReduce computations – Coded Distributed Computing (CDC) [113]. In the MapReduce formulation considered in [113], there are n input files and the goal is to compute Q output functions, where *each* of the Q output functions depends on *all* of the n input files. In CDC, each Map computation is carefully repeated at r servers. The injected redundancy provides coded multicast opportunities in the Shuffle phase where servers exchange coded messages that are simultaneously useful for multiple servers. Each server then decodes the received messages and executes the Reduce computations assigned to it. Compared to uncoded Shuffle, where the required intermediate values are transmitted without leveraging coded multicast, CDC slashes the communication load by r. However, in contrast to graph based MapReduce considered in our framework, CDC does not incorporate the heterogeneity in the file requirements by the Reducers, as each Reducer in CDC is assumed to need intermediate values corresponding to all input files.

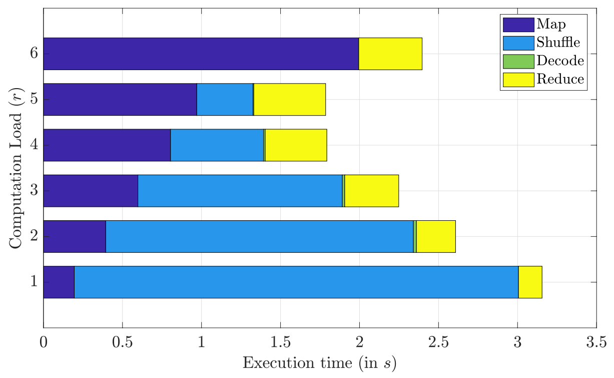


Figure 7.2: Demonstrating the impact of our proposed coded scheme in practice. We consider PageRank implementation over a real-world dataset in an Amazon EC2 cluster consisting of 6 servers. In this figure, we have illustrated the *overall execution time* as well as the times spent in different phases of execution, as a function of computation load r (details of implementation are provided in Section **??**). One can observe that the Shuffle phase is the major component of the overall execution time in conventional PageRank implementation (computation load $r = 1$), and our proposed coded scheme slashes the overall execution time by shortening the Shuffle phase (i.e., reducing the communication load) at the expense of increasing the Map phase (i.e. increasing the Map computations).

Moving from the MapReduce framework in [113] to graph based MapReduce, the key challenge is that the computation associated with each vertex highly depends on the graph structure. In particular, graph computation at each vertex requires data *only* from the neighboring vertices, while in the MapReduce framework in [113], each output computation needs all the input files (which in graph based MapReduce shall correspond to a complete graph). This asymmetry in the data requirements of the graph computations is the main challenge in developing efficient subgraph and Reduce computation allocations and Shuffling schemes for graph based MapReduce. As a key component of our proposed coding framework, we propose a coded scheme that creates coding opportunities for communicating messages across servers by Mapping the same graph vertex at different servers, so that each coded transmission satisfies the data demand of multiple servers. Within each multicast group, each server communicates a coded message which is generated using careful alignment of the intermediate values that the server needs to communicate to all the remaining members of the multicast group. Each server retrieves the missing intermediate values required for its Reduce computations using the locally available intermediate values from the Map phase and the coded messages received during the Shuffle phase.

For characterization of the performance of our proposed coding framework for distributed graph analytics, we focus on random undirected graph models. In popular graph processing frameworks such as Pregel [19], the graph partitioning for distributed processing among a set of servers is solely based on vertex ID, such as using $\mathsf{hash}(\text{ID}) \bmod K$, where K is the number of servers. Therefore, in our problem formulation, for a given computation load r and a random graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, we focus on subgraph and Reduce computation allocations $\mathcal{A}(r)$ that are based only on vertex IDs and not on graph connectivity. Here, $\mathcal{V}$ and $\mathcal{E}$ respectively denote the vertex set and edge set of $\mathcal{G}$. Although the Map and Reduce allocations are functions solely of vertex IDs, the Shuffle design needs to incorporate the graph connectivity of the graph realizations so that the communication load is minimized. This motivates us to consider the characterization of the minimum average normalized communication load $L^*(r)$, which is defined as follows:

$$L^*(r) := \inf_{A \in \mathcal{A}(r)} \mathbb{E}_{\mathcal{G}}[L_A(r, \mathcal{G})],$$

where $L_A(r, G)$ denotes the minimum normalized communication load for a realization G of $\mathcal{G}$ for a given subgraph and Reduce allocation tuple $A \in \mathcal{A}(r)$. The normalization is with respect to the total size of all the intermediate values corresponding to a fully connected graph with same number of vertices. Further details are deferred to Section 7.2, where we describe our problem formulation in detail.

For two popular random graph models, Erdös-Rényi model and power law model, we prove that our proposed coded scheme asymptotically achieves an inverse-linear trade-off between computation load in the Map phase and average normalized communication load in the Shuffle phase. Furthermore, for the Erdös-Rényi model, we develop an information-theoretic converse for the average communication load given a computation load of r. Using the asymptotic achievability result, we prove that the converse for the Erdös-Rényi model is asymptotically tight, thus proving the asymptotic optimality of our proposed coded scheme. Specifically, for a given computation load r, we show that the minimum average normalized communication load is as follows:

$$L^*(r) \approx \frac{1}{r} p \left(1 - \frac{r}{K}\right),$$

where p is the edge probability in the Erdös-Rényi model of size n, and K denotes the number of servers.

To illustrate the benefits of our proposed coded scheme in practice, we demonstrate via simulation results that even for the Erdös-Rényi model with finite n, our proposed coded scheme achieves an average communication load which is within a small gap from the information-theoretic lower bound. Furthermore, it provides a gain of (almost) r in comparison to the baseline scheme with uncoded Shuffling. Additionally, we implement the PageRank algorithm over Amazon EC2 servers using artificial as well as real-world graphs, demonstrating how our proposed coded scheme can be applied in practice. Fig. 7.2 illustrates the results of our experiments over the conventional PageRank approach ($r = 1$) for a social network webgraph Marker Cafe Dataset [183]. As demonstrated in Fig. 7.2, our proposed coded scheme achieves a speedup of up to 43.4% over the conventional PageRank implementation and a speedup of 25.5% over the single server implementation. The details of the implementation are provided in Section **??**.

We also specialize our coded scheme and extend the achievability results to two additional random graph models, random bi-partite model and stochastic block model. Specifically, we leverage the community structure in these models to adapt our proposed scheme to these models. In the random bi-partite model, we observe that there are no intra-cluster edges, due to which intermediate values for a particular Reducer in one cluster only comes from Mappers in the other cluster. Therefore, we specialize our proposed coded scheme from Section 7.4 for the random bi-partite model, partitioning the available servers in proportion to the cluster sizes, so that there is maximum overlap between Reducers corresponding to vertices in one cluster and Mappers corresponding to vertices in other cluster. Similarly, for the stochastic block model, we specialize our proposed coded scheme based on the observation that Reducers corresponding to vertices in one cluster depend on the Mappers corresponding to the vertices within the cluster with one probability (due to intra-cluster edges), and on the vertices in the other cluster with another probability (due to cross-cluster edges).

For both the random bi-partite model and the stochastic block model, we provide converse bounds. For the random bi-partite model, we remove vertices (and the edges corresponding to them) from the larger cluster so that the reduced graph has two clusters of equal sizes. The reduced graph model thus has two sets of Mappers and Reducers, which correspond to two different Erdös-Rényi models. Applying our converse bound for the Erdös-Rényi model, we arrive at the converse of the random bi-partite model. For the stochastic block model converse, the key idea is to randomly remove edges from the graph such that a larger Erdös-Rényi graph is obtained, then utilize a coupling argument, and finally use our information theoretic converse bound for the Erdös-Rényi model. Therefore, the modified coded schemes for these models demonstrate that inverse-linear trade-offs between computation and communication loads in distributed graph processing exists for these graph models as well.

Related Work. A number of coding theoretic strategies have been recently proposed to mitigate the bottlenecks in large scale distributed computing [113, 173]. Several generalizations to the Coded Distributed Computing (CDC) technique proposed in [113] have been developed. The authors in [184] extend CDC to wireless scenarios. The work in [185] extends CDC to multistage dataflows. An alternative trade-off between communication and distributed computation has been explored in [120] for MapReduce framework under predetermined storage constraints. Coding using resolvable designs has been proposed in [186]. [118] extends CDC to heterogeneous computing environments. The work in [187] proposes coding scheme for reducing communication load for computations associated with linear aggregation of intermediate results in the final Reduce stage. The key difference between our framework and each of these works is that general MapReduce computations over graphs have heterogeneity in the data requirements for the Reduce functions associated with the vertices. Other notable works that deal with communication bottleneck in distributed computation include [119, 188, 189], where the authors propose techniques to reduce communication load in data shuffling in distributed learning.

Notation. We denote by $[n]$ the set $\{1, 2, \ldots, n\}$ for $n \in \mathbb{N}$. For non-negative functions f and g of n, we denote $f = \Theta(g)$ if there are positive constants c_1, c_2 and $n_0 \in \mathbb{N}$ such that $c_1 \leq f(n)/g(n) \leq c_2$ for every $n \geq n_0$, and $f = o(g)$ if $f(n)/g(n)$ converges to 0 as n goes to infinity. We define $f = \omega(g)$, if for any positive constant c, there exists a constant $n_0 \in \mathbb{N}$ such that $f(n) > c \cdot g(n)$ for every $n \geq n_0$. To ease the notation, we let $2 \times Bern(p)$ denote a random variable that takes on the value 2 w.p. p and 0 otherwise.

7.2 Problem Setting

We now describe the setting and formulate our distributed graph analytics problem. In particular, we specify our computation model, distributed implementation model and our problem formulation based on random graphs.

7.2.1 Computation Model

We consider an undirected graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ where $\mathcal{V} = [n]$ and $\mathcal{E} = \{(i, j) : i, j \in \mathcal{V}\}$ denote the set of graph vertices and the set of edges respectively. A binary file $w_i \in \mathbb{F}_{2^F}$ of size $F \in \mathbb{N}$ containing vertex state and neighborhood parameters is associated with each graph vertex $i \in \mathcal{V}$. We denote by $\mathcal{W} = \{w_i : i \in \mathcal{V}\}$ the set of files associated with all vertices in the graph. The neighborhood of vertex i is denoted by $\mathcal{N}(i) = \{j \in \mathcal{V} : (j, i) \in \mathcal{E}\}$ and the set of files in the neighborhood of i is represented by $\mathcal{W}_{\mathcal{N}(i)} = \{w_j : j \in \mathcal{N}(i)\}$. In general, $\mathcal{G}$ can have self-loops, i.e., vertex i can be contained in $\mathcal{N}(i)$. Furthermore, a graph computation is associated with each vertex $i \in \mathcal{V}$ as follows:

$$\phi_i : \mathbb{F}_{2^F}^{|\mathcal{N}(i)|} \to \mathbb{F}_{2^B},$$

where $\phi_i(\cdot)$ is a function that maps the input files in $\mathcal{W}_{\mathcal{N}(i)}$ to a length B binary stream $o_i = \phi_i(\mathcal{W}_{\mathcal{N}(i)})$.

The computation $\phi_i(\cdot)$ can be represented as a MapReduce computation:

$$\phi_i(\mathcal{W}_{\mathcal{N}(i)}) = h_i(\{g_{i,j}(w_j) : w_j \in \mathcal{W}_{\mathcal{N}(i)}\}), \tag{7.1}$$

where the Map function $g_{i,j} : \mathbb{F}_{2^F} \to \mathbb{F}_{2^T}$ Maps file w_j to a length T binary intermediate value $v_{i,j} = g_{i,j}(w_j)$, $\forall i \in \mathcal{N}(j)$. The Reduce function $h_i : \mathbb{F}_{2^T}^{|\mathcal{N}(i)|} \to \mathbb{F}_{2^B}$ Reduces the intermediate values associated with the output function $\phi_i(\cdot)$ into the final output value $o_i = h_i(\{v_{i,j} : j \in \mathcal{N}(i)\})$.

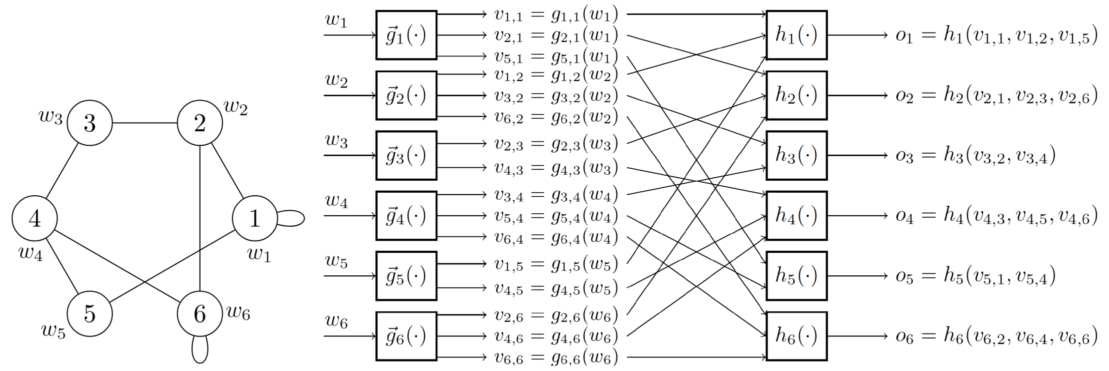
We illustrate our computation model using the graph presented in the previous section. Figure 7.3(a) (left) illustrates the graph with $n = 6$ vertices, where each vertex is associated with a file, while Figure 7.3(a) (right) illustrates the corresponding MapReduce computations.

Common graph based algorithms can be expressed in the MapReduce computation framework described above [190]. For brevity, we present two popular graph algorithms and describe how they can be expressed in the proposed MapReduce computation framework.

Example 7.1 (PageRank [191, 192]) *PageRank is a popular algorithm to measure the importance of the vertices in a webgraph based on the underlying hyperlink structure. In particular, the algorithm computes the likelihood that a random surfer would visit a page. Mathematically, the rank of a vertex i satisfies the following relation:*

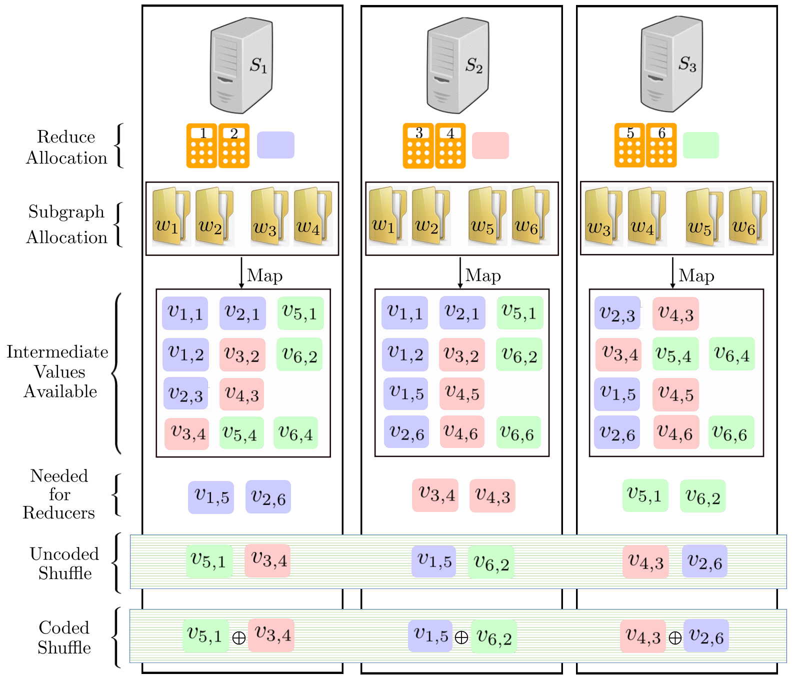
$$\Pi(i) = (1-d) \sum_{j \in \mathcal{N}(i)} \Pi(j) \Pr(j \to i) + d\frac{1}{|\mathcal{V}|},$$

where $(1-d)$ is referred to as the damping factor, $\Pi(i)$ denotes the likelihood that the ran-



w_1 w_2 w_3 w_4 w_5 w_6
1 2 3 4 5 6
$\vec{g}_1(\cdot)$ $\vec{g}_2(\cdot)$ $\vec{g}_3(\cdot)$ $\vec{g}_4(\cdot)$ $\vec{g}_5(\cdot)$ $\vec{g}_6(\cdot)$
$v_{1,1} = g_{1,1}(w_1)$
$v_{2,1} = g_{2,1}(w_1)$
$v_{5,1} = g_{5,1}(w_1)$
$v_{1,2} = g_{1,2}(w_2)$
$v_{3,2} = g_{3,2}(w_2)$
$v_{6,2} = g_{6,2}(w_2)$
$v_{2,3} = g_{2,3}(w_3)$
$v_{4,3} = g_{4,3}(w_3)$
$v_{3,4} = g_{3,4}(w_4)$
$v_{5,4} = g_{5,4}(w_4)$
$v_{6,4} = g_{6,4}(w_4)$
$v_{1,5} = g_{1,5}(w_5)$
$v_{4,5} = g_{4,5}(w_5)$
$v_{2,6} = g_{2,6}(w_6)$
$v_{4,6} = g_{4,6}(w_6)$
$v_{6,6} = g_{6,6}(w_6)$
$h_1(\cdot)$ $h_2(\cdot)$ $h_3(\cdot)$ $h_4(\cdot)$ $h_5(\cdot)$ $h_6(\cdot)$
$o_1 = h_1(v_{1,1}, v_{1,2}, v_{1,5})$
$o_2 = h_2(v_{2,1}, v_{2,3}, v_{2,6})$
$o_3 = h_3(v_{3,2}, v_{3,4})$
$o_4 = h_4(v_{4,3}, v_{4,5}, v_{4,6})$
$o_5 = h_5(v_{5,1}, v_{5,4})$
$o_6 = h_6(v_{6,2}, v_{6,4}, v_{6,6})$

(a) An example of a graph with 6 vertices, each of which has a file associated with it that contains its state and neighborhood parameters (left). MapReduce decomposition of the graph computations for the left graph (right).



S_1 S_2 S_3
Reduce Allocation
1 2 3 4 5 6
Subgraph Allocation
w_1 w_2 w_3 w_4
w_1 w_2 w_5 w_6
w_3 w_4 w_5 w_6
Map
Intermediate Values Available
$v_{1,1}$ $v_{2,1}$ $v_{5,1}$ $v_{1,2}$ $v_{3,2}$ $v_{6,2}$ $v_{2,3}$ $v_{4,3}$ $v_{3,4}$ $v_{5,4}$ $v_{6,4}$
$v_{1,1}$ $v_{2,1}$ $v_{5,1}$ $v_{1,2}$ $v_{3,2}$ $v_{6,2}$ $v_{1,5}$ $v_{4,5}$ $v_{2,6}$ $v_{4,6}$ $v_{6,6}$
$v_{2,3}$ $v_{4,3}$ $v_{3,4}$ $v_{5,4}$ $v_{6,4}$ $v_{1,5}$ $v_{4,5}$ $v_{2,6}$ $v_{4,6}$ $v_{6,6}$
Needed for Reducers
$v_{1,5}$ $v_{2,6}$
$v_{3,4}$ $v_{4,3}$
$v_{5,1}$ $v_{6,2}$
Uncoded Shuffle
$v_{5,1}$ $v_{3,4}$ $v_{1,5}$ $v_{6,2}$ $v_{4,3}$ $v_{2,6}$
Coded Shuffle
$v_{5,1} \oplus v_{3,4}$ $v_{1,5} \oplus v_{6,2}$ $v_{4,3} \oplus v_{2,6}$

(b) Illustration of subgraph and Reduce allocations for the above graph with computation load $r = 2$ and $K = 3$ servers. Each server is allocated a subgraph of size 4 and 2 Reducers. After the Map phase, each server needs to obtain the missing intermediate values that are needed to compute the Reduce functions allocated to it. Due to redundant subgraph allocation, each of the intermediate values missing at a server is available at both other servers. We illustrate two Shuffling schemes. In the uncoded Shuffle, a missing intermediate value is obtained from one of the other two servers, and each server is assigned the task of sending two intermediate values, one for each of the other two servers. In coded Shuffle, each server sends a XOR of the assigned intermediate values and sends *only one* coded message which is simultaneously useful for the both other servers.

Figure 7.3: An illustrative example.

dom surfer will arrive at vertex i, $|\mathcal{V}|$ is the total number of vertices in the webgraph, and $\Pr(j \to i)$ is the transition probability from vertex j to vertex i. The graph computation can be carried out iteratively as follows:

$$\Pi^k(i) = (1-d) \sum_{j \in \mathcal{N}(i)} \Pi^{k-1}(j) \Pr(j \to i) + d \frac{1}{|\mathcal{V}|},$$

where k and $k-1$ are respectively the current and previous iterations and $\Pi^0(i) = \frac{1}{|\mathcal{V}|}$ for all $i \in \mathcal{V}$ and $k = 1, 2, \cdots$. The number of iterations depends on the stopping criterion for the algorithm. Usually, the algorithm is stopped when the change in the PageRank mass of each vertex is less than a pre-defined tolerance. The rank update at each vertex can be decomposed into Map and Reduce functions for each iteration k. For a given vertex i and iteration k, let $w_i^k = \{\Pi^{k-1}(i)\} \cup \{\mathbb{P}(i \to j) : j \in \mathcal{N}(i)\}$, and $\phi_i^k(\mathcal{W}_{\mathcal{N}(i)}^k) = (1-d) \sum_{j \in \mathcal{N}(i)} \Pi^{k-1}(j) \Pr(j \to i) + d\frac{1}{|\mathcal{V}|}$. The Mapper $g_{i,j}(\cdot)$ maps file w_j^k to the intermediate values $v_{i,j}^k = g_{i,j}(w_j^k) = \Pi^{k-1}(j) \Pr(j \to i)$ for all neighboring vertices $i \in \mathcal{N}(j)$. Using the intermediate values from the Map computations, the Reducer $h_i(\cdot)$ computes vertex i's updated rank as $\Pi^k(i) = h_i\big(\{v_{i,j}^k : j \in \mathcal{N}(i)\}\big) = (1-d)\sum_{j \in \mathcal{N}(i)} v_{i,j}^k + d\frac{1}{|\mathcal{V}|}$.

Example 7.2 (Shortest path) *Single-source shortest path is one of the most studied problems in graph theory. The task here is to find the shortest path to each vertex i in the graph from a source vertex s. A sub-problem for this task is to compute the distance of each vertex i from the source vertex s, where distance $D(i)$ is the length of the shortest path from s to i. This can be carried out iteratively in parallel. First, initialize $D^0(s) = 0$ and $D^0(i) = +\infty, \forall i \in \mathcal{V} \setminus \{s\}$. Subsequently, each vertex i is updated as follows at each iteration k:*

$$D^k(i) = \min_{j \in \mathcal{N}(i)} \left\{ D^{k-1}(j) + t(j,i) \right\},$$

where $t(j,i)$ is the weight of the edge (j,i). The algorithm is stopped when the change in *the distance value for each vertex is within a pre-defined tolerance. The distance computation for each vertex at iteration k can be decomposed into Map and Reduce computations. Particularly, for each vertex i and iteration k, let $w_i^k = \{D^{k-1}(i)\} \cup \{t(i,j) : j \in \mathcal{N}(i)\}$, and $\phi_i^k(\mathcal{W}_{\mathcal{N}(i)}^k) = \min_{j\in\mathcal{N}(i)}(D^{k-1}(j) + t(j,i))$. The Mapper $g_{i,j}(\cdot)$ Maps the file w_j^k to the intermediate values $v_{i,j}^k = g_{i,j}(w_j^k) = D^{k-1}(j) + t(j,i)$ for all neighboring vertices $i \in \mathcal{N}(j)$. Using the intermediate values from the Map computations, the Reducer $h_i(\cdot)$ computes i's updated distance value as $D^k(i) = h_i\big(\{v_{i,j}^k : j \in \mathcal{N}(i)\}\big) = \min_{j\in\mathcal{N}(i)} v_{i,j}^k$.*

7.2.2 Distributed Implementation

For distributing the graph processing task, we consider K servers that are connected through a shared multicast network. Furthermore, at any given time, only one server can multicast over the shared network. Additionally, we assume that a multicast takes the same amount of time as a unicast. As described next, in order to distribute the Map computation tasks among the servers, each server is allocated a subgraph which is comprised of a subset of graph vertices and associated files that contain state and neighborhood information of vertices.

Subgraph Allocation: Each server is assigned the Map computations in (7.1) associated with a subgraph, which consists of a subset of vertices and associated files containing state and neighborhood information of the vertices. We denote the subgraph that is allocated to each server $k \in [K]$ by $\mathcal{M}_k \subseteq \mathcal{V}$. Thus, server k will then store all the files in $\mathcal{M}_k$, and will be responsible for computing the Map functions on those files. Note that each file should be Mapped by at least one server. Additionally, we allow redundant computations, i.e., each file can be Mapped by more than one server. The key idea in leveraging redundancy in the Map computation phase is to trade the computational resources in order to reduce the communication load in the Shuffle phase.

We define the computation load as follows.

Definition 7.1 (Computation Load) *For a subgraph allocation, $(\mathcal{M}_1, \cdots, \mathcal{M}_K)$, the computation load, $r \in [K]$, is defined as*

$$r := \frac{\sum_{k=1}^{K} |\mathcal{M}_k|}{n},$$

where $|\mathcal{M}_k|$ denotes the number of vertices in the subgraph $\mathcal{M}_k$ for $k \in [K]$.

Remark 7.1 *For a desired computation load r, each server is assigned a subgraph with the same number of vertices, i.e. for each server $k \in [K]$, $|\mathcal{M}_k| = \frac{rn}{K}$.*

To carry out the Reduce computation in (7.1) for all vertices, each server is assigned a subset of Reduce functions as follows.

Reduce Allocation: A Reducer is associated with each vertex of the graph $\mathcal{G}$ as represented in (7.1). We use $\mathcal{R}_k \subseteq \mathcal{V}$ to denote the set of vertices whose Reduce computations are assigned to server $k \in [K]$. The set of Reduce computations are partitioned into K equal parts and each part is associated exclusively with one server, i.e., $\cup_{k=1}^{K} \mathcal{R}_k = \mathcal{V}$ and $\mathcal{R}_m \cap \mathcal{R}_n = \phi$ for $m, n \in [K], m \neq n$. Therefore, $|\mathcal{R}_k| = \frac{n}{K}$, $\forall k \in [K]$.

For the graph in Figure 7.3(a) (left) and a computation load of $r = 2$, we illustrate a scheme for subgraph allocation and Reduce allocation in Figure 7.3(b). Here, each vertex appears in exactly two subgraphs, i.e. Map computation associated with each vertex is assigned to exactly two servers. The subgraph and Reduce allocations in Figure 7.3(b) form key components of our proposed scheme in Section 7.4, in which for a computation load of r, every unique set of r servers is allocated a unique batch of $n/\binom{K}{r}$ files for Map computations.

For a given scheme with subgraph allocation and Reduce allocation tuple denoted by $A = (\mathcal{M}, \mathcal{R})$, where $\mathcal{M} = (\mathcal{M}_1, \cdots, \mathcal{M}_K)$ and $\mathcal{R} = (\mathcal{R}_1, \cdots, \mathcal{R}_K)$, the distributed graph processing proceeds in three phases as described next.

Map phase: Each server first Maps the files associated with the subgraph that is allocated to it. More specifically, for each $i \in \mathcal{M}_k$, server k computes a vector of intermediate values corresponding to the vertices in $\mathcal{N}(i)$ that is $\vec{g}_i = (v_{j,i} : j \in \mathcal{N}(i))$. For the running example, we illustrate the intermediate values generated at each server during the Map phase in Figure 7.3(b), where the color of an intermediate value denotes the server that is allocated the task to execute the corresponding Reducer.

Shuffle phase: To be able to do the final Reduce computations, each server needs the intermediate values corresponding to the neighbors of each vertex that it is responsible for its Reduction. Servers exchange messages so that at the end of the Shuffle phase, each server is able to recover its required set of intermediate values. More formally, the Shuffle phase proceeds as follows. For each $k \in [K]$,

(i) server k creates a message $X_k \in \mathbb{F}_{2^{c_k}}$ as a function of intermediate values computed locally at that server during the Map phase, i.e. $X_k = \psi_k(\{\vec{g}_i : i \in \mathcal{M}_k\})$, where c_k is the length of the binary message X_k,

(ii) server k multicasts X_k to all the remaining servers,

(iii) server k recovers the missing intermediate values $\{v_{i,j} : i \in \mathcal{R}_k, j \in \mathcal{N}(i), j \notin \mathcal{M}_k\}$ using locally computed intermediate values $\{v_{i,j} : i \in \mathcal{N}(j), j \in \mathcal{M}_k\}$ and received messages $\{X_{k'} : k' \in [K] \setminus \{k\}\}$.

We define the normalized communication load of the Shuffle phase as follows.

Definition 7.2 (Normalized Communication Load) *The normalized communication load, denoted by L, is defined as the number of bits communicated by K servers during the Shuffle phase, normalized by the maximum possible total number of bits in the inter-*

mediate values associated with all the Reduce functions, i.e.

$$L := \frac{\sum_{k=1}^{K} c_k}{n^2 T}.$$

For the running example in Figure 7.3(b), after the Map phase, each server obtains the intermediate values corresponding to the files in its subgraph. The intermediate values that are needed for computing the allocated Reduce functions but are not available after the Map phase have also been highlighted. We illustrate an uncoded Shuffling scheme in which each server is assigned the task of sending some of its locally available intermediate values to other server over the shared multicast network. We highlight here that each intermediate value missing at a server is available at two other servers. For example, $v_{5,1}$ and $v_{6,2}$ are missing at server 3, and both of them are available at servers 1 and 2. In this uncoded Shuffle, exactly one of the two servers is uniquely assigned the task to communicate the missing intermediate value to the server. For example, $v_{5,1}$ is multicasted by server 1 while $v_{6,2}$ is multicasted by server 2. As a total of 6 intermediate values are sent over the shared multicast network, the normalized communication load of the uncoded Shuffle is $L = \frac{6}{36}$.

The servers can instead send linear combinations of the intermediate values over the multicast network. For example, server 1 multicasts $v_{5,1} \oplus v_{3,4}$. As $v_{5,1}$ is locally available at server 2, server 2 can compute $(v_{5,1} \oplus v_{3,4}) \oplus v_{5,1}$ and obtain the missing intermediate value $v_{3,4}$. Similarly, server 3 can obtain the missing intermediate value $v_{5,1}$. This illustrates that by using *coded* Shuffle, in which each server sends a combination of locally available intermediate values over the multicast network, the communication load can be improved over the uncoded Shuffle. In this case specifically, the communication load for the coded Shuffle is $L = \frac{3}{36}$, which is factor of two (same as the computation load $r = 2$) improvement over uncoded Shuffle. This forms the motivation behind our proposed scheme in Section 7.4.

Reduce phase: Using its locally computed intermediate values and the intermediate values recovered from the messages received from other servers during the Shuffle phase, server $k \in [K]$ computes the Reduce functions in $\mathcal{R}_k$ to calculate $o_i = h_i(\{v_{i,j} : j \in \mathcal{N}(i)\})$ for all $i \in \mathcal{R}_k$.

In Figure 7.3(b), each server has all the intermediate values that are needed to compute the allocated Reduce functions. For example, for computing the Reduce function associated with vertex 1, server 1 has intermediate values $v_{1,1}$ and $v_{1,2}$ available locally from the Map phase and the intermediate value $v_{1,5}$ obtained from server 2 in the Shuffle phase. Therefore, each of the three servers can compute the Reduce functions allocated to it.

7.2.3 Problem Formulation

As illustrated in Figure 7.3, the communication load during Shuffle phase depends on subgraph allocation, Reduce allocation, and Shuffle strategy. For an allowed computation load r, our broader goal is to minimize the communication load during Shuffle phase through efficient schemes for allocation of subgraphs and Reducers to servers and for *coded* Shuffling of intermediate values among the servers. We consider a random undirected graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, where edges independently exist with probability $\Pr[(i,j) \in \mathcal{E}]$ for all $i, j \in \mathcal{V}$. Let $\mathcal{A}(r)$ be the set of all possible subgraph and Reduce allocations for a given computation load r (as defined in the previous subsection). For a graph realization G of $\mathcal{G}$ and an allocation $A \in \mathcal{A}(r)$, a coded Shuffling scheme is feasible if each server can compute all the Reduce functions assigned to it. We denote by $L_A(r, G)$ the minimum (normalized) communication load (as defined in Definition 7.2) over all feasible Shuffling coding schemes that enable each server to compute all the Reduce functions assigned to it.[1] Hence, for a given realization G of the random graph $\mathcal{G}$, the minimum communication load among all possible subgraph and Reduce allocations and feasible coded Shuffling schemes is as follows:

$$L^*_G(r) \coloneqq \inf_{A\in\mathcal{A}(r)} L_A(r,G). \tag{7.2}$$

Remark 7.2 *Partitioning of graphs in popular graph processing frameworks such as Pregel [19] is solely based on the vertex ID and not on the vertex neighborhood density. Furthermore, designing subgraph allocation, Reduce allocation and Shuffling schemes for characterizing the minimum communication load in (7.2) is NP-hard in general. This is because for the case of computation load $r = 1$, finding the minimum communication load is equivalent to finding the minimum K-cut over the graph, which is NP-hard for general graphs [193]. Additionally, existing heuristics for load balancing in distributed graph processing involve additional steps such as migration of vertex files during graph algorithm execution [194], which adds latency to the overall execution time. Hence, we focus on the problem of finding the subgraph and Reduce allocation tuple $A \in \mathcal{A}(r)$ that minimizes the average normalized communication load across all graph realizations G of $\mathcal{G}$.*

We formally define our problem as follows.

Problem: For a given random undirected graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ and a computation load $r \in [K]$, our goal is to characterize the minimum average normalized communication load, i.e.

$$L^*(r) \coloneqq \inf_{A\in\mathcal{A}(r)} \mathbb{E}_{\mathcal{G}}[L_A(r,\mathcal{G})]. \tag{7.3}$$

Remark 7.3 *For $r \geq K$, $L^*(r)$ is trivially 0 as each vertex can be mapped at each server,*

[1]The uncoded Shuffling schemes are special cases of the coded Shuffling schemes and are thus included in the set of all feasible coded Shuffling schemes under consideration.

so all the intermediate values associated with the Reducers of any server is available at the server.

Remark 7.4 *As defined above, $L^*(r)$ essentially reveals a fundamental trade-off between computation and communication in distributed graph processing.*

Remark 7.5 *In the above problem formulation, for a given subgraph and Reduce allocation tuple $A \in \mathcal{A}(r)$, in order to minimize the average communication load, the Shuffle scheme needs to take into consideration the connectivity of each realization G of $\mathcal{G}$. As we describe in Section 7.4, our proposed coded scheme utilizes careful alignment of intermediate values for creating coded messages for multicast during the Shuffle phase, leading to significant improvement in the average communication load.*

Remark 7.6 *Although the main focus of our problem formulation is on minimizing the average communication load for random graph models, our proposed coded scheme in Section 7.4 is applicable to any real-world graph. As demonstrated in Section 7.7, our proposed coded scheme can provide significant performance gains in practice. Specifically, for implementing PageRank over the real-world social webgraph TheMarker Cafe [183], our proposed scheme provides a gain of up to 43.4% in the overall execution time in comparison to the conventional PageRank implementation.*

In the next Section, we discuss our main results for four popular random graph models.

7.3 Main Results

In this section, we present the main results of our work. Our first result is the characterization of $L^*(r)$ (defined in (7.3)) for the Erdös-Rényi model that is defined below.

Erdös-Rényi Model: Denoted by $\mathrm{ER}(n,p)$, this model consists of graphs of size n in which each edge exists with probability $p \in (0,1]$, independently of other edges (Figure 7.5(a)).

Theorem 7.1 *For the Erdös-Rényi model $ER(n,p)$ with $p = \omega(\frac{1}{n^2})$, we have*

$$\lim_{n\to\infty} \frac{L^*(r)}{p} = \frac{1}{r}\left(1 - \frac{r}{K}\right).$$

Proof: We prove the achievability and the converse in Theorem 7.1 in Sections 7.4 and 7.5, respectively. ■

Remark 7.7 *Theorem 7.1 reveals an interesting inverse-linear trade-off between computation and communication in distributed graph processing. Specifically, our proposed coded scheme in Section 7.4 asymptotically gives a gain of r in the average normalized communication load in comparison to the uncoded Shuffling scheme that as we discuss later in Section 7.4, only achieves an average normalized communication load of $p(1-\frac{r}{K})$. This trade-off can be used to leverage additional computing resources and capabilities to alleviate the costly communication bottleneck. Moreover, we numerically demonstrate that even for finite graphs, not only the proposed scheme significantly reduces the communication load in comparison to the uncoded scheme, but also has a small gap from the optimal average normalized communication load (Figure 7.4). Finally, the assumption $p = \omega(\frac{1}{n^2})$ implies the regime of interest in which the average number of edges in the graph is growing with n. Otherwise, the problem would not be of interest since the communication load would become negligible even without redundancy/coding in computation.*

Remark 7.8 *Achievability Theorem 7.1 is proved in Section 7.4, where we provide subgraph and Reduce allocations followed by the code design for Shuffling for our proposed*

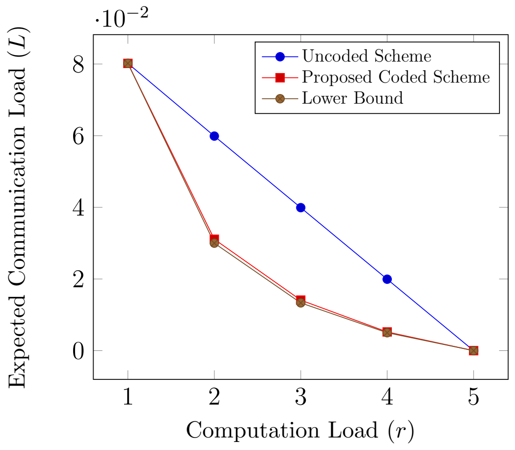
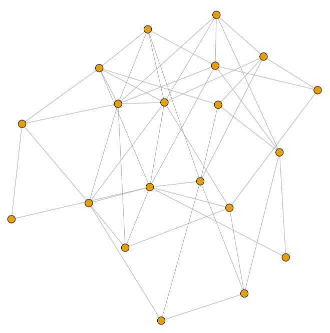


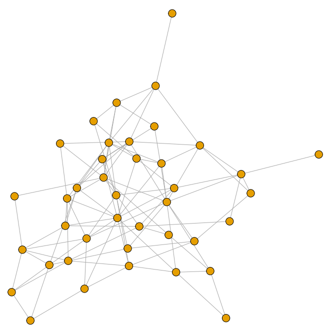
Figure 7.4: Performance comparison of our proposed coded scheme with uncoded Shuffle scheme and the proposed lower bound. The averages for the communication load for the two schemes were obtained over graph realizations of the Erdös-Rényi model with $n = 300$, $p = 0.1$ and $K = 5$.

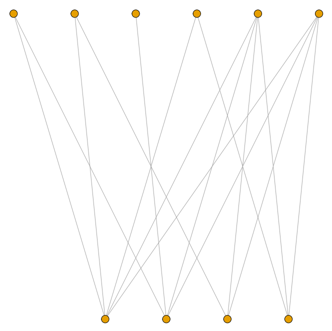
scheme. The main idea is to leverage the coded multicast opportunities offered by the injected redundancy and create coded messages which simultaneously satisfy the data demand of multiple servers. Careful combination of available intermediate values during the Shuffle phase benefits from the missing graph connections by aligning the intermediate values assigned to be communicated over the shared network. Conversely, Theorem 7.1 demonstrates that the asymptotic bandwidth gain r achieved by the proposed scheme is optimal and can not be improved. For the proof of converse provided in Section 7.5, we use induction to derive information-theoretic lower bounds on the average normalized communication load required by any subset of servers and then use the induction on the set of all the K servers.

Our second result is the characterization of $L^*(r)$ for the power law model that is defined below.

Power Law Model: Denoted by PL(n, γ, ρ), this model consists of graphs of size n in which degrees are i.i.d random variables drawn from a power law distribution with exponent γ and edge probabilities are ρ-proportional to product of the degrees of the two


(a) Erdös-Rényi model with $n = 20$.


(b) Power law model with $n = 40$, $\gamma = 2.3$ and 100 edges.


(c) Random bipartite model with $n_1 = 6$ and $n_2 = 4$.

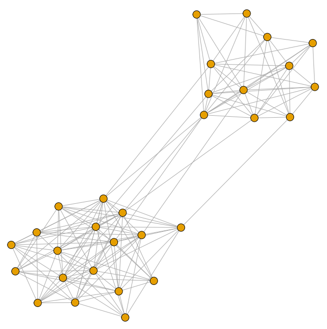

(d) Stochastic block model with $n_1 = 12$ and $n_2 = 18$.

Figure 7.5: Illustrative instances of the random graph models considered in the paper. In Figure 7.5(a), each edge exists with a given probability p. In Figure 7.5(b), expected degree of each vertex follows a power law distribution with exponent γ. In Figure 7.5(c), each cross-edge exists with a given probability q. In Figure 7.5(d), each intra-cluster edge exists with a given probability p and each cross-edge exists with a given probability q.

end vertices (Figure 7.5(b)).

Theorem 7.2 *For the power law model graph $PL(n, \gamma, \rho)$ with node degrees $\{d_1, \cdots, d_n\}$, $\gamma > 2$ and $\rho = \frac{1}{\sum_{i=1}^{n} d_i}$, we have*

$$\limsup_{n\to\infty} \frac{nL^*(r)}{\left(\frac{\gamma-1}{\gamma-2}\right)} \leq \frac{1}{r}\left(1 - \frac{r}{K}\right).$$

Proof: We prove the achievability in Theorem 7.2 in Section 7.6. ■

Remark 7.9 *Theorem 7.2 demonstrates that an inverse-linear trade-off between computation load and communication load can also be achieved in the power law model. We leverage our coded scheme proposed in Section 7.4 for the proof of Theorem 7.2 in Section 7.6.*

Furthermore, we specialize our proposed coded scheme in Section 7.4 to develop subgraph allocation and Reduce allocation schemes along with coded Shuffling schemes for two other popular random graph models which are described below:

Random Bi-partite Model: Denoted by $\mathrm{RB}(n_1, n_2, q)$, this model consists of graphs with two disjoint clusters of sizes n_1 and n_2 in which each inter-cluster edge exists with probability $q \in (0, 1]$, independently of other inter-cluster edges (Figure 7.5(c)). No intra-cluster edge exists in this model.

Stochastic Block Model: Denoted by $\mathrm{SBM}(n_1, n_2, p, q)$, this model consists of graphs with two disjoint clusters of sizes n_1 and n_2 such that each intra-cluster edge exists with probability p and each inter-cluster edge exists with probability $q, 0 < q < p \leq 1$, all independent of each other (Figure 7.5(d)).

The following theorems provide the achievability and converse results for RB and SBM models.

Theorem 7.3 *For the random bi-partite model $RB(n_1, n_2, q)$ with $n = n_1 + n_2$, $n_1 = \Theta(n)$, $n_2 = \Theta(n)$, $|n_1 - n_2| = o(n)$ and $q = \omega(\frac{1}{n^2})$, we have*

$$\frac{1}{8r}\left(1 - \frac{2r}{K}\right) \leq \limsup_{n\to\infty} \frac{L^*(r)}{q} \leq \frac{1}{2r}\left(1 - \frac{2r}{K}\right).$$

Proof: We defer the proof of the achievability and the converse in Theorem 7.3 to Appendices F.3 and F.4, respectively. ■

Remark 7.10 *Theorem 7.3 characterizes the optimal average normalized communica-*

tion load within a factor of 4 *for the random bi-partite model. We provide the proofs for achievability and converse of Theorem 7.3 in Appendices and F.3 and F.4 respectively. For achievability, we observe that there are no intra-cluster edges in the random bi-partite model, due to which intermediate values for a particular Reducer in one cluster only comes from Mappers in the other cluster. Therefore, we specialize our proposed coded scheme in Section 7.4 for the random bi-partite model, partitioning the available servers in proportion to the cluster sizes. Therefore, there is maximum overlap between Reducers corresponding to vertices in one cluster and Mappers corresponding to vertices in other cluster. For proving the converse, we remove vertices (and the edges corresponding to them) from the larger cluster so that the reduced graph has two clusters of equal sizes. The reduced graph model thus has two sets of Mappers and Reducers, which correspond to two different Erdös-Rényi models. Applying our lower bound for the Erdös-Rényi model in Theorem 7.1, we arrive at the converse of the bi-partite model.*

Theorem 7.4 *For the stochastic block model SBM*(n_1, n_2, p, q) *with* $n = n_1 + n_2$, $n_1 = \Theta(n)$, $n_2 = \Theta(n)$, *and* $p = \omega(\frac{1}{n^2}), q = \omega(\frac{1}{n^2})$, *we have*

$$\limsup_{n\to\infty} \frac{L^*(r)}{\frac{pn_1^2+pn_2^2+2qn_1n_2}{(n_1+n_2)^2}} \leq \frac{1}{r}\left(1-\frac{r}{K}\right). \tag{7.4}$$

Moreover, the following converse inequality holds:

$$\frac{L^*(r)}{q} \geq \frac{1}{r}\left(1-\frac{r}{K}\right). \tag{7.5}$$

Proof: We defer the proof of the achievability and the converse in Theorem 7.4 to Appendices F.5 and F.6, respectively. ∎

Remark 7.11 *Using (7.4) and (7.5), it can be easily verified that for the stochastic block model, the converse is within a constant factor of achievability if* $p = \Theta(q)$. *The achiev-*

ability and converse of Theorem 7.4 are proved in Appendices F.5 and F.6 respectively. For achievability, we specialize our proposed coded scheme from Section 7.4 based on the observation that in SBM, the Reducers corresponding to vertices in one cluster depend on the Mappers corresponding to the vertices within the cluster with one probability (due to intra-cluster edges), and on the vertices in the other cluster with another probability (due to cross-cluster edges). For the converse, the key idea is to randomly remove edges from the SBM model such that a larger ER model is obtained, then utilize a coupling argument, and finally use our information theoretic converse bound in Theorem 7.1.

7.4 Proposed Scheme and Proof of Achievability of Theorem 7.1

In this section, we first describe our proposed coded scheme for distributed graph analytics, and then leverage it to prove the achievability for the Erdös-Rényi model in Theorem 7.1.

7.4.1 Proposed Scheme

As described in our distributed graph processing framework in Section 7.2, a scheme for distributed implementation of the graph computations consists of subgraph allocation, Reduce allocation, and Shuffling algorithm. We next precisely describe our proposed scheme for a given realization G of the underlying random graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$.

Subgraph Allocation: The n files associated with the n vertices of G are first partitioned serially into $\binom{K}{r}$ batches $\mathcal{B}_1, \mathcal{B}_2, \ldots, \mathcal{B}_{\binom{K}{r}}$, where $\mathcal{B}_j$ comprises of the files associated with the vertices with IDs in the range $\{(j-1)g+1, (j-1)g+2, \ldots, jg\}$. Here, $g = n/\binom{K}{r}$ denotes the number of files in each batch. For our example with a graph of 6 vertices, 3 servers, and computation load 2 presented in Section 7.2, the 6 files are partitioned into $\binom{K}{r} = 3$ batches each of size $g = 2$ as follows (see Figure 7.6(a)):

$$\mathcal{B}_1 = \{1, 2\},$$
$$\mathcal{B}_2 = \{3, 4\},$$
$$\mathcal{B}_3 = \{5, 6\}.$$

Each of the $\binom{K}{r}$ batches of files is associated with a unique set of r servers. Specifically, let $\mathcal{F}_1, \mathcal{F}_2, \ldots, \mathcal{F}_{\binom{K}{r}}$ denote all possible combinations of the elements of $\{1, 2, \ldots, K\}$. Then, each of the servers with indices in $\mathcal{F}_j$ is allocated each of the files contained in batch $\mathcal{B}_j$. Thus, server $k \in [K]$ Maps the vertices in $\mathcal{B}_j$ if $k \in \mathcal{F}_j$. Equivalently, $\mathcal{B}_j \subseteq \mathcal{M}_k$ if $k \in \mathcal{F}_j$. Therefore, we have the following for the subgraph allocation for server k:

$$\mathcal{M}_k = \cup_{j \in \left[\binom{K}{r}\right], k \in \mathcal{F}_j} \mathcal{B}_j.$$

As each server is present in $\binom{K-1}{r-1}$ of the $\binom{K}{r}$ unique combinations of servers, we have the following for each server $k \in [K]$:

$$|\mathcal{M}_k| = \binom{K-1}{r-1} g = \binom{K-1}{r-1} \frac{n}{\binom{K}{r}} = \frac{rn}{K}.$$

In Figure 7.6(a), we illustrate the subgraph allocation for our running example. $\mathcal{F}_1 = \{1, 2\}$, $\mathcal{F}_2 = \{1, 3\}$ and $\mathcal{F}_3 = \{2, 3\}$. Each of the two files in batch $\mathcal{B}_j$ is assigned to each of the servers in $\mathcal{F}_j$, for $j \in \{1, 2, 3\}$. Thus, server 1 is allocated files $\mathcal{B}_1 \cup \mathcal{B}_2 = \{w_1, w_2, w_3, w_4\}$, server 2 is allocated files $\mathcal{B}_1 \cup \mathcal{B}_3 = \{w_1, w_2, w_5, w_6\}$ and server 3 is allocated $\mathcal{B}_2 \cup \mathcal{B}_3 = \{w_3, w_4, w_5, w_6\}$. Thus, $|\mathcal{M}_1| = |\mathcal{M}_2| = |\mathcal{M}_3| = 4$.

Reduce Allocation: The n Reduce functions associated with the n graph vertices are disjointly and uniformly partitioned into K subsets and each subset is assigned exclusively to one server. Specifically, for $k \in [K]$, $|\mathcal{R}_k| = \frac{n}{K}$ and $\mathcal{R}_k = \{(k-1)\frac{n}{K}+1, (k-1)\frac{n}{K}+2, \dots, k\frac{n}{K}\}$. In our running example, $\mathcal{R}_1 = \{1,2\}$, $\mathcal{R}_2 = \{3,4\}$ and $\mathcal{R}_3 = \{5,6\}$.

For notational convenience, we denote our proposed subgraph allocation and Reduce allocation by A_{C}.

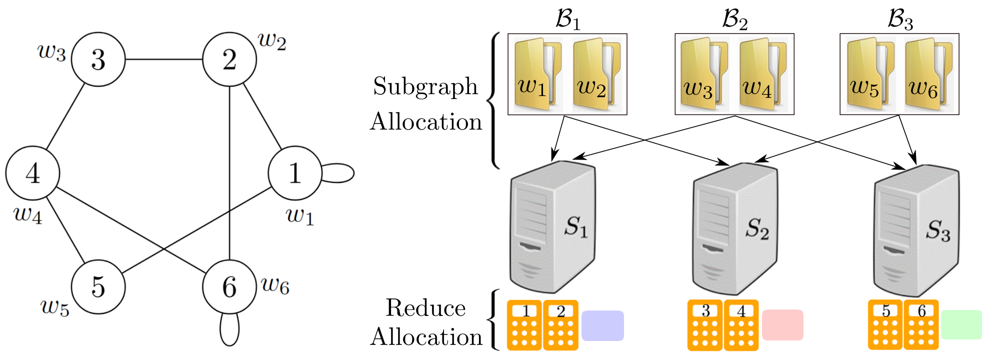
Coded Shuffle: As illustrated in Figure 7.3(b), the key idea in coded Shuffling is to create coded combinations of locally available intermediate values so that the same message can be useful for many servers simultaneously. Due to the subgraph and Reduce allocation A_{C} described above, every set $\mathcal{F}_j$ of r servers has a unique batch of files $\mathcal{B}_j$. Thus, all the intermediate values corresponding to the Map computations associated with the files in $\mathcal{B}_j$ are available at every server in $\mathcal{F}_j$ after the Map phase. With this observation, consider without loss of generality the set of servers $\mathcal{S} = \{1, 2, \dots, r+1\}$. For each server $k \in \mathcal{S}$, let $\mathcal{Z}^k_{\mathcal{S}\setminus\{k\}}$ be the set of all intermediate values that are needed by Reduce functions in k, and are available exclusively at each server $k' \in \mathcal{S}\setminus\{k\}$, i.e.

$$\mathcal{Z}^k_{\mathcal{S}\setminus\{k\}} = \{v_{i,j} : (i,j) \in \mathcal{E}, i \in \mathcal{R}_k, j \in \cap_{k' \in \mathcal{S}\setminus\{k\}} \mathcal{M}_{k'}\}. \tag{7.6}$$

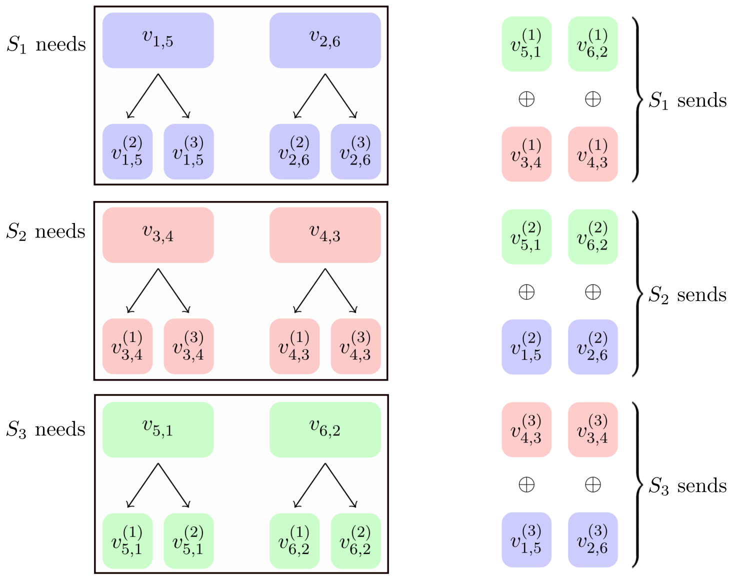
We observe that after the Map phase, server $r+1$ has $\mathcal{Z}^k_{\mathcal{S}\setminus\{k\}}$ for $k \in \{1, \dots, r\}$. Furthermore, server 1 has $\mathcal{Z}^k_{\mathcal{S}\setminus\{k\}}$ for $k \in \{2, \dots, r\}$, server 2 has $\mathcal{Z}^k_{\mathcal{S}\setminus\{k\}}$ for $k \in \{1, 3, \dots, r\}$, and so on. Therefore, server $r+1$ can create a coded message by selecting one intermediate value each from $\mathcal{Z}^k_{\mathcal{S}\setminus\{k\}}$ for $k \in \{1, \dots, r\}$, and taking a XOR of them. The coded message is simultaneously useful for the servers $\{1, \dots, r\}$ as each of them can XOR out *its* own missing intermediate value as it has the remaining intermediate values associated with the coded message. Similar arguments hold for the coded messages from other servers within $\mathcal{S}$.

In light of the above arguments, for each $k \in \mathcal{S}$, each intermediate value $v_{i,j} \in \mathcal{Z}^k_{\mathcal{S}\setminus\{k\}}$

is evenly split into r segments $v_{i,j}^{(1)}, \cdots, v_{i,j}^{(r)}$, each of size $\frac{T}{r}$ bits. Each segment is associated with a distinct server in $\mathcal{S} \setminus \{k\}$, where the segment assignment is based on the order



(a) Illustrating the subgraph allocation and Reduce allocation A_C for the example graph with 6 vertices. The 6 files are partitioned into 3 batches and each batch is assigned to a unique subset of 2 servers. The Reduce functions are partitioned into 3 sets, one set is assigned to each server.



(b) For the subgraph and Reduce allocations A_C in Figure 7.6(a), we illustrate our proposed coded Shuffle scheme. For each intermediate value needed by a server, each of the remaining two servers is assigned the task of communicating a segment which is one-half of the intermediate value. The servers create a table of the segments that they are assigned to send, with each row corresponding to the intermediate values required exclusively by one of the remaining servers. Each server sends two coded messages, each of which is simultaneously useful for both the remaining servers.

Figure 7.6: Illustration of our proposed scheme.

of the indices of the r servers $\mathcal{S} \setminus \{k\}$. Therefore, $\mathcal{Z}^k_{\mathcal{S}\setminus\{k\}}$ is evenly partitioned to r sets, which are denoted by $\mathcal{Z}^k_{\mathcal{S}\setminus\{k\},s}$ for $s \in \mathcal{S} \setminus \{k\}$. Depending on the connectivity of G, the number of intermediate values in $\mathcal{Z}^k_{\mathcal{S}\setminus\{k\}}$ shall vary, and the maximum possible size of $\mathcal{Z}^k_{\mathcal{S}\setminus\{k\}}$ is $\tilde{g} = g\frac{n}{K} = \frac{n^2}{K\binom{K}{r}}$. Each server $s \in \mathcal{S}$ creates an $r \times \tilde{g}$ table and fills that out with segments which are associated with it. Each row of the table is *filled from left* by the segments in one of the sets $\mathcal{Z}^k_{\mathcal{S}\setminus\{k\},s}$, where $k \in \mathcal{S} \setminus \{s\}$ (see Figure 7.7). Then, server s broadcasts the XOR of all the segments in each non-empty column of the table, where for each non-empty column, *the empty entries are zero padded*. Clearly, there exist at most $\tilde{g}$ of such coded messages. The process is carried out similarly for every other subset $\mathcal{S} \subseteq [K]$ of servers with $|\mathcal{S}| = r + 1$.

After the Shuffle phase, for each multicast group of $r + 1$ servers, all but one intermediate values contributed in each coded message are locally available. Moreover, all possible subsets of multicast servers have sent their corresponding messages. Therefore, each server can recover all of the intermediate values associated with its assigned set of Reduce functions using the received coded messages and the locally computed intermediate values. Thus, our proposed coded Shuffling scheme is feasible, i.e. for any given graph, and subgraph and Reduce allocation A_{C}, our proposed Shuffling enables each server to compute all the Reduce functions assigned to it.

Remark 7.12 *The proposed scheme carefully aligns and combines the existing intermediate values to benefit from the coding opportunities. This resolves the issue posed by the asymmetry in the data requirements of the Reducers which is one of the main challenges in moving from the general MapReduce framework in [113] to graph analytics.*

In Figure 7.6(b), every intermediate value in $\mathcal{Z}^3_{\{1,2\}} = \{v_{5,1}, v_{6,2}\}$ is split into $r = 2$ segments, each associated with a distinct server in $\{1, 2\}$. This is done similarly for servers 1 and 2. Then, servers 1, 2, and 3 broadcast their coded messages $X_1 = \{v^{(1)}_{5,1} \oplus v^{(1)}_{4,3}, v^{(1)}_{3,4} \oplus$

$v_{6,2}^{(1)}\}$, $X_2 = \{v_{5,1}^{(2)} \oplus v_{1,5}^{(1)}, v_{6,2}^{(2)} \oplus v_{2,6}^{(1)}\}$, and $X_3 = \{v_{4,3}^{(2)} \oplus v_{1,5}^{(2)}, v_{3,4}^{(2)} \oplus v_{2,6}^{(2)}\}$, respectively. All three servers can recover their missing intermediate values. For instance, server 3 needs $v_{5,1}$ to carry out the Reduce function associated with vertex 5. Since it has already Mapped vertices 3 and 5, intermediate values $v_{4,3}$ and $v_{1,5}$ are available locally. Server 3 can recover $v_{5,1}^{(1)}$ and $v_{5,1}^{(2)}$ from $v_{5,1}^{(1)} \oplus v_{4,3}^{(1)}$ and $v_{5,1}^{(2)} \oplus v_{1,5}^{(1)}$, respectively. As each server sends 2 coded messages to other servers and each coded message is half the size of an intermediate value, therefore, the overall normalized communication load is $\frac{3}{36}$, which is two times better than the normalized communication load for uncoded Shuffling.

7.4.2 Proof of Achievability of Theorem 7.1

We now analyze the performance of our proposed coded scheme in Section 7.4.1 for the Erdös-Rényi random graph model to prove the achievability of Theorem 7.1. For our proposed subgraph and Reduce allocation A_{C}, we first compute the average communication for uncoded Shuffle where no coding is utilized during the Shuffle phase.

Uncoded Shuffle: Given the subgraph and Reduce allocation A_{C}, consider a server $k \in [K]$. Due to symmetry, the total expected communication load is sum of the communication loads of each server. Hence we can focus on finding the communication load of server 1. Note that there are n/K Reducers assigned to server 1, and $\frac{rn}{K}$ Mappers assigned to server 1. Therefore, for each Reducer in server 1, the expected communication required is $(pn - p\frac{rn}{K})T$. Summing over the expected communication loads for all the Reducers in server 1 and appropriate normalization, the total expected communication load for server 1 is $\frac{n}{K}(pn - p\frac{rn}{K})T$. Summing over all the K servers, we get the average normalized communication load for the uncoded Shuffle as follows:

$$\bar{L}_{A_{\mathsf{C}}}^{\mathsf{UC}} := \mathbb{E}_{\mathcal{G}}[L_{A_{\mathsf{C}}}^{\mathsf{UC}}(r, \mathcal{G})] = K\frac{n}{K}\left(pn - p\frac{rn}{K}\right)T\frac{1}{n^2T} = p\left(1 - \frac{r}{K}\right),$$

where $L^{\mathsf{UC}}_{A_{\mathsf{C}}}(r, G)$ denotes the normalized communication load for uncoded Shuffle for the graph realization G of the Erdös-Rényi random graph model $\mathcal{G}$.

We now apply our proposed coded Shuffle scheme and compute the induced average communication load. Without loss of generality, we analyze our algorithm by a generic argument for servers $\mathcal{S} = \{1, \cdots, r+1\}$ which can be similarly applied for any other set of servers $\mathcal{S}$ with $|\mathcal{S}| = r+1$, due to the symmetric structure induced by the graph model and subgraph allocation and Reduce allocation A_{C}. Denote $r+1$ servers as $s_1, \cdots, s_{r+1}$, and consider the messages that s_1 is assigned to send within the multicast group $\mathcal{S}$, the coded messages that are sent by other servers within $\mathcal{S}$ are also created similarly. As described in Section 7.4.1 and illustrated in Figure 7.7, server s_1 creates a table of intermediate value segments for transmission. In this table, each row is filled from the left, and for $i \in [r]$, i'th row contains the allocated segments for the intermediate values in the set $\mathcal{Z}^{s_{i+1}}_{\mathcal{S}\setminus\{s_{i+1}\}, s_1}$. The number of segments in $\mathcal{Z}^{s_{i+1}}_{\mathcal{S}\setminus\{s_{i+1}\}, s_1}$, denoted by $\tilde{g}_i$, depends on the connectivity of the graph G and is upper bounded by $\tilde{g}$, the total number of intermediate values in $\mathcal{Z}^{s_{i+1}}_{\mathcal{S}\setminus\{s_{i+1}\}}$ for a completely connected graph. Server s_1 broadcasts at most $\tilde{g}_{\max} = \max(\tilde{g}_1, \tilde{g}_2, \ldots, \tilde{g}_r)$ coded messages $X^1, \cdots, X^{\tilde{g}_{\max}}$, zero padding the empty entries in the non-empty columns. These coded messages are simultaneously and exclusively useful for the servers $s_2, \cdots, s_{r+1}$. For each non-empty column $j \in [\tilde{g}_{\max}]$, X^j is XOR of at most r non-zero segments of size $\frac{T}{r}$ bits, associated with server s_1. More formally, for each non-empty column $j \in [\tilde{g}_{\max}]$, we have the following:

$$X^j = \bigoplus_{i=1}^{r} v^{(1)}_{\alpha(i,j)}. \tag{7.7}$$

In (7.7), for $i \in [r]$ and $j \in [\tilde{g}_i]$, we have used $v^{(1)}_{\alpha(i,j)}$ to denote the non-zero segment in the table in i'th row and j'th column, while for $j \in \{\tilde{g}_i + 1, \tilde{g}_i + 2, \ldots, \tilde{g}\}$, $v^{(1)}_{\alpha(i,j)}$ denotes the zero padding segment.

Let Bern(p) random variable $E_{\alpha(i,j)}$ indicate the existence of the edge $\alpha(i,j) \in \mathcal{V} \times \mathcal{V}$, i.e. $E_{\alpha(i,j)} = 1$, if $\alpha(i,j) \in \mathcal{E}$, and $E_{\alpha(i,j)} = 0$, otherwise. Clearly, for all vertices $i, j, t, u \in \mathcal{V}$, $E_{\alpha(i,j)}$ is independent of $E_{\alpha(t,u)}$ if $\alpha(i,j)$ and $\alpha(t,u)$ do not represent the same edge, and $E_{\alpha(i,j)} = E_{\alpha(t,u)}$, otherwise. For $i \in [r]$, the random variable P_i is defined as

$$P_i = \sum_{j=1}^{\tilde{g}} E_{\alpha(i,j)}, \tag{7.8}$$

i.e. each P_i is sum of $\tilde{g}$ possibly dependent Bern(p) random variables. Note that P_i's are not independent in general. By careful alignment of present intermediate values (Figure 7.7), s_1 broadcasts Q coded messages each of size $\frac{T}{r}$ bits, where $Q = \max_{i \in [r]} P_i$. Thus, the total coded communication load sent from server s_1 exclusively for servers $s_2, \cdots, s_{r+1}$ is $\frac{T}{r}Q$ bits. By similar arguments for other sets of servers, we can characterize the average

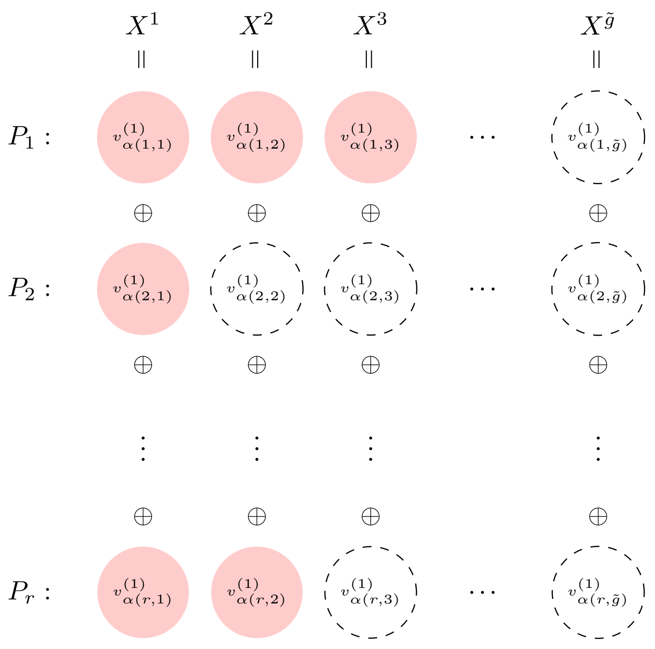


Figure 7.7: Creating coded messages by aligning the associated intermediate value segments.

normalized coded communication load of the proposed scheme as follows:

$$\bar{L}^{\mathsf{C}}_{A_{\mathsf{C}}} := \mathbb{E}_{\mathcal{G}}[L^{\mathsf{C}}_{A_{\mathsf{C}}}(r,\mathcal{G})] = \frac{1}{rn^2} K \binom{K-1}{r} \mathbb{E}[Q], \tag{7.9}$$

where $L^{\mathsf{C}}_{A_{\mathsf{C}}}(r,G)$ denotes the normalized communication load for the proposed coded Shuffle for the graph realization G of the Erdös-Rényi random graph model $\mathcal{G}$.

The following lemma asymptotically upper bounds $\mathbb{E}[Q]$.

Lemma 7.1 *For $ER(n,p)$ graphs with $p = \omega(\frac{1}{n^2})$, we have*

$$\mathbb{E}[Q] \leq p\tilde{g} + o(p\tilde{g}).$$

Proof: We defer the proof to Appendix F.1. ■

Putting (A.6) and Lemma 7.1 together, we have

$$L^*(r) \leq \bar{L}^{\mathsf{C}}_{A_{\mathsf{C}}} \leq \frac{1}{r} p \left(1 - \frac{r}{K}\right) + o(p),$$

hence the achievability claimed in Theorem 7.1 is proved. Finally, we note that as explained in the uncoded Shuffle algorithm, the average normalized uncoded communication load of the proposed scheme is $\bar{L}^{\mathsf{UC}}_{A_{\mathsf{C}}} = p\left(1 - \frac{r}{K}\right)$, which implies that our scheme achieves an asymptotic gain of r.

Remark 7.13 *As we next show in the proof of Lemma 7.1, the regime $p = \omega(1/n^2)$ is essential in order to have $p\tilde{g} = \omega(1)$. As $\tilde{g} = \frac{n^2}{K\binom{K}{r}} = \Theta(n^2)$ is a deterministic function of n, the regime $p = \omega(1/n^2)$ is needed to get the achievability and asymptotic optimality of Theorem 1.*

7.5 Converse for the Erdös-Rényi Model

In this section, we prove the asymptotic optimality of our proposed coded scheme for the Erdös-Rényi model, by leveraging the techniques employed in [113]. More precisely, we complete the proof of Theorem 7.1 by deriving the lower bound on the best average communication load for the Erdös-Rényi model, that matches the achievability in (7.10).

Let $\mathcal{G}$ be an ER(n,p) random graph and consider a subgraph and Reduce allocation $A = (\mathcal{M}, \mathcal{R}) \in \mathcal{A}(r)$, where $\sum_{k=1}^{K} |\mathcal{M}_k| = rn$ and $|\mathcal{R}_k| = \frac{n}{K}$, for all $k \in [K]$. We denote the number of files that are Mapped at j vertices under Map assignment $\mathcal{M}$, as $a^j_{\mathcal{M}}$, for all $j \in [K]$. The following lemma holds.

Lemma 7.2 $\mathbb{E}_{\mathcal{G}}[L_A(r,\mathcal{G})] \geq p \sum_{j=1}^{K} \frac{a^j_{\mathcal{M}}}{n} \frac{K-j}{Kj}$.

Proof: We let intermediate values $v_{i,j}$ be realizations of random variables $V_{i,j}$, uniformly distributed over $\mathbb{F}_{2^T}$. For a random graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ and subsets $\mathcal{I}, \mathcal{J} \subseteq \mathcal{V} = [n]$, define $V^{\mathcal{G}}_{\mathcal{I},\mathcal{J}} = \{V_{i,j} : (i,j) \in \mathcal{E}, i \in \mathcal{I}, j \in \mathcal{J}\}$ as the set of present intermediate values in graph $\mathcal{G}$ corresponding to Reducers in $\mathcal{I}$ and Mappers in $\mathcal{J}$. For a given allocation $A = (\mathcal{M}, \mathcal{R}) \in \mathcal{A}(r)$ and a subset of servers $\mathcal{S} \subseteq [K]$, we define $X_{\mathcal{S}} = \{X_k : k \in \mathcal{S}\}$ and $Y^{\mathcal{G}}_{\mathcal{S}} = (V^{\mathcal{G}}_{\mathcal{R}_{\mathcal{S}},:}, V^{\mathcal{G}}_{:,\mathcal{M}_{\mathcal{S}}})$, where ":" denotes all possible indices (which depend on both allocation and graph realization). As described in Section 7.2.2, each coded message is a function of the present intermediate values Mapped at the corresponding server. Moreover, all the intermediate values required by the Reducers are decodable from the locally available intermediate values and received messages at the corresponding server. That is, $H(X_k | V^{\mathcal{G}}_{:,\mathcal{M}_k}) = 0$ and $H(V^{\mathcal{G}}_{\mathcal{R}_k,:} | X_{[K]}, V^{\mathcal{G}}_{:,\mathcal{M}_k}) = 0$ for all servers $k \in [K]$ and graphs $\mathcal{G}$. We denote the number of vertices that are exclusively Mapped by j servers in $\mathcal{S}$ as $a^{j,\mathcal{S}}_{\mathcal{M}}$, that is

$$a^{j,\mathcal{S}}_{\mathcal{M}} := \sum_{\mathcal{S}_1 \subseteq \mathcal{S} : |\mathcal{S}_1| = j} |(\cap_{k \in \mathcal{S}_1} \mathcal{M}_k) \setminus (\cup_{k' \notin \mathcal{S}_1} \mathcal{M}_{k'})|.$$

We prove the following claim by induction in Appendix F.2.

Claim 7.1 *For any subset $\mathcal{S} \subseteq [K]$,*

$$\mathbb{E}_{\mathcal{G}}\Big[H(X_{\mathcal{S}}|Y^{\mathcal{G}}_{\mathcal{S}^c})\Big] \geq pT \sum_{j=1}^{|\mathcal{S}|} a^{j,\mathcal{S}}_{\mathcal{M}} \frac{n}{K} \frac{|\mathcal{S}|-j}{j}. \tag{7.10}$$

Proof: We defer the proof to Appendix F.2. ■

Now, pick $\mathcal{S} = [K]$. Then,

$$\mathbb{E}_{\mathcal{G}}\big[L_A(r,\mathcal{G})\big] \geq \frac{\mathbb{E}_{\mathcal{G}}\Big[H(X_{\mathcal{S}}|Y^{\mathcal{G}}_{\mathcal{S}^c})\Big]}{n^2 T} \geq p \sum_{j=1}^{K} \frac{a^j_{\mathcal{M}}}{n} \frac{K-j}{Kj}.$$

■

Proof of Converse for Theorem 7.1. First, we use the result in Claim E.1 and bound the best average normalized communication load as follows:

$$L^*(r) \geq \inf_A \mathbb{E}_{\mathcal{G}}\big[L_A(r,\mathcal{G})\big] \geq \inf_A p \sum_{j=1}^{K} \frac{a^j_{\mathcal{M}}}{n} \frac{K-j}{Kj},$$

where the infimum is over all subgraph and Reduce allocations $A = (\mathcal{M}, \mathcal{R}) \in \mathcal{A}(r)$ for which $\sum_{k=1}^{K} |\mathcal{M}_k| = rn$ and $|\mathcal{R}_k| = \frac{n}{K}$, $\forall k \in [K]$. Additionally, for any Map allocation with computation load r, we have the following equations:

$$\sum_{j=1}^{K} a^j_{\mathcal{M}} = n, \ \sum_{j=1}^{K} j a^j_{\mathcal{M}} = rn. \tag{7.11}$$

Using convexity of $\frac{K-j}{Kj}$ in j and (7.11), the converse is proved as follows:

$$L^*(r) \geq \inf_A p \sum_{j=1}^{K} \frac{a^j_{\mathcal{M}}}{n} \frac{K-j}{Kj} \geq \inf_A p \frac{K - \sum_{j=1}^{K} j \frac{a^j_{\mathcal{M}}}{n}}{K \sum_{j=1}^{K} j \frac{a^j_{\mathcal{M}}}{n}} = \frac{1}{r} p \left(1 - \frac{r}{K}\right).$$

7.6 Achievability for the Power Law Model

We consider a general model for random graphs where the expected degree sequence $\mathbf{d} = (d_1, \cdots, d_n)$ is independently drawn from a power law distribution with exponent γ, i.e. $\Pr[d_i = d] = cd^{-\gamma}$ for $i \in [n]$ and $d \geq 1$ and proper constant c [195]. Given the realization of the expected degrees $\mathbf{d}$, for $\rho = \frac{1}{\sum_{i=1}^{n} d_i}$ and all $i, j \in [n]$, vertices i and j are connected with probability $p_{i,j} = \Pr[(i,j) \in \mathcal{E}] = \rho d_i d_j$, independently of other edges. We now proceed to analyze the coded and uncoded communication loads averaged over the random connections and random degrees induced by the subgraph and Reduce allocation A_{C} proposed in Section 7.4.1.

Consider the allocation $A_{\mathsf{C}} = (\mathcal{M}, \mathcal{R})$ and a subset of servers $\mathcal{S} \subseteq [K]$ of size $|\mathcal{S}| = r + 1$. According to the proposed scheme in Section 7.4.1, for every server $s \in \mathcal{S}$, servers in $\mathcal{S} \setminus \{s\}$ form a table and construct coded messages using the intermediate values in the sets $\mathcal{Z}^k_{\mathcal{S}\setminus\{k\}}$ (defined in (7.6)) where $k \in \mathcal{S} \setminus \{s\}$. Therefore, $r+1$ tables are formed each constructing coded messages of size $\max_{k \in \mathcal{S}\setminus\{s\}} |\mathcal{Z}^k_{\mathcal{S}\setminus\{k\}}| \frac{T}{r}$ bits. The total coded load induced by the subset $\mathcal{S}$ (and exclusively for the use of servers in $\mathcal{S}$) denoted by $L^{\mathsf{C}}_{A_{\mathsf{C}}}(\mathcal{S})$ is

$$L^{\mathsf{C}}_{A_{\mathsf{C}}}(\mathcal{S}) = \frac{1}{n^2 r} \sum_{s \in \mathcal{S}} \max_{k \in \mathcal{S}\setminus\{s\}} |\mathcal{Z}^k_{\mathcal{S}\setminus\{k\}}|.$$

However, in uncoded scenarios, denoted by $L^{\mathsf{UC}}_{A_{\mathsf{C}}}(\mathcal{S})$ the total uncoded load induced by subset $\mathcal{S}$ (and exclusively for the use of servers in $\mathcal{S}$) is

$$L^{\mathsf{UC}}_{A_{\mathsf{C}}}(\mathcal{S}) = \frac{1}{n^2} \sum_{s \in \mathcal{S}} |\mathcal{Z}^s_{\mathcal{S}\setminus\{s\}}|.$$

We have

$$|\mathcal{Z}^s_{\mathcal{S}\backslash\{s\}}| = 2\sum_{i\in\mathcal{R}_s} |\mathcal{N}(i)\cap(\cap_{k'\in\mathcal{S}\backslash\{s\}}\mathcal{M}_{k'})| = \sum_{\substack{i\in\mathcal{R}_s \\ m\in\cap_{k'\in\mathcal{S}\backslash\{s\}}\mathcal{M}_{k'}}} \mathbb{1}\{(i,m)\in\mathcal{E}\}, \tag{7.12}$$

where the random Bernoulli $\mathbb{1}\{(i,m)\in\mathcal{E}\}$ indicates the realization of the edge connecting vertices i and m, i.e. $\mathbb{E}[\mathbb{1}\{(i,m)\in\mathcal{E}\}|\mathbf{d}] = \rho d_i d_m$. We note that $|\mathcal{R}_s| = n/K$ and $|\cap_{k'\in\mathcal{S}\backslash\{s\}}\mathcal{M}_{k'}| = n/\binom{K}{r}$. Therefore, there are $\tilde{g} = \frac{n^2}{K\binom{K}{r}}$ Bernoulli summands in (7.12) in which every two summands are either independent or equal and independent of other summands. More precisely, (7.12) can be decomposed to sum of all independent Bernoulli random variables and sum of dependent ones as follows:

$$\begin{aligned} |\mathcal{Z}^s_{\mathcal{S}\backslash\{s\}}| &= \sum_{\substack{i\in\mathcal{R}_s \\ m\in\cap_{k'\in\mathcal{S}\backslash\{s\}}\mathcal{M}_{k'}}} \mathbb{1}\{(i,m)\in\mathcal{E}\} \\ &= \sum_{\mathcal{F}_1 \text{ or } \mathcal{F}_2 \text{ or } \mathcal{F}_3} \mathbb{1}\{(i,m)\in\mathcal{E}\} + 2\sum_{\substack{i,m\in\mathcal{R}_s\cap(\cap_{k'\in\mathcal{S}\backslash\{s\}}\mathcal{M}_{k'}) \\ i<m}} \mathbb{1}\{(i,m)\in\mathcal{E}\}, \end{aligned} \tag{7.13}$$

where we denote the events

$$\begin{aligned} \mathcal{F}_1 &\coloneqq \{i\in\mathcal{R}_s\setminus\cap_{k'\in\mathcal{S}\backslash\{s\}}\mathcal{M}_{k'}\,,\, m\in\cap_{k'\in\mathcal{S}\backslash\{s\}}\mathcal{M}_{k'}\}, \\ \mathcal{F}_2 &\coloneqq \{i\in\mathcal{R}_s\,,\, m\in\cap_{k'\in\mathcal{S}\backslash\{s\}}\mathcal{M}_{k'}\setminus\mathcal{R}_s\}, \\ \mathcal{F}_3 &\coloneqq \{i=m\in\mathcal{R}_s\cap(\cap_{k'\in\mathcal{S}\backslash\{s\}}\mathcal{M}_{k'})\}. \end{aligned}$$

Note that with this decompostion, all the Bernoulli summands in both terms in (7.13) are independent. Assume that the first and second terms in (7.13) contain $\tilde{g}-2J$ and J summands respectively.

According to Kolmogorov's strong law of large numbers (Proposition 7.1 provided at the end of this section) and given that the second condition in the proposition is satisfied for Bernoullis, we have

$$\frac{1}{\tilde{g}-2J}\sum_{\mathcal{F}_1 \text{ or } \mathcal{F}_2 \text{ or } \mathcal{F}_3} \mathbb{1}\{(i,m)\in\mathcal{E}\} - \mathbb{E}[\rho d_i d_m] \xrightarrow{\text{a.s.}} 0,$$

and

$$\frac{1}{J}\sum_{\substack{i,m\in\mathcal{R}_s\cap(\cap_{k'\in\mathcal{S}\setminus\{s\}}\mathcal{M}_{k'})\\ i<m}} \mathbb{1}\{(i,m)\in\mathcal{E}\} - \mathbb{E}[\rho d_i d_m] \xrightarrow{\text{a.s.}} 0.$$

Therefore, size of the set $\mathcal{Z}^s_{\mathcal{S}\setminus\{s\}}$ converges almost surely, that is

$$\begin{aligned}\frac{1}{\tilde{g}}\left(|\mathcal{Z}^s_{\mathcal{S}\setminus\{s\}}| - \mathbb{E}\big[|\mathcal{Z}^s_{\mathcal{S}\setminus\{s\}}|\big]\right) &= \frac{\tilde{g}-2J}{\tilde{g}}\frac{1}{\tilde{g}-2J}\sum_{\mathcal{F}_1 \text{ or } \mathcal{F}_2 \text{ or } \mathcal{F}_3} \mathbb{1}\{(i,m)\in\mathcal{E}\} - \mathbb{E}[\rho d_i d_m] \\ &\quad + \frac{J}{\tilde{g}}\frac{1}{J}2\sum_{\substack{i,m\in\mathcal{R}_s\cap(\cap_{k'\in\mathcal{S}\setminus\{s\}}\mathcal{M}_{k'})\\ i<m}} \mathbb{1}\{(i,m)\in\mathcal{E}\} - \mathbb{E}[\rho d_i d_m] \\ &\xrightarrow{\text{a.s.}} 0,\end{aligned}$$

where

$$\mathbb{E}\big[|\mathcal{Z}^s_{\mathcal{S}\setminus\{s\}}|\big] = \sum_{\substack{i\in\mathcal{R}_s\\ m\in\cap_{k'\in\mathcal{S}\setminus\{s\}}\mathcal{M}_{k'}}} \mathbb{E}[\rho d_i d_m] = \mathbb{E}\big[\rho\,\mathrm{vol}(\mathcal{R}_s)\mathrm{vol}(\cap_{k'\in\mathcal{S}\setminus\{s\}}\mathcal{M}_{k'})\big],$$

and $\mathrm{vol}(V) = \sum_{v\in V} d_v$ for any subset of vertices $V \subseteq [n]$. Moreover,

$$\lim_{n\to\infty}\frac{n}{\tilde{g}}\mathbb{E}\big[|\mathcal{Z}^s_{\mathcal{S}\setminus\{s\}}|\big] = \lim_{n\to\infty}\mathbb{E}\left[(\rho n)\frac{1}{n/K}\mathrm{vol}(\mathcal{R}_s)\frac{1}{n/\binom{K}{r}}\mathrm{vol}(\cap_{k'\in\mathcal{S}\setminus\{s\}}\mathcal{M}_{k'})\right]. \tag{7.14}$$

Each of the terms $\mathrm{vol}(\mathcal{R}_s)$, $\mathrm{vol}(\cap_{k'\in\mathcal{S}\setminus\{s\}}\mathcal{M}_{k'})$ and inverse of ρ are summation of i.i.d power law random variables for which the expected value exists for $\gamma > 2$ and $\mathbb{E}[d_1] = \frac{\gamma-1}{\gamma-2}$.

Therefore, by strong law of large numbers (Proposition 7.1) each term approaches its average almost surely, that is for $\gamma > 2$

$$\frac{1}{n/K}\mathrm{vol}(\mathcal{R}_s) \xrightarrow{\text{a.s.}} \mathbb{E}[d_1] = \frac{\gamma-1}{\gamma-2},$$

$$\frac{1}{n/\binom{K}{r}}\mathrm{vol}(\cap_{k' \in \mathcal{S}\setminus\{s\}}\mathcal{M}_{k'}) \xrightarrow{\text{a.s.}} \mathbb{E}[d_1] = \frac{\gamma-1}{\gamma-2}.$$

$$\rho n \xrightarrow{\text{a.s.}} \frac{1}{\mathbb{E}[d_1]} = \frac{\gamma-2}{\gamma-1}.$$

Plugging into (7.14), we have $\lim_{n\to\infty} \frac{n}{\tilde{g}}\mathbb{E}\big[|\mathcal{Z}^s_{\mathcal{S}\setminus\{s\}}|\big] = \left(\frac{\gamma-1}{\gamma-2}\right)$. Therefore, $\frac{n}{\tilde{g}}|\mathcal{Z}^s_{\mathcal{S}\setminus\{s\}}| \xrightarrow{\text{a.s.}} \left(\frac{\gamma-1}{\gamma-2}\right)$ for any $s \in \mathcal{S}$ and $\mathcal{S} \subseteq [K]$. Putting all together, we have for $\gamma > 2$,

$$\begin{aligned}
\lim_{n\to\infty} n\mathbb{E}[L^{\mathsf{UC}}_{A_{\mathsf{C}}}(\mathcal{S})] &= \lim_{n\to\infty} \frac{n}{n^2}\sum_{s\in\mathcal{S}} \mathbb{E}\big[|\mathcal{Z}^s_{\mathcal{S}\setminus\{s\}}|\big] \\
&= \frac{1}{K\binom{K}{r}} \lim_{n\to\infty} \sum_{s\in\mathcal{S}} \frac{n}{\tilde{g}}\mathbb{E}\big[|\mathcal{Z}^s_{\mathcal{S}\setminus\{s\}}|\big] \\
&= \frac{r+1}{K\binom{K}{r}}\left(\frac{\gamma-1}{\gamma-2}\right).
\end{aligned}$$

Therefore, denoted by $L^{\mathsf{UC}}_{A_{\mathsf{C}}}$ the total uncoded communication load, we have

$$\begin{aligned}
\lim_{n\to\infty} n\mathbb{E}[L^{\mathsf{UC}}_{A_{\mathsf{C}}}] &= \lim_{n\to\infty} \sum_{\substack{\mathcal{S}\subseteq[K] \\ n|\mathcal{S}|=r+1}} \mathbb{E}[L^{\mathsf{UC}}_{A_{\mathsf{C}}}(\mathcal{S})] \\
&= \binom{K}{r+1}\frac{r+1}{K\binom{K}{r}}\left(\frac{\gamma-1}{\gamma-2}\right) \\
&= (1-\frac{r}{K})\left(\frac{\gamma-1}{\gamma-2}\right).
\end{aligned}$$

For the coded scheme, we have

$$\begin{aligned}
\lim_{n\to\infty} n\mathbb{E}[L^{\mathsf{C}}_{A_{\mathsf{C}}}(\mathcal{S})] &= \lim_{n\to\infty} \frac{n}{n^2 r} \sum_{s\in\mathcal{S}} \mathbb{E}\left[\max_{k\in\mathcal{S}\setminus\{s\}} |\mathcal{Z}^k_{\mathcal{S}\setminus\{k\}}|\right] \\
&\le \lim_{n\to\infty} \frac{n(r+1)}{n^2 r} \mathbb{E}\left[\max_{s\in\mathcal{S}} |\mathcal{Z}^s_{\mathcal{S}\setminus\{s\}}|\right] \\
&= \frac{r+1}{rK\binom{K}{r}} \left(\frac{\gamma-1}{\gamma-2}\right). \qquad (7.15)
\end{aligned}$$

The last equality follows the fact that $\frac{n}{\bar{g}} \max_{s\in\mathcal{S}} |\mathcal{Z}^s_{\mathcal{S}\setminus\{s\}}| \xrightarrow{\text{a.s.}} \left(\frac{\gamma-1}{\gamma-2}\right)$, since $\frac{n}{\bar{g}} |\mathcal{Z}^s_{\mathcal{S}\setminus\{s\}}|$ converges almost surely for any $s\in\mathcal{S}$. Plugging into (7.15), the expected coded load is

$$\begin{aligned}
\lim_{n\to\infty} n\mathbb{E}[L^{\mathsf{C}}_{A_{\mathsf{C}}}] &= \lim_{n\to\infty} n \sum_{\substack{\mathcal{S}\subseteq[K]\\ |\mathcal{S}|=r+1}} \mathbb{E}[L^{\mathsf{C}}_{A_{\mathsf{C}}}(\mathcal{S})] \\
&\le \binom{K}{r+1} \frac{r+1}{rK\binom{K}{r}} \left(\frac{\gamma-1}{\gamma-2}\right) \\
&= \frac{1}{r}\left(1-\frac{r}{K}\right)\left(\frac{\gamma-1}{\gamma-2}\right),
\end{aligned}$$

which yields

$$\lim_{n\to\infty} \frac{nL^*(r)}{\left(\frac{\gamma-1}{\gamma-2}\right)} \le \lim_{n\to\infty} \frac{n\mathbb{E}[L^{\mathsf{C}}_{A_{\mathsf{C}}}]}{\left(\frac{\gamma-1}{\gamma-2}\right)} \le \frac{1}{r}\left(1-\frac{r}{K}\right).$$

Comparing the coded load with uncoded load proves the achievability of gain r for the power law model.

Proposition 7.1 (Kolmogorov's Strong Law of Large Numbers [196, 197]) *Let X_1, $X_2, \cdots, X_n, \cdots$ be a sequence of independent random variables with $|\mathbb{E}[X_n]| < \infty$ for $n \ge 1$. Then*

$$\frac{1}{n}\sum_{i=1}^{n} \left(X_i - \mathbb{E}[X_i]\right) \xrightarrow{a.s.} 0,$$

if one of the following conditions are satisfied:

(1) X_i's are identically distributed,

(2) $\forall n$, $var(X_n) < \infty$ and $\sum_{n=1}^{\infty} \frac{var(X_n)}{n^2} < \infty$.

7.7 Experiments over Amazon EC2 Clusters

In this section, we demonstrate the practical impact of our proposed coded scheme via experiments over Amazon EC2 clusters. We first present our implementation choices and experimental scenarios. Then, we discuss the results and provide some remarks. Implementation codes are available at [198].

7.7.1 Implementation Details

We implement one iteration of the popular PageRank algorithm (Example 7.1), for a real-world graph as well as artificially generated graphs. For real-world dataset, we use TheMarker Cafe Dataset [183]. For generating artificial graph datasets, we use the Erdös-Rényi model, where each edge in the graph is present with probability p. We consider the following three scenarios:

- **Scenario 1**: We use a subgraph of size $n = 69360$ of TheMarker Cafe Dataset [183]. The computing cluster consists of $K = 6$ servers and one master with communication bandwidth of 100 Mbps at each server.
- **Scenario 2**: We generate a graph using the Erdös-Rényi model with $n = 12600$ vertices and $p = 0.3$. The computing cluster consists of $K = 10$ servers and one master with communication bandwidth of 100 Mbps at each server.
- **Scenario 3**: We generate a graph using the Erdös-Rényi model with $n = 90090$ vertices and $p = 0.01$. The computing cluster consists of $K = 15$ servers and one

master with communication bandwidth of 100 Mbps at each server.

For each scenario, we carry out PageRank implementation for different values of the computation load r. The case of $r = 1$ corresponds to the conventional PageRank implementation, where each vertex $i \in \mathcal{V} = [n]$ is stored at exactly one server and $\mathcal{M}_k = \mathcal{R}_k$ for each server $k \in [K]$, i.e. the Map and Reduce tasks associated with any vertex i take place in the same server. For $r > 1$, we increase the computation load until the overall execution time starts increasing.

We now describe our implementation choices. We use Python with `mpi4py` package. In all of our experiments, master is of type r4.large and servers are of type m4.large. For Scenario 2 and Scenario 3, we use a sample from the Erdös-Rényi model. This process is carried out using a c4.8xlarge server instance. For each scenario, the graphs are processed and subgraph allocation is done as a pre-processing step. For $r = 1$, the graph is partitioned into smaller instances which have equal numbers of vertices. Each such partition consists of two Python `lists`, one that consists of the vertices that will be Mapped by the corresponding server, and the other one that consists of the neighborhood information of each vertex to be Mapped. The position of the neighborhood `tuple` in the neighborhood `list` is same as the position of the corresponding vertex in the vertex `list`, so that one can iterate over the two together during the Map stage. For $r > 1$, the graph is divided into $\binom{K}{r}$ batches, where each batch consists of equal numbers of vertices. Then each batch is included in the subgraph of the corresponding set of r servers. This way, we get a a computation load of r.

The overall execution consists of the following phases:

(1) **Map**: Without loss of generality, the rank for each vertex is initialized to $\frac{1}{n}$. Each server goes over its subgraph and Maps the rank associated with a vertex to intermediate values that are required by the neighboring vertices during the Reduce

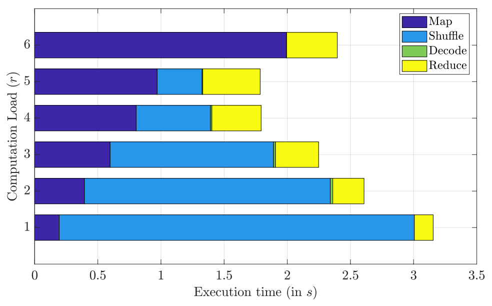
stage. Each intermediate value consists of key-value pair, where the key is an integer storing the vertex ID, while the value is a real number storing the associated value. Based on the vertex ID, the intermediate value is associated with the partition where the vertex is Reduced, which is obtained by hashing the vertex ID. For each partition, a separate `list` is created for storing keys and values.

(2) **Encode/Pack**: In conventional PageRank, no encoding is done as the transfer of intermediate values is done directly. For $r > 1$, coded multicast packets are created using the proposed encoding scheme. Transmission data is serialized before Shuffling.

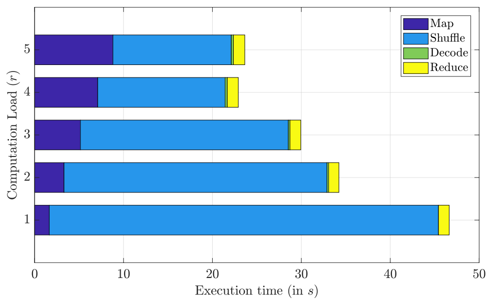
(3) **Shuffle**: At any time, only one server is allowed to use the network for transmission. In conventional PageRank, each server unicasts its message to different servers, while for $r > 1$, the communication takes place in multicast groups. For any multicast group, each server takes its turn to broadcast its message to all the remaining servers in the group.

(4) **Unpack/Decode**: The messages received during the Shuffle phase are de-serialized. For $r > 1$, each server decodes the coded packets received from other servers in accordance with the proposed coded scheme to recover the intermediate values. After the decoding phase, all intermediate values that are needed for Reduce phase are available at the servers.

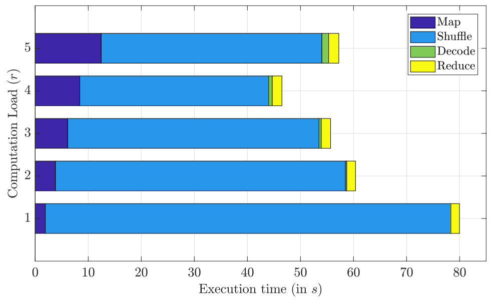
(5) **Reduce**: Each server goes over its set of vertices that it needs to Reduce and updates the corresponding PageRank values. In conventional PageRank, for any vertex $i \in \mathcal{V}$, the Map and Reduce operations associated with it are done at the same server. Therefore, no further data transmission is needed to communicate the updated ranks for the Map phase in next iteration. In the proposed coded



(a) Scenario 1



(b) Scenario 2



(c) Scenario 3

Figure 7.8: Overall execution times for distributed PageRank implementation for different computation load for the three scenarios.

scheme, message passing is done in order to transmit the updated PageRanks to the Mappers.

Next, we discuss the results of our experiments.

7.7.2 Experimental Results

We now present the results from our experiments. The overall execution times for the three scenarios have been presented in Figure 7.8.[2] We make the following observations from the results:

- As demonstrated in Figure 7.8(a), maximum gain for Scenario 1 is obtained with a computation load of $r = 5$. Our proposed scheme achieves a speedup of 43.4%

[2]The Map time includes the time spent in Encode/Pack stage, while the Unpack stage is combined with Reduce phase.

over conventional PageRank implementation ($r = 1$) and a speedup of 25.5% over the single server implementation ($r = 6$).

- For Scenarios 2 and 3, the optimal gain is obtained for $r = 4$, after which the overall execution time increases due to saturation of gain in Shuffling time and large Map time. As demonstrated by Figure 7.8(b) and Figure 7.8(c), our proposed scheme achieves speedups of 50.8% and 41.8% for Scenarios 2 and 3 respectively, in comparison to the conventional PageRank.

- As demonstrated by Figure 7.8, Shuffle phase dominates the overall execution time in the naive implementation of PageRank. By increasing the computation load, our proposed coded scheme leverages extra computing in the Map phase to slash the Shuffle phase, thus speeding up the overall execution time.

- Theoretically, we demonstrated that by increasing the computation load by r, we slash the expected communication load in Shuffle phase by nearly r. Here, we empirically observe that due to large size of the graph model, we have a similar trade-off between computation load and communication load for each sample of the graph model as well.

- While the Map phase increases almost linearly with r, the overall gain begins to saturate, since the Shuffle phase does not decrease linearly with r. This is because as we increase r, the overheads in multicast data transmissions increase and start to dominate the overall Shuffling time. Furthermore, unicasting one packet is smaller than the time for broadcasting the same packet to multiple servers [173].

Remark 7.14 *The overall execution time can be approximated as follows:*

$$T_{Total}(r) \approx rT_{Map} + T_{Shuffle}/r + T_{Reduce}, \tag{7.16}$$

where T_{Map}, $T_{Shuffle}$ and T_{Reduce} are the Map, Shuffle and Reduce times for the naive MapReduce implementation. For selecting the computation load for coded implementation, one heuristic [113] is to choose r that is the nearest integer to the minimizer r^* of (7.16) where

$$r^* = \sqrt{\frac{T_{Shuffle}}{T_{Map}}} = \arg\min_r T_{Total}(r).$$

For instance, in Scenario 2, $T_{Map} = 1.649$, $T_{Shuffle} = 43.78$ and $r^* = 5.15$. As demonstrated by Figure 7.8(b), a computation load of $r = 5$ gives close to the optimal performance attained at $r = 4$.

7.8 Concluding Remarks

We described a mathematical model for graph based MapReduce computations and demonstrated how coding theoretic strategies can be employed to substantially reduce the communication load in distributed graph analytics. Our results reveal that an inverse-linear trade-off exists between computation load and communication load in distributed graph processing. This trade-off can be used to leverage additional computing resources and capabilities to alleviate the costly communication bottleneck in distributed graph processing systems.

As a key contribution of this work, we developed a novel coding scheme that systematically injects structured redundancy in the computation phase to enable coded multicasting opportunities during message exchange between servers, reducing the communication load substantially in large-scale graph processing. For theoretical analysis, we considered random graph models, and proved that our proposed scheme enables an asymptotically inverse-linear trade-off between computation load and average normalized communication load for two popular random graph models – Erdös-Rényi model, and power law model. Furthermore, for the Erdös-Rényi model, we provided proof for a matching converse, showing the optimality of our proposed scheme. We also carried out experiments over Amazon EC2 clusters to corroborate our claims using real-world as well as artificial graphs, demonstrating speedups of up to 50.8% in the overall execution time of PageRank over the conventional approach. Additionally, we specialized our coded scheme and extended our theoretical results to two other random graph models – random bi-partite model, and stochastic block model. Our specialized schemes asymptotically enable inverse-linear trade-offs between computation and communication loads in distributed graph processing for these popular random graph models as well. We complemented the achievability results with converse bounds for both of these models.

Lastly, we note that we focused on subgraph allocation and Reduce allocation schemes that are oblivious to graph realizations. Our motivation came from popular graph processing frameworks such as Pregel [19], where partitioning of graphs is solely based on the vertex ID and not on the vertex neighborhood density. Also, designing subgraph allocation, Reduce allocation and Shuffling schemes for characterizing the minimum communication load in (7.2) is NP-hard in general. It might, however, be an interesting future direction to explore the development of coded schemes that allocate resources *after* looking at the graph.

Bibliography

[1] B. McMahan and D. Ramage, "Federated learning: Collaborative machine learning without centralized training data." https://ai.googleblog.com/2017/04/federated-learning-collaborative.html, 2017. Accessed: 2019-09-13.

[2] T. Li, A. K. Sahu, A. Talwalkar, and V. Smith, *Federated learning: Challenges, methods, and future directions*, *IEEE Signal Processing Magazine* **37** (2020), no. 3 50–60.

[3] P. Kairouz, H. B. McMahan, B. Avent, A. Bellet, M. Bennis, A. N. Bhagoji, K. Bonawitz, Z. Charles, G. Cormode, R. Cummings, *et. al.*, *Advances and open problems in federated learning*, *arXiv preprint arXiv:1912.04977* (2019).

[4] A. Reisizadeh, A. Mokhtari, H. Hassani, A. Jadbabaie, and R. Pedarsani, *Fedpaq: A communication-efficient federated learning method with periodic averaging and quantization*, in *International Conference on Artificial Intelligence and Statistics*, pp. 2021–2031, 2020.

[5] A. Reisizadeh, I. Tziotis, H. Hassani, A. Mokhtari, and R. Pedarsani, *Straggler-resilient federated learning: Leveraging the interplay between statistical accuracy and system heterogeneity*, *arXiv preprint arXiv:2012.14453* (2020).

[6] A. Reisizadeh, F. Farnia, R. Pedarsani, and A. Jadbabaie, *Robust federated learning: The case of affine distribution shifts*, *Advances in Neural Information Processing Systems* **33** (2020).

[7] J. Konečnỳ, H. B. McMahan, F. X. Yu, P. Richtárik, A. T. Suresh, and D. Bacon, *Federated learning: Strategies for improving communication efficiency*, *arXiv preprint arXiv:1610.05492* (2016).

[8] S. P. Karimireddy, S. Kale, M. Mohri, S. J. Reddi, S. U. Stich, and A. T. Suresh, *Scaffold: Stochastic controlled averaging for on-device federated learning*, *arXiv preprint arXiv:1910.06378* (2019).

[9] K. Pei, Y. Cao, J. Yang, and S. Jana, *Deepxplore: Automated whitebox testing of deep learning systems*, in *proceedings of the 26th Symposium on Operating Systems Principles*, pp. 1–18, 2017.

[10] D. Hendrycks and T. Dietterich, *Benchmarking neural network robustness to common corruptions and perturbations*, *arXiv preprint arXiv:1903.12261* (2019).

[11] A. Robey, H. Hassani, and G. J. Pappas, *Model-based robust deep learning*, *arXiv preprint arXiv:2005.10247* (2020).

[12] J. Dean and S. Ghemawat, *Mapreduce: simplified data processing on large clusters*, *Communications of the ACM* **51** (2008), no. 1 107–113.

[13] M. Zaharia, M. Chowdhury, M. J. Franklin, S. Shenker, and I. Stoica, *Spark: cluster computing with working sets*, *HotCloud* **10** (2010) 10–10.

[14] M. Zaharia, A. Konwinski, A. D. Joseph, R. H. Katz, and I. Stoica, *Improving mapreduce performance in heterogeneous environments.*, in *OSDI*, vol. 8, p. 7, 2008.

[15] Q. Pu, G. Ananthanarayanan, P. Bodik, S. Kandula, A. Akella, P. Bahl, and I. Stoica, *Low latency geo-distributed data analytics*, *ACM SIGCOMM Computer Communication Review* **45** (2015), no. 4 421–434.

[16] A. Reisizadeh, S. Prakash, R. Pedarsani, and A. S. Avestimehr, *Codedreduce: A fast and robust framework for gradient aggregation in distributed learning*, *arXiv preprint arXiv:1902.01981* (2019).

[17] S. Prakash, A. Reisizadeh, R. Pedarsani, and A. S. Avestimehr, *Coded computing for distributed graph analytics*, *IEEE Transactions on Information Theory* **66** (2020), no. 10 6534–6554.

[18] T. Lin, C. Jin, and M. I. Jordan, *On gradient descent ascent for nonconvex-concave minimax problems*, *arXiv preprint arXiv:1906.00331* (2019).

[19] G. Malewicz, M. H. Austern, A. J. Bik, J. C. Dehnert, I. Horn, N. Leiser, and G. Czajkowski, *Pregel: a system for large-scale graph processing*, *SIGMOD* (2010).

[20] J. E. Gonzalez, Y. Low, H. Gu, D. Bickson, and C. Guestrin, *Powergraph: distributed graph-parallel computation on natural graphs.*, in *OSDI*, 2012.

[21] Y. Low, J. Gonzalez, A. Kyrola, D. Bickson, C. Guestrin, and J. M. Hellerstein, *Distributed graphlab: A framework for machine learning in the cloud*, *PVLDB* **5** (2012), no. 8 716–727.

[22] R. R. McCune, T. Weninger, and G. Madey, *Thinking like a vertex: a survey of vertex-centric frameworks for large-scale distributed graph processing*, *ACM Computing Surveys* (2015).

[23] A. Lumsdaine, D. Gregor, B. Hendrickson, and J. Berry, *Challenges in parallel graph processing*, *Parallel Processing Letters* (2007).

[24] R. Chen, X. Ding, P. Wang, H. Chen, B. Zang, and H. Guan, *Computation and communication efficient graph processing with distributed immutable view*, *HPDC* (2014).

[25] T. Li, A. K. Sahu, A. Talwalkar, and V. Smith, *Federated learning: Challenges, methods, and future directions*, *arXiv preprint arXiv:1908.07873* (2019).

[26] K. Bonawitz, H. Eichner, W. Grieskamp, D. Huba, A. Ingerman, V. Ivanov, C. Kiddon, J. Konecny, S. Mazzocchi, H. B. McMahan, *et. al.*, *Towards federated learning at scale: System design*, *arXiv preprint arXiv:1902.01046* (2019).

[27] L. Huang, Y. Yin, Z. Fu, S. Zhang, H. Deng, and D. Liu, *Loadaboost: Loss-based adaboost federated machine learning on medical data*, *arXiv preprint arXiv:1811.12629* (2018).

[28] V. Smith, C.-K. Chiang, M. Sanjabi, and A. S. Talwalkar, *Federated multi-task learning*, in *Advances in Neural Information Processing Systems*, pp. 4424–4434, 2017.

[29] S. Samarakoon, M. Bennis, W. Saady, and M. Debbah, *Distributed federated learning for ultra-reliable low-latency vehicular communications*, *arXiv preprint arXiv:1807.08127* (2018).

[30] H. B. McMahan, E. Moore, D. Ramage, S. Hampson, *et. al.*, *Communication-efficient learning of deep networks from decentralized data*, *arXiv preprint arXiv:1602.05629* (2016).

[31] N. Guha, A. Talwlkar, and V. Smith, *One-shot federated learning*, *arXiv preprint arXiv:1902.11175* (2019).

[32] D. Alistarh, D. Grubic, J. Li, R. Tomioka, and M. Vojnovic, *Qsgd: Communication-efficient sgd via gradient quantization and encoding*, in *Advances in Neural Information Processing Systems*, pp. 1709–1720, 2017.

[33] F. Seide, H. Fu, J. Droppo, G. Li, and D. Yu, *1-bit stochastic gradient descent and its application to data-parallel distributed training of speech dnns*, in *Fifteenth Annual Conference of the International Speech Communication Association*, 2014.

[34] J. Bernstein, Y.-X. Wang, K. Azizzadenesheli, and A. Anandkumar, *signsgd: Compressed optimisation for non-convex problems*, *arXiv preprint arXiv:1802.04434* (2018).

[35] V. Smith, S. Forte, C. Ma, M. Takac, M. I. Jordan, and M. Jaggi, *Cocoa: A general framework for communication-efficient distributed optimization*, *arXiv preprint arXiv:1611.02189* (2016).

[36] A. Reisizadeh, A. Mokhtari, H. Hassani, and R. Pedarsani, *An exact quantized decentralized gradient descent algorithm*, *IEEE Transactions on Signal Processing* **67** (2019), no. 19 4934–4947.

[37] X. Zhang, J. Liu, Z. Zhu, and E. S. Bentley, *Compressed distributed gradient descent: Communication-efficient consensus over networks*, *arXiv preprint arXiv:1812.04048* (2018).

[38] A. Koloskova, S. U. Stich, and M. Jaggi, *Decentralized stochastic optimization and gossip algorithms with compressed communication*, .

[39] J. Wang, A. K. Sahu, Z. Yang, G. Joshi, and S. Kar, *Matcha: Speeding up decentralized sgd via matching decomposition sampling*, *arXiv preprint arXiv:1905.09435* (2019).

[40] S. U. Stich, *Local sgd converges fast and communicates little*, *arXiv preprint arXiv:1805.09767* (2018).

[41] T. Lin, S. U. Stich, K. K. Patel, and M. Jaggi, *Don't use large mini-batches, use local sgd*, *arXiv preprint arXiv:1808.07217* (2018).

[42] J. Wang and G. Joshi, *Cooperative sgd: A unified framework for the design and analysis of communication-efficient sgd algorithms*, *arXiv preprint arXiv:1808.07576* (2018).

[43] L. Bottou and O. Bousquet, *The tradeoffs of large scale learning*, in *Advances in neural information processing systems*, pp. 161–168, 2008.

[44] K. G. Murty and S. N. Kabadi, *Some np-complete problems in quadratic and nonlinear programming*, *Mathematical programming* **39** (1987), no. 2 117–129.

[45] S. Mei, Y. Bai, A. Montanari, *et. al.*, *The landscape of empirical risk for nonconvex losses*, *The Annals of Statistics* **46** (2018), no. 6A 2747–2774.

[46] A. K. Sahu, T. Li, M. Sanjabi, M. Zaheer, A. Talwalkar, and V. Smith, *On the convergence of federated optimization in heterogeneous networks*, *arXiv preprint arXiv:1812.06127* (2018).

[47] H. Yu, S. Yang, and S. Zhu, *Parallel restarted sgd with faster convergence and less communication: Demystifying why model averaging works for deep learning*, in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 33, pp. 5693–5700, 2019.

[48] A. Berahas, R. Bollapragada, N. S. Keskar, and E. Wei, *Balancing communication and computation in distributed optimization*, *IEEE Transactions on Automatic Control* (2018).

[49] A. Reisizadeh, H. Taheri, A. Mokhtari, H. Hassani, and R. Pedarsani, *Robust and communication-efficient collaborative learning*, in *Advances in Neural Information Processing Systems*, pp. 8388–8399, 2019.

[50] K. Lee, M. Lam, R. Pedarsani, D. Papailiopoulos, and K. Ramchandran, *Speeding up distributed machine learning using codes*, *IEEE Transactions on Information Theory* **64** (2017), no. 3 1514–1529.

[51] B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. y Arcas, *Communication-efficient learning of deep networks from decentralized data*, in *Artificial Intelligence and Statistics*, pp. 1273–1282, PMLR, 2017.

[52] S. U. Stich, *Local sgd converges fast and communicates little*, in *ICLR 2019 ICLR 2019 International Conference on Learning Representations*, no. CONF, 2019.

[53] C. Xie, S. Koyejo, and I. Gupta, *Asynchronous federated optimization*, *arXiv preprint arXiv:1903.03934* (2019).

[54] T. Nishio and R. Yonetani, *Client selection for federated learning with heterogeneous resources in mobile edge*, in *ICC 2019-2019 IEEE International Conference on Communications (ICC)*, pp. 1–7, IEEE, 2019.

[55] J. Wang, Q. Liu, H. Liang, G. Joshi, and H. V. Poor, *Tackling the objective inconsistency problem in heterogeneous federated optimization*, *arXiv preprint arXiv:2007.07481* (2020).

[56] F. Haddadpour, M. M. Kamani, A. Mokhtari, and M. Mahdavi, *Federated learning with compression: Unified analysis and sharp guarantees*, *arXiv preprint arXiv:2007.01154* (2020).

[57] T. Li, A. K. Sahu, M. Zaheer, M. Sanjabi, A. Talwalkar, and V. Smith, *Federated optimization in heterogeneous networks*, *arXiv preprint arXiv:1812.06127* (2018).

[58] M. Mohri, G. Sivek, and A. T. Suresh, *Agnostic federated learning*, in *International Conference on Machine Learning*, pp. 4615–4625, 2019.

[59] S. Reddi, Z. Charles, M. Zaheer, Z. Garrett, K. Rush, J. Konečnỳ, S. Kumar, and H. B. McMahan, *Adaptive federated optimization*, *arXiv preprint arXiv:2003.00295* (2020).

[60] A. Mokhtari and A. Ribeiro, *First-order adaptive sample size methods to reduce complexity of empirical risk minimization*, in *NeurIPS*, 2017.

[61] A. Mokhtari, A. Ozdaglar, and A. Jadbabaie, *Efficient nonconvex empirical risk minimization via adaptive sample size methods*, in *AISTATS*, 2019.

[62] A. Mokhtari, H. Daneshmand, A. Lucchi, T. Hofmann, and A. Ribeiro, *Adaptive Newton method for empirical risk minimization to statistical accuracy*, in *NeurIPS*, 2016.

[63] M. Eisen, A. Mokhtari, and A. Ribeiro, *Large scale empirical risk minimization via truncated adaptive Newton method*, in *AISTATS*, 2018.

[64] M. Jahani, X. He, C. Ma, A. Mokhtari, D. Mudigere, A. Ribeiro, and M. Takác, *Efficient distributed hessian free algorithm for large-scale empirical risk minimization via accumulating sample strategy*, in *AISTATS*, 2020.

[65] X. Li, K. Huang, W. Yang, S. Wang, and Z. Zhang, *On the convergence of fedavg on non-iid data*, in *International Conference on Learning Representations*, 2019.

[66] Z. Huo, Q. Yang, B. Gu, L. C. Huang, *et. al.*, *Faster on-device training using new federated momentum algorithm*, *arXiv preprint arXiv:2002.02090* (2020).

[67] G. Malinovsky, D. Kovalev, E. Gasanov, L. Condat, and P. Richtarik, *From local sgd to local fixed point methods for federated learning*, *arXiv preprint arXiv:2004.01442* (2020).

[68] S. Wang, T. Tuor, T. Salonidis, K. K. Leung, C. Makaya, T. He, and K. Chan, *Adaptive federated learning in resource constrained edge computing systems*, *IEEE Journal on Selected Areas in Communications* **37** (2019), no. 6 1205–1221.

[69] F. Zhou and G. Cong, *On the convergence properties of a k-step averaging stochastic gradient descent algorithm for nonconvex optimization*, *arXiv preprint arXiv:1708.01012* (2017).

[70] F. Haddadpour and M. Mahdavi, *On the convergence of local descent methods in federated learning*, *arXiv preprint arXiv:1910.14425* (2019).

[71] F. Haddadpour, M. M. Kamani, M. Mahdavi, and V. Cadambe, *Local sgd with periodic averaging: Tighter analysis and adaptive synchronization*, in *Advances in Neural Information Processing Systems*, pp. 11082–11094, 2019.

[72] A. K. R. Bayoumi, K. Mishchenko, and P. Richtarik, *Tighter theory for local sgd on identical and heterogeneous data*, in *International Conference on Artificial Intelligence and Statistics*, pp. 4519–4529, 2020.

[73] S. U. Stich and S. P. Karimireddy, *The error-feedback framework: Better rates for sgd with delayed gradients and compressed communication*, *arXiv preprint arXiv:1909.05350* (2019).

[74] J. Wang and G. Joshi, *Adaptive communication strategies to achieve the best error-runtime trade-off in local-update sgd*, *arXiv preprint arXiv:1810.08313* (2018).

[75] A. Koloskova, N. Loizou, S. Boreiri, M. Jaggi, and S. U. Stich, *A unified theory of decentralized sgd with changing topology and local updates*, *arXiv preprint arXiv:2003.10422* (2020).

[76] X. Liang, S. Shen, J. Liu, Z. Pan, E. Chen, and Y. Cheng, *Variance reduced local sgd with lower communication complexity*, *arXiv preprint arXiv:1912.12844* (2019).

[77] V. Vapnik, *The nature of statistical learning theory.* Springer science & business media, 2013.

[78] O. Bousquet, *Concentration inequalities and empirical processes theory applied to the analysis of learning algorithms.* PhD thesis, École Polytechnique: Department of Applied Mathematics Paris, France, 2002.

[79] P. L. Bartlett, M. I. Jordan, and J. D. McAuliffe, *Convexity, classification, and risk bounds*, *Journal of the American Statistical Association* **101** (2006), no. 473 138–156.

[80] R. Frostig, R. Ge, S. M. Kakade, and A. Sidford, *Competing with the empirical risk minimizer in a single pass*, in *Conference on learning theory*, pp. 728–763, 2015.

[81] K. Mishchenko, E. Gorbunov, M. Takáč, and P. Richtárik, *Distributed learning with compressed gradient differences*, *arXiv preprint arXiv:1901.09269* (2019).

[82] A. Reisizadeh, S. Prakash, R. Pedarsani, and A. S. Avestimehr, *Coded computation over heterogeneous clusters*, *IEEE Transactions on Information Theory* **65** (2019), no. 7 4227–4242.

[83] A. Madry, A. Makelov, L. Schmidt, D. Tsipras, and A. Vladu, *Towards deep learning models resistant to adversarial attacks*, *arXiv preprint arXiv:1706.06083* (2017).

[84] I. J. Goodfellow, J. Shlens, and C. Szegedy, *Explaining and harnessing adversarial examples*, *arXiv preprint arXiv:1412.6572* (2014).

[85] A. Shafahi, M. Najibi, Z. Xu, J. Dickerson, L. S. Davis, and T. Goldstein, *Universal adversarial training*, *arXiv preprint arXiv:1811.11304* (2018).

[86] S.-M. Moosavi-Dezfooli, A. Fawzi, O. Fawzi, and P. Frossard, *Universal adversarial perturbations*, in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 1765–1773, 2017.

[87] D. A. McAllester, *Pac-bayesian model averaging*, in *Proceedings of the twelfth annual conference on Computational learning theory*, pp. 164–170, 1999.

[88] B. Neyshabur, S. Bhojanapalli, and N. Srebro, *A pac-bayesian approach to spectrally-normalized margin bounds for neural networks*, *arXiv preprint arXiv:1707.09564* (2017).

[89] W. Wiesemann, D. Kuhn, and M. Sim, *Distributionally robust convex optimization*, *Operations Research* **62** (2014), no. 6 1358–1376.

[90] S. Shafieezadeh-Abadeh, D. Kuhn, and P. M. Esfahani, *Regularization via mass transportation*, *Journal of Machine Learning Research* **20** (2019), no. 103 1–68.

[91] A. Khaled, K. Mishchenko, and P. Richtárik, *Tighter theory for local sgd on identical and heterogeneous data*, in *The 23rd International Conference on Artificial Intelligence and Statistics (AISTATS 2020)*, 2020.

[92] S. Caldas, J. Konečny, H. B. McMahan, and A. Talwalkar, *Expanding the reach of federated learning by reducing client resource requirements*, *arXiv preprint arXiv:1812.07210* (2018).

[93] A. Reisizadeh, A. Mokhtari, H. Hassani, A. Jadbabaie, and R. Pedarsani, *Fedpaq: A communication-efficient federated learning method with periodic averaging and quantization*, *arXiv preprint arXiv:1909.13014* (2019).

[94] A. Bhowmick, J. Duchi, J. Freudiger, G. Kapoor, and R. Rogers, *Protection against reconstruction and its applications in private federated learning*, *arXiv preprint arXiv:1812.00984* (2018).

[95] R. C. Geyer, T. Klein, and M. Nabi, *Differentially private federated learning: A client level perspective*, *arXiv preprint arXiv:1712.07557* (2017).

[96] J. Li, M. Khodak, S. Caldas, and A. Talwalkar, *Differentially private meta-learning*, *arXiv preprint arXiv:1909.05830* (2019).

[97] O. Thakkar, G. Andrew, and H. B. McMahan, *Differentially private learning with adaptive clipping*, *arXiv preprint arXiv:1905.03871* (2019).

[98] J. Yang, N. Kiyavash, and N. He, *Global convergence and variance-reduced optimization for a class of nonconvex-nonconcave minimax problems*, *arXiv preprint arXiv:2002.09621* (2020).

[99] M. Nouiehed, M. Sanjabi, T. Huang, J. D. Lee, and M. Razaviyayn, *Solving a class of non-convex min-max games using iterative first order methods*, in *Advances in Neural Information Processing Systems*, pp. 14905–14916, 2019.

[100] H. Yu, R. Jin, and S. Yang, *On the linear speedup analysis of communication efficient momentum sgd for distributed non-convex optimization*, *arXiv preprint arXiv:1905.03817* (2019).

[101] B. T. Polyak, *Gradient methods for minimizing functionals*, *Zhurnal Vychislitel'noi Matematiki i Matematicheskoi Fiziki* **3** (1963), no. 4 643–653.

[102] P. L. Bartlett, D. J. Foster, and M. J. Telgarsky, *Spectrally-normalized margin bounds for neural networks*, in *Advances in Neural Information Processing Systems*, pp. 6240–6249, 2017.

[103] F. Farnia, J. M. Zhang, and D. Tse, *Generalizable adversarial training via spectral normalization*, *arXiv preprint arXiv:1811.07457* (2018).

[104] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, *et. al.*, *Tensorflow: Large-scale machine learning on heterogeneous distributed systems*, *arXiv preprint arXiv:1603.04467* (2016).

[105] Y. LeCun, *The mnist database of handwritten digits*, *http://yann. lecun. com/exdb/mnist/* (1998).

[106] A. Krizhevsky, G. Hinton, *et. al.*, *Learning multiple layers of features from tiny images*, .

[107] A. Krizhevsky, I. Sutskever, and G. E. Hinton, *Imagenet classification with deep convolutional neural networks*, in *Advances in neural information processing systems*, pp. 1097–1105, 2012.

[108] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, *Going deeper with convolutions*, in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 1–9, 2015.

[109] K. He, X. Zhang, S. Ren, and J. Sun, *Deep residual learning for image recognition*, in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016.

[110] S. Ioffe and C. Szegedy, *Batch normalization: Accelerating deep network training by reducing internal covariate shift*, *arXiv preprint arXiv:1502.03167* (2015).

[111] D. P. Kingma and J. Ba, *Adam: A method for stochastic optimization*, *arXiv preprint arXiv:1412.6980* (2014).

[112] S. Li, M. A. Maddah-Ali, and A. S. Avestimehr, *Coded MapReduce*, in *Communication, Control, and Computing (Allerton), 2015 53rd Annual Allerton Conference on*, pp. 964–971, IEEE, 2015.

[113] S. Li, M. A. Maddah-Ali, Q. Yu, and A. S. Avestimehr, *A fundamental tradeoff between computation and communication in distributed computing*, *IEEE Transactions on Information Theory* **64** (2018), no. 1 109–128.

[114] S. Li, S. Supittayapornpong, M. A. Maddah-Ali, and S. Avestimehr, *Coded terasort*, in *Parallel and Distributed Processing Symposium Workshops (IPDPSW), 2017 IEEE International*, pp. 389–398, IEEE, 2017.

[115] F. Bonomi, R. Milito, J. Zhu, and S. Addepalli, *Fog computing and its role in the internet of things*, in *Proceedings of the first edition of the MCC workshop on Mobile cloud computing*, pp. 13–16, ACM, 2012.

[116] K. Lee, M. Lam, R. Pedarsani, D. Papailiopoulos, and K. Ramchandran, *Speeding up distributed machine learning using codes*, *IEEE Transactions on Information Theory* **64** (2018), no. 3 1514–1529.

[117] L. Song and C. Fragouli, *A pliable index coding approach to data shuffling*, *arXiv preprint arXiv:1701.05540* (2017).

[118] M. Kiamari, C. Wang, and A. S. Avestimehr, *On heterogeneous coded distributed computing*, in *GLOBECOM 2017-2017 IEEE Global Communications Conference*, pp. 1–7, IEEE, 2017.

[119] M. Attia and R. Tandon, *Information theoretic limits of data shuffling for distributed learning*, *arXiv preprint arXiv:1609.05181* (2016).

[120] Y. H. Ezzeldin, M. Karmoose, and C. Fragouli, *Communication vs distributed computation: an alternative trade-off curve*, in *Information Theory Workshop (ITW), 2017 IEEE*, pp. 279–283, IEEE, 2017.

[121] S. Dutta, V. Cadambe, and P. Grover, *Short-dot: Computing large linear transforms distributedly using coded short dot products*, in *Advances In Neural Information Processing Systems*, pp. 2092–2100, 2016.

[122] R. Tandon, Q. Lei, A. G. Dimakis, and N. Karampatziakis, *Gradient coding*, *arXiv preprint arXiv:1612.03301* (2016).

[123] Q. Yu, M. A. Maddah-Ali, and A. S. Avestimehr, *Polynomial codes: an optimal design for high-dimensional coded matrix multiplication*, *arXiv preprint arXiv:1705.10464* (2017).

[124] M. Fahim, H. Jeong, F. Haddadpour, S. Dutta, V. Cadambe, and P. Grover, *On the optimal recovery threshold of coded matrix multiplication*, in *Communication, Control, and Computing (Allerton), 2017 55th Annual Allerton Conference on*, pp. 1264–1270, IEEE, 2017.

[125] K. Lee, C. Suh, and K. Ramchandran, *High-dimensional coded matrix multiplication*, in *Information Theory (ISIT), 2017 IEEE International Symposium on*, pp. 2418–2422, IEEE, 2017.

[126] S. Wang, J. Liu, N. Shroff, and P. Yang, *Fundamental limits of coded linear transform*, *arXiv preprint arXiv:1804.09791* (2018).

[127] Q. Yu, M. A. Maddah-Ali, and A. S. Avestimehr, *Straggler mitigation in distributed matrix multiplication: Fundamental limits and optimal coding*, *arXiv preprint arXiv:1801.07487* (2018).

[128] A. Reisizadeh and R. Pedarsani, *Latency analysis of coded computation schemes over wireless networks*, in *Communication, Control, and Computing (Allerton), 2017 55th Annual Allerton Conference on*, pp. 1256–1263, IEEE, 2017.

[129] K. Lee, R. Pedarsani, D. Papailiopoulos, and K. Ramchandran, *Coded computation for multicore setups*, in *Information Theory (ISIT), 2017 IEEE International Symposium on*, pp. 2413–2417, IEEE, 2017.

[130] N. S. Ferdinand and S. C. Draper, *Anytime coding for distributed computation*, in *Communication, Control, and Computing (Allerton), 2016 54th Annual Allerton Conference on*, pp. 954–960, IEEE, 2016.

[131] G. Suh, K. Lee, and C. Suh, *Matrix sparsification for coded matrix multiplication*, in *Communication, Control, and Computing (Allerton), 2017 55th Annual Allerton Conference on*, pp. 1271–1278, IEEE, 2017.

[132] M. Aliasgari, J. Kliewer, and O. Simeone, *Coded computation against straggling decoders for network function virtualization*, in *2018 IEEE International Symposium on Information Theory (ISIT)*, pp. 711–715, IEEE, 2018.

[133] Y. Yang, P. Grover, and S. Kar, *Computing linear transformations with unreliable components*, *IEEE Transactions on Information Theory* **63** (2017), no. 6 3729–3756.

[134] C. Karakus, Y. Sun, S. Diggavi, and W. Yin, *Straggler mitigation in distributed optimization through data encoding*, in *Advances in Neural Information Processing Systems*, pp. 5440–5448, 2017.

[135] A. Severinson, E. Rosnes, *et. al.*, *Block-diagonal and LT codes for distributed computing with straggling servers*, *arXiv preprint arXiv:1712.08230* (2017).

[136] M. F. Aktas, P. Peng, and E. Soljanin, *Effective straggler mitigation: which clones should attack and when?*, *ACM SIGMETRICS Performance Evaluation Review* **45** (2017), no. 2 12–14.

[137] D. Wang, G. Joshi, and G. Wornell, *Using straggler replication to reduce latency in large-scale parallel computing*, *ACM SIGMETRICS Performance Evaluation Review* **43** (2015), no. 3 7–11.

[138] M. Rudelson and R. Vershynin, *Non-asymptotic theory of random matrices: extreme singular values*, in *Proceedings of the International Congress of Mathematicians 2010 (ICM 2010) (In 4 Volumes) Vol. I: Plenary Lectures and Ceremonies Vols. II–IV: Invited Lectures*, pp. 1576–1602, World Scientific, 2010.

[139] G. Liang and U. C. Kozat, *Tofec: achieving optimal throughput-delay trade-off of cloud storage using erasure codes*, in *IEEE INFOCOM 2014-IEEE Conference on Computer Communications*, pp. 826–834, IEEE, 2014.

[140] S. Dutta, V. Cadambe, and P. Grover, *Coded convolution for parallel and distributed computing within a deadline*, *arXiv preprint arXiv:1705.03875* (2017).

[141] L. Dalcín, R. Paz, and M. Storti, *MPI for Python*, *Journal of Parallel and Distributed Computing* **65** (2005), no. 9 1108–1115.

[142] J. Dean and L. A. Barroso, *The tail at scale*, *Communications of the ACM* **56** (2013), no. 2 74–80.

[143] S. Li, M. A. Maddah-Ali, and A. S. Avestimehr, *Coding for distributed fog computing*, *IEEE Communications Magazine* **55** (2017), no. 4 34–40.

[144] D. J. MacKay and D. J. Mac Kay, *Information theory, inference and learning algorithms.* Cambridge university press, 2003.

[145] A. Mallick, M. Chaudhari, and G. Joshi, *Rateless codes for near-perfect load balancing in distributed matrix-vector multiplication*, *arXiv preprint arXiv:1804.10331* (2018).

[146] S. Su, J. Li, Q. Huang, X. Huang, K. Shuang, and J. Wang, *Cost-efficient task scheduling for executing large programs in the cloud*, *Parallel Computing* **39** (2013), no. 4-5 177–188.

[147] E. Deelman, G. Singh, M. Livny, B. Berriman, and J. Good, *The cost of doing science on the cloud: the montage example*, in *High Performance Computing, Networking, Storage and Analysis, 2008. SC 2008. International Conference for*, pp. 1–12, IEEE, 2008.

[148] D. Kondo, B. Javadi, P. Malecot, F. Cappello, and D. P. Anderson, *Cost-benefit analysis of cloud computing versus desktop grids.*, in *IPDPS*, vol. 9, pp. 1–12, 2009.

[149] S. Yi, A. Andrzejak, and D. Kondo, *Monetary cost-aware checkpointing and migration on amazon cloud spot instances*, *IEEE Transactions on Services Computing* **5** (2012), no. 4 512–524.

[150] M. Malawski, G. Juve, E. Deelman, and J. Nabrzyski, *Algorithms for cost-and deadline-constrained provisioning for scientific workflow ensembles in iaas clouds*, *Future Generation Computer Systems* **48** (2015) 1–18.

[151] https://aws.amazon.com/ec2/pricing/.

[152] O. Dekel, R. Gilad-Bachrach, O. Shamir, and L. Xiao, *Optimal Distributed Online Prediction Using Mini-Batches*, *Journal of Machine Learning Research* **13** (2012), no. Jan 165–202.

[153] M. Zinkevich, M. Weimer, A. J. Smola, and L. Li, *Parallelized stochastic gradient descent.*, in *NIPS*, vol. 4, p. 4, Citeseer, 2010.

[154] J. Chen, X. Pan, R. Monga, S. Bengio, and R. Jozefowicz, *Revisiting distributed synchronous SGD*, *arXiv preprint arXiv:1604.00981* (2016).

[155] B. Recht, C. Re, S. Wright, and F. Niu, *Hogwild: A Lock-Free Approach to Parallelizing Stochastic Gradient Descent*, in *Advances in Neural Information Processing Systems*, pp. 693–701, 2011.

[156] J. Dean, G. Corrado, R. Monga, K. Chen, M. Devin, M. Mao, A. Senior, P. Tucker, K. Yang, Q. V. Le, *et. al.*, *Large scale distributed deep networks*, in *Advances in neural information processing systems*, pp. 1223–1231, 2012.

[157] T. M. Chilimbi, Y. Suzue, J. Apacible, and K. Kalyanaraman, *Project Adam: Building an Efficient and Scalable Deep Learning Training System.*, in *OSDI*, vol. 14, pp. 571–582, 2014.

[158] G. Cong, O. Bhardwaj, and M. Feng, *An efficient, distributed stochastic gradient descent algorithm for deep-learning applications*, in *Parallel Processing (ICPP), 2017 46th International Conference on*, pp. 11–20, IEEE, 2017.

[159] P. Patarasuk and X. Yuan, *Bandwidth optimal all-reduce algorithms for clusters of workstations*, *Journal of Parallel and Distributed Computing* **69** (2009), no. 2 117–124.

[160] P. Patarasuk and X. Yuan, *Bandwidth Efficient All-reduce Operation on Tree Topologies*, in *Parallel and Distributed Processing Symposium, 2007. IPDPS 2007. IEEE International*, pp. 1–8, IEEE, 2007.

[161] R. Thakur, R. Rabenseifner, and W. Gropp, *Optimization of Collective Communication Operations in MPICH*, *The International Journal of High Performance Computing Applications* **19** (2005), no. 1 49–66.

[162] K. Kandalla, H. Subramoni, A. Vishnu, and D. K. Panda, *Designing topology-aware collective communication algorithms for large scale infiniband clusters: Case studies with Scatter and Gather*, in *Parallel & Distributed Processing, Workshops and Phd Forum (IPDPSW), 2010 IEEE International Symposium on*, pp. 1–8, IEEE, 2010.

[163] A. Gibiansky, *Bringing HPC Techniques to Deep Learning*, tech. rep., Baidu Research, Tech. Rep., 2017, http://research. baidu. com/bringing-hpc-techniques-deep-learning/. Bingjing Zhang TESTS & CERTIFICATIONS IBM Certified Database Associate-DB2 Universal Database, 2017.

[164] A. Sergeev and M. Del Balso, *Horovod: fast and easy distributed deep learning in TensorFlow*, *arXiv preprint arXiv:1802.05799* (2018).

[165] P. H. Jin, Q. Yuan, F. Iandola, and K. Keutzer, *How to scale distributed deep learning?*, *ML Systems Workshop, NIPS* (2016).

[166] Y. Li, M. Yu, S. Li, S. Avestimehr, N. S. Kim, and A. Schwing, *Pipe-SGD: A Decentralized Pipelined SGD Framework for Distributed Deep Net Training*, in *Advances in Neural Information Processing Systems*, pp. 8056–8067, 2018.

[167] M. Yu, Z. Lin, K. Narra, S. Li, Y. Li, N. S. Kim, A. Schwing, M. Annavaram, and S. Avestimehr, *GradiVeQ: Vector Quantization for Bandwidth-Efficient Gradient Aggregation in Distributed CNN Training*, in *Advances in Neural Information Processing Systems*, pp. 5129–5139, 2018.

[168] J. Sun, T. Chen, G. B. Giannakis, Q. Yang, and Z. Yang, *Lazily aggregated quantized gradient innovation for communication-efficient federated learning*, *IEEE Transactions on Pattern Analysis and Machine Intelligence* (2020).

[169] Y. Zhao, L. Wang, W. Wu, G. Bosilca, R. Vuduc, J. Ye, W. Tang, and Z. Xu, *Efficient Communications in Training Large Scale Neural Networks*, in *Proceedings of the on Thematic Workshops of ACM Multimedia 2017*, pp. 110–116, ACM, 2017.

[170] G. Ananthanarayanan, A. Ghodsi, S. Shenker, and I. Stoica, *Effective straggler mitigation: Attack of the clones*, in *Presented as part of the 10th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 13)*, pp. 185–198, 2013.

[171] D. Wang, G. Joshi, and G. Wornell, *Efficient task replication for fast response times in parallel computation*, in *ACM SIGMETRICS Performance Evaluation Review*, vol. 42, pp. 599–600, ACM, 2014.

[172] N. B. Shah, K. Lee, and K. Ramchandran, *When Do Redundant Requests Reduce Latency?*, *IEEE Transactions on Communications* **64** (2016), no. 2 715–722.

[173] K. Lee, M. Lam, R. Pedarsani, D. Papailiopoulos, and K. Ramchandran, *Speeding Up Distributed Machine Learning Using Codes*, *IEEE Transactions on Information Theory* **64** (2018), no. 3 1514–1529.

[174] M. Ye and E. Abbe, *Communication-Computation Efficient Gradient Coding*, in *Proceedings of the 35th International Conference on Machine Learning*, vol. 80, pp. 5610–5619, 10–15 Jul, 2018.

[175] Q. Yu, S. Li, N. Raviv, M. Kalan, M. Soltanolkotabi, and A. S. Avestimehr, *Lagrange Coded Computing: Optimal Design for Resiliency, Security and Privacy*, *To appear in Proceedings of 2019 AISTATS* (2019).

[176] R. Tandon, Q. Lei, A. G. Dimakis, and N. Karampatziakis, *Gradient Coding: Avoiding Stragglers in Distributed Learning* , in *International Conference on Machine Learning*, pp. 3368–3376, 2017.

[177] J. D. Lee, M. Simchowitz, M. I. Jordan, and B. Recht, *Gradient descent only converges to minimizers*, in *Conference on learning theory*, pp. 1246–1257, PMLR, 2016.

[178] A. Reisizadeh, S. Prakash, R. Pedarsani, and S. Avestimehr, *Coded computation over heterogeneous clusters*, in *IEEE International Symposium on Information Theory (ISIT), 2017*, pp. 2408–2412.

[179] S. Li, S. M. M. Kalan, A. S. Avestimehr, and M. Soltanolkotabi, *Near-optimal straggler mitigation for distributed gradient methods*, in *2018 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, pp. 857–866, IEEE, 2018.

[180] I. Guyon, J. Li, T. Mader, P. A. Pletscher, G. Schneider, and M. Uhr, *Competitive baseline methods set new standards for the NIPS 2003 feature selection benchmark*, *Pattern recognition letters* **28** (2007), no. 12 1438–1444.

[181] A. Krizhevsky, V. Nair, and G. Hinton, *Cifar-10 (canadian institute for advanced research)*, .

[182] M. Isard, M. Budiu, Y. Yu, A. Birrell, and D. Fetterly, *Dryad: distributed data-parallel programs from sequential building blocks*, *EuroSys* (2007).

[183] M. Fire, L. Tenenboim, O. Lesser, R. Puzis, L. Rokach, and Y. Elovici, *Link prediction in social networks using computationally efficient topological features*, in *IEEE Third International Confernece on Social Computing (SocialCom)*, pp. 73–80, IEEE, 2011.

[184] S. Li, Q. Yu, M. A. Maddah-Ali, and A. S. Avestimehr, *A scalable framework for wireless distributed computing*, *IEEE/ACM Transactions on Networking* (2017).

[185] S. Li, M. A. Maddah-Ali, and A. S. Avestimehr, *Coded distributed computing: Straggling servers and multistage dataflows*, in *Communication, Control, and Computing (Allerton), 2016 54th Annual Allerton Conference on*, pp. 164–171, IEEE, 2016.

[186] K. Konstantinidis and A. Ramamoorthy, *Leveraging coding techniques for speeding up distributed computing*, *arXiv preprint arXiv:1802.03049* (2018).

[187] S. Li, M. A. Maddah-Ali, and A. S. Avestimehr, *Compressed coded distributed computing*, *arXiv preprint arXiv:1805.01993* (2018).

[188] M. A. Attia and R. Tandon, *Near optimal coded data shuffling for distributed learning*, *arXiv preprint arXiv:1801.01875* (2018).

[189] J. Chung, K. Lee, R. Pedarsani, D. Papailiopoulos, and K. Ramchandran, *Ubershuffle: Communication-efficient data shuffling for sgd via coding theory*, *NIPS Workshop on ML Systems* (2017).

[190] J. Lin and M. Schatz, *Design patterns for efficient graph algorithms in mapreduce*, *MLG Workshop* (2010).

[191] L. Page, S. Brin, R. Motwani, and T. Winograd, *The pagerank citation ranking: Bringing order to the web.*, Tech. Rep. 1999-66, Stanford InfoLab, 1999.

[192] W. Xing and A. Ghorbani, *Weighted pagerank algorithm*, in *Communication Networks and Services Research, 2004. Proceedings. Second Annual Conference on*, pp. 305–314, IEEE, 2004.

[193] E. Dahlhaus, D. S. Johnson, C. H. Papadimitriou, P. D. Seymour, and M. Yannakakis, *The complexity of multiway cuts*, *STOC* (1992).

[194] Z. Khayyat, K. Awara, A. Alonazi, H. Jamjoom, D. Williams, and P. Kalnis, *Mizan: a system for dynamic load balancing in large-scale graph processing*, in *Proceedings of the 8th ACM European Conference on Computer Systems*, pp. 169–182, 2013.

[195] F. Chung and L. Lu, *The average distance in a random graph with given expected degrees*, *Internet Mathematics* **1** (2004), no. 1 91–113.

[196] R. P. E., *Sen, p. k.; singer, j. m.: Large sample methods in statistics. an introduction with applications. chapman & hall, new york-london 1993, xii, 382pp., £35.00, isbn 0–412–04221–5*, *Biometrical Journal* **36** no. 5 602–602, [https://onlinelibrary.wiley.com/doi/pdf/10.1002/bimj.4710360511].

[197] M. Loeve, *Probability Theory I*. Graduate Texts in Mathematics. Springer New York, 1977.

[198] `https://github.com/AvestimehrResearchGroup/Coded-PageRank`.

Appendix A

Supplements to Chapter 2

A.1 Proof of Theorem 2.1

We first introduce some additional notations which will be used throughput the proofs.

Additional notations. For each period $k = 0, 1, \cdots, K-1$ and iteration $t = 0, 1, \cdots, \tau-1$ we denote

$$\begin{aligned} \mathbf{w}_{k+1} &:= \mathbf{w}_k + \frac{1}{r} \sum_{i \in \mathcal{S}_k} Q\left(\mathbf{w}_{k,\tau}^{(i)} - \mathbf{w}_k\right), \\ \widehat{\mathbf{w}}_{k+1} &:= \mathbf{w}_k + \frac{1}{n} \sum_{i \in [n]} Q\left(\mathbf{w}_{k,\tau}^{(i)} - \mathbf{w}_k\right), \\ \overline{\mathbf{w}}_{k,t} &:= \frac{1}{n} \sum_{i \in [n]} \mathbf{w}_{k,t}^{(i)}. \end{aligned} \tag{A.1}$$

We begin the proof of Theorem 2.1 by noting a few key observations. Based on the above notations and the assumptions we made earlier, the optimality gap of the parameter server's model at period k, i.e. $\mathbb{E}\|\mathbf{w}_{k+1} - \mathbf{w}^*\|^2$, can be decomposed as stated in the following lemma.

Lemma A.1 *Consider any period* $k = 0, \cdots, K-1$ *and the sequences* $\{\mathbf{w}_{k+1}, \widehat{\mathbf{w}}_{k+1}, \overline{\mathbf{w}}_{k,\tau}\}$

generated by the **`FedPAQ`** *method in Algorithm 2.1. If Assumption 2.1 holds, then*

$$\mathbb{E}\|\mathbf{w}_{k+1} - \mathbf{w}^*\|^2 = \mathbb{E}\|\mathbf{w}_{k+1} - \widehat{\mathbf{w}}_{k+1}\|^2 + \mathbb{E}\left\|\widehat{\mathbf{w}}_{k+1} - \overline{\mathbf{w}}_{k,\tau}\right\|^2 + \mathbb{E}\left\|\overline{\mathbf{w}}_{k,\tau} - \mathbf{w}^*\right\|^2, \tag{A.2}$$

where the expectation is with respect to all sources of randomness.

Proof: See Section A.1.1. ■

In the following three lemmas, we characterize each of the terms in the right-hand side (RHS) of (A.2).

Lemma A.2 *Consider the sequence of local updates in the* **`FedPAQ`** *method in Algorithm 2.1 and let Assumptions 2.2, 2.3 and 2.4 hold. The optimality gap for the average model at the end of period k, i.e. $\overline{\mathbf{w}}_{k,\tau}$, relates to that of the initial model of the k-th period $\mathbf{w}_k$ as follows:*

$$\begin{aligned}\mathbb{E}\left\|\overline{\mathbf{w}}_{k,\tau} - \mathbf{w}^*\right\|^2 \leq \left(1 + n\eta_k^2\right)\left(1 - \mu\eta_k\right)^\tau \mathbb{E}\|\mathbf{w}_k - \mathbf{w}^*\|^2 + \tau(\tau-1)^2 L^2 \frac{\sigma^2}{n} e\eta_k^2 + \tau^2 \frac{\sigma^2}{n}\eta_k^2 \\ + \tau^2(\tau-1)L^2\sigma^2 e\eta_k^4, \end{aligned} \tag{A.3}$$

for the stepsize $\eta_k \leq \min\{\frac{\mu}{L^2}, \frac{1}{L\tau}\}$.

Proof: See Section A.1.2. ■

Lemma A.3 *For the proposed* **`FedPAQ`** *method in Algorithm 2.1 with stepsize $\eta_k \leq \min\{\frac{\mu}{L^2}, \frac{1}{L\tau}\}$ and under Assumptions 2.1, 2.2, 2.3 and 2.4, we have*

$$\mathbb{E}\left\|\widehat{\mathbf{w}}_{k+1} - \overline{\mathbf{w}}_{k,\tau}\right\|^2 \leq 2\frac{q}{n}\tau^2 L^2 \eta_k^2 \mathbb{E}\|\mathbf{w}_k - \mathbf{w}^*\|^2 + 2q\tau^2\frac{\sigma^2}{n}\eta_k^2 + 2q(\tau-1)\tau^2 L^2 \frac{\sigma^2}{n} e\eta_k^4, \tag{A.4}$$

where $\widehat{\mathbf{w}}_{k+1}$ and $\overline{\mathbf{w}}_{k,\tau}$ are defined in (A.1).

Proof: See Section A.1.3. ■

Lemma A.4 *For the proposed* **`FedPAQ`** *method in Algorithm 2.1 with stepsize* $\eta_k \leq \min\{\frac{\mu}{L^2}, \frac{1}{L\tau}\}$ *and under Assumptions 2.1–2.4, we have*

$$\begin{aligned}
&\mathbb{E}\|\mathbf{w}_{k+1} - \widehat{\mathbf{w}}_{k+1}\|^2 \\
&\leq \frac{n-r}{r(n-1)} 8(1+q) \left\{ \tau^2 L^2 \eta_k^2 \mathbb{E}\|\mathbf{w}_k - \mathbf{w}^*\|^2 + \tau^2\sigma^2\eta_k^2 + (\tau-1)\tau^2 L^2 \sigma^2 e \eta_k^4 \right\}, \qquad \text{(A.5)}
\end{aligned}$$

where r denotes the number of nodes contributing in each period of the **`FedPAQ`** *method.*

Proof: See Section A.1.4. ■

Now that we have established the main building modules for proving Theorem 2.1, let us proceed with the proof by putting together the results in Lemmas A.1–A.4. That is,

$$\begin{aligned}
&\mathbb{E}\|\mathbf{w}_{k+1} - \mathbf{w}^*\|^2 \\
&\leq \mathbb{E}\|\mathbf{w}_k - \mathbf{w}^*\|^2 \left(\left(1 + n\eta_k^2\right)\left(1 - \mu\eta_k\right)^\tau + 2L^2\tau^2\eta_k^2 \left(\frac{q}{n} + \frac{n-r}{r(n-1)} 4(1+q) \right) \right) \\
&\quad + \left(1 + 2q + 8(1+q)\frac{n(n-r)}{r(n-1)} \right) \frac{\sigma^2}{n}\tau^2\eta_k^2 + L^2 \frac{\sigma^2}{n} e\tau(\tau-1)^2\eta_k^2 \\
&\quad + \left(n + 2q + 8(1+q)\frac{n(n-r)}{r(n-1)} \right) L^2 \frac{\sigma^2}{n} e(\tau-1)\tau^2\eta_k^4 \qquad \text{(A.6)}
\end{aligned}$$

Let us set the following notations:

$$\begin{aligned}
\delta_k &:= \mathbb{E}\|\mathbf{w}_k - \mathbf{w}^*\|^2, \\
C_0 &:= \left(1 + n\eta_k^2\right)\left(1 - \mu\eta_k\right)^\tau + 2L^2\tau^2\eta_k^2 \left(\frac{q}{n} + \frac{n-r}{r(n-1)} 4(1+q) \right), \\
C_1 &:= \frac{16}{\mu^2} \left(1 + 2q + 8(1+q)\frac{n(n-r)}{r(n-1)} \right) \frac{\sigma^2}{n}, \\
C_2 &:= \frac{16}{\mu^2} L^2 \frac{\sigma^2}{n} e,
\end{aligned}$$

$$C_3 := \frac{256}{\mu^4}\left(n + 2q + 8(1+q)\frac{n(n-r)}{r(n-1)}\right) L^2\frac{\sigma^2}{n}e. \tag{A.7}$$

Consider C_0, the coefficient of $\mathbb{E}\|\mathbf{w}_k - \mathbf{w}^*\|^2$ in (A.6). One can show that if the condition in (2.11) in Theorem 2.1 is satisfied, then we have $C_0 \leq 1 - \frac{1}{2}\mu\tau\eta_k$ (See Section A.1.6). Therefore, for each period $k \geq k_0$ we have

$$\delta_{k+1} \leq \left(1 - \frac{1}{2}\mu\tau\eta_k\right)\delta_k + \frac{\mu^2}{16}C_1\tau^2\eta_k^2 + \frac{\mu^2}{16}C_2\tau(\tau-1)^2\eta_k^2 + \frac{\mu^4}{256}C_3(\tau-1)\tau^2\eta_k^4. \tag{A.8}$$

Now, we substitute the stepsize $\eta_k = \frac{4\mu^{-1}}{k\tau+1}$ in (A.8) which yields

$$\delta_{k+1} \leq \left(1 - \frac{2}{k+1/\tau}\right)\delta_k + C_1\frac{1}{(k+1/\tau)^2} + C_2\frac{(\tau-1)^2}{\tau}\frac{1}{(k+1/\tau)^2} + C_3\frac{\tau-1}{\tau^2}\frac{1}{(k+1/\tau)^4}. \tag{A.9}$$

In Lemma A.5, we show the convergence analysis of such sequence. In particular, we take $k_1 = 1/\tau$, $a = C_1 + C_2(\tau-1)^2/\tau$ and $b = C_3(\tau-1)/\tau^2$ in Lemma A.5 and conclude for any $k \geq k_0$ that

$$\delta_k \leq \frac{(k_0+1/\tau)^2}{(k+1/\tau)^2}\delta_{k_0} + C_1\frac{1}{k+1/\tau} + C_2\frac{(\tau-1)^2}{\tau}\frac{1}{k+1/\tau} + C_3\frac{\tau-1}{\tau^2}\frac{1}{(k+1/\tau)^2}. \tag{A.10}$$

Finally, rearranging the terms in (A.10) yields the desired result in Theorem 2.1, that is

$$\mathbb{E}\|\mathbf{w}_k - \mathbf{w}^*\|^2 \leq \frac{(k_0\tau+1)^2}{(k\tau+1)^2}\mathbb{E}\|\mathbf{w}_{k_0} - \mathbf{w}^*\|^2 + C_1\frac{\tau}{k\tau+1} + C_2\frac{(\tau-1)^2}{k\tau+1} + C_3\frac{\tau-1}{(k\tau+1)^2}. \tag{A.11}$$

A.1.1 Proof of Lemma A.1

Let $\mathcal{F}_{k,t}$ denote the history of all sources of randomness by the t-th iteration in period k. The following expectation arguments are conditional on the history $\mathcal{F}_{k,\tau}$ which we remove in our notations for simplicity. Since the random subset of nodes $\mathcal{S}_k$ is uniformly picked from the set of all the nodes $[n]$, we can write

$$\begin{aligned}
\mathbb{E}_{\mathcal{S}_k} \mathbf{w}_{k+1} &= \mathbf{w}_k + \mathbb{E}_{\mathcal{S}_k} \frac{1}{r} \sum_{i \in \mathcal{S}_k} Q\left(\mathbf{w}_{k,\tau}^{(i)} - \mathbf{w}_k\right) \\
&= \mathbf{w}_k + \sum_{\substack{\mathcal{S} \subseteq [n] \\ |\mathcal{S}| = r}} \Pr \mathcal{S}_k = \mathcal{S} \frac{1}{r} \sum_{i \in \mathcal{S}_k} Q\left(\mathbf{w}_{k,\tau}^{(i)} - \mathbf{w}_k\right) \\
&= \mathbf{w}_k + \frac{1}{\binom{n}{r}} \frac{1}{r} \binom{n-1}{r-1} \sum_{i \in [n]} Q\left(\mathbf{w}_{k,\tau}^{(i)} - \mathbf{w}_k\right) \\
&= \mathbf{w}_k + \frac{1}{n} \sum_{i \in [n]} Q\left(\mathbf{w}_{k,\tau}^{(i)} - \mathbf{w}_k\right) \\
&= \widehat{\mathbf{w}}_{k+1}. \qquad \text{(A.12)}
\end{aligned}$$

Moreover, the quantizer $Q(\cdot)$ is unbiased according to Assumption 2.1, which yields

$$\begin{aligned}
\mathbb{E}_Q \, \widehat{\mathbf{w}}_{k+1} &= \mathbf{w}_k + \frac{1}{n} \sum_{i \in [n]} \mathbb{E}_Q \, Q\left(\mathbf{w}_{k,\tau}^{(i)} - \mathbf{w}_k\right) \\
&= \frac{1}{n} \sum_{i \in [n]} \mathbf{w}_{k,\tau}^{(i)} \\
&= \overline{\mathbf{w}}_{k,\tau}. \qquad \text{(A.13)}
\end{aligned}$$

Finally, since the two randomnesses induced by the quantization and random sampling are independent, together with (A.12) and (A.13) we can conclude that:

$$\begin{aligned}
\mathbb{E}\|\mathbf{w}_{k+1} - \mathbf{w}^*\|^2 &= \mathbb{E}\left\|\mathbf{w}_{k+1} - \widehat{\mathbf{w}}_{k+1} + \widehat{\mathbf{w}}_{k+1} - \overline{\mathbf{w}}_{k,\tau} + \overline{\mathbf{w}}_{k,\tau} - \mathbf{w}^*\right\|^2 \\
&= \mathbb{E}\|\mathbf{w}_{k+1} - \widehat{\mathbf{w}}_{k+1}\|^2 + \mathbb{E}\left\|\widehat{\mathbf{w}}_{k+1} - \overline{\mathbf{w}}_{k,\tau}\right\|^2 + \mathbb{E}\left\|\overline{\mathbf{w}}_{k,\tau} - \mathbf{w}^*\right\|^2. \qquad \text{(A.14)}
\end{aligned}$$

A.1.2 Proof of Lemma A.2

According to update rule in Algorithm 2.1, local model at node i for each iteration $t = 0, \cdots, \tau - 1$ of period $k = 0, \cdots, K-1$ can be written as follows:

$$\mathbf{w}_{k,t+1}^{(i)} = \mathbf{w}_{k,t}^{(i)} - \eta_k \widetilde{\nabla} f_i \left(\mathbf{w}_{k,t}^{(i)} \right), \tag{A.15}$$

where all the nodes start the period with the initial model $\mathbf{w}_{k,0}^{(i)} = \mathbf{w}_k$. In parallel, let us define another sequence of updates as follows:

$$\beta_{k,t+1} = \beta_{k,t} - \eta_k \nabla f \left(\beta_{k,t} \right), \tag{A.16}$$

also starting with $\beta_{k,0} = \mathbf{w}_k$. The auxiliary sequence $\{\beta_{k,t}\}$ represents Gradient Descent updates over the global loss function f while $\mathbf{w}_{k,t}^{(i)}$ captures the sequence of SGD updates on each local node. However, both sequences are initialized with $\mathbf{w}_k$ at the beginning of each period k. To evaluate the deviation $\left\| \overline{\mathbf{w}}_{k,\tau} - \mathbf{w}^* \right\|^2$, we link the two sequences. In particular, let us define the following notations for each $k = 0, \cdots, K-1$ and $t = 0, \cdots, \tau - 1$:

$$\mathbf{e}_{k,t} = \frac{1}{n} \sum_{i \in [n]} \widetilde{\nabla} f_i \left(\mathbf{w}_{k,t}^{(i)} \right) - \nabla f \left(\beta_{k,t} \right). \tag{A.17}$$

One can easily observe that $\mathbb{E}\mathbf{e}_{k,0} = 0$ as $\mathbf{w}_{k,0}^{(i)} = \beta_{k,0} = \mathbf{w}_k$ and $\widetilde{\nabla} f_i$ is unbiased for ∇f. However, $\mathbb{E}\mathbf{e}_{k,t} \neq 0$ for $t \geq 1$. In other words, $\frac{1}{n}\sum_{i \in [n]} \widetilde{\nabla} f_i(\mathbf{w}_{k,t}^{(i)})$ is not unbiased for $\nabla f(\beta_{k,t})$. We also define $\mathbf{e}_k = \mathbf{e}_{k,0} + \cdots + \mathbf{e}_{k,\tau-1}$ and $\mathbf{g}_k = \nabla f(\beta_{k,0}) + \cdots + \nabla f(\beta_{k,\tau-1})$. Now, the average model obtained at the end of period k can be written as

$$\overline{\mathbf{w}}_{k,\tau} = \frac{1}{n} \sum_{i \in [n]} \mathbf{w}_{k,\tau}^{(i)}$$

$$
\begin{aligned}
&= \mathbf{w}_k - \eta_k \left(\frac{1}{n} \sum_{i \in [n]} \widetilde{\nabla} f_i \left(\mathbf{w}_{k,0}^{(i)} \right) + \cdots + \frac{1}{n} \sum_{i \in [n]} \widetilde{\nabla} f_i \left(\mathbf{w}_{k,\tau-1}^{(i)} \right) \right) \\
&= \mathbf{w}_k - \eta_k \left(\mathbf{g}_k + \mathbf{e}_k \right). \qquad \text{(A.18)}
\end{aligned}
$$

Therefore, the optimality gap for the averaged model can be written as

$$
\begin{aligned}
\mathbb{E}\left\|\overline{\mathbf{w}}_{k,\tau} - \mathbf{w}^*\right\|^2 &= \mathbb{E}\|\mathbf{w}_k - \eta_k \mathbf{g}_k - \mathbf{w}^*\|^2 - 2\eta_k \mathbb{E}\left\langle \mathbf{w}_k - \eta_k \mathbf{g}_k - \mathbf{w}^*, \mathbf{e}_k \right\rangle + \eta_k^2 \mathbb{E}\|\mathbf{e}_k\|^2 \\
&\le \mathbb{E}\|\mathbf{w}_k - \eta_k \mathbf{g}_k - \mathbf{w}^*\|^2 \\
&\quad + n\eta_k^2 \mathbb{E}\|\mathbf{w}_k - \eta_k \mathbf{g}_k - \mathbf{w}^*\|^2 + \frac{1}{n}\|\mathbb{E}\mathbf{e}_k\|^2 \\
&\quad + \eta_k^2 \mathbb{E}\|\mathbf{e}_k\|^2 \\
&= \left(1 + n\eta_k^2\right) \mathbb{E}\|\mathbf{w}_k - \eta_k \mathbf{g}_k - \mathbf{w}^*\|^2 + \frac{1}{n}\|\mathbb{E}\mathbf{e}_k\|^2 + \eta_k^2 \mathbb{E}\|\mathbf{e}_k\|^2, \qquad \text{(A.19)}
\end{aligned}
$$

where we used the inequality $-2\langle \mathbf{a}, \mathbf{b} \rangle \le \alpha \|\mathbf{a}\|^2 + \alpha^{-1}\|\mathbf{b}\|^2$ for any two vectors $\mathbf{a}, \mathbf{b}$ and scalar $\alpha > 0$. In the following, we bound each of the three terms in the RHS of (A.19). First, consider the term $\|\mathbf{w}_k - \eta_k \mathbf{g}_k - \mathbf{w}^*\|^2$ and recall the auxiliary sequence $\{\beta_{k,t}\}$ defined in (A.20). For every t and k we have

$$
\begin{aligned}
\left\|\beta_{k,t+1} - \mathbf{w}^*\right\|^2 &= \left\|\beta_{k,t} - \eta_k \nabla f(\beta_{k,t}) - \mathbf{w}^*\right\|^2 \\
&= \left\|\beta_{k,t} - \mathbf{w}^*\right\|^2 - 2\eta_k \left\langle \beta_{k,t} - \mathbf{w}^*, \nabla f(\beta_{k,t}) \right\rangle + \eta_k^2 \left\|\nabla f(\beta_{k,t})\right\|^2 \\
&\le \left(1 - 2\mu\eta_k + L^2\eta_k^2\right) \left\|\beta_{k,t} - \mathbf{w}^*\right\|^2 \\
&\le (1 - \mu\eta_k) \left\|\beta_{k,t} - \mathbf{w}^*\right\|^2. \qquad \text{(A.20)}
\end{aligned}
$$

In the above derivations, we used the facts that f is μ-strongly convex and its gradient is L-Lipschitz (Assumptions 2.2 and 2.4). The stepsize is also picked such that $\eta_k \le \frac{\mu}{L^2}$.

Now, conditioned on the history $\mathcal{F}_{k,0}$ and using (A.20) we have

$$\begin{aligned}
\left\|\mathbf{w}_k - \eta_k \mathbf{g}_k - \mathbf{w}^*\right\|^2 &= \left\|\beta_{k,\tau} - \mathbf{w}^*\right\|^2 \\
&\leq (1-\mu\eta_k)^\tau \left\|\beta_{k,0} - \mathbf{w}^*\right\|^2 \\
&= (1-\mu\eta_k)^\tau \left\|\mathbf{w}_k - \mathbf{w}^*\right\|^2 . \qquad \text{(A.21)}
\end{aligned}$$

Secondly, consider the term $\left\|\mathbb{E}\mathbf{e}_k\right\|^2$ in (A.19). By definition, we have $\mathbb{E}\mathbf{e}_k = \mathbb{E}\mathbf{e}_{k,1} + \cdots + \mathbb{E}\mathbf{e}_{k,\tau-1}$ and hence $\left\|\mathbb{E}\mathbf{e}_k\right\|^2 \leq (\tau-1)\left\|\mathbb{E}\mathbf{e}_{k,1}\right\|^2 + \cdots + (\tau-1)\left\|\mathbb{E}\mathbf{e}_{k,\tau-1}\right\|^2$. The first term $\left\|\mathbb{E}\mathbf{e}_{k,1}\right\|^2$ can be bounded using Assumptions 2.2 and 2.3 as follows:

$$\begin{aligned}
\left\|\mathbb{E}\mathbf{e}_{k,1}\right\|^2 &= \left\|\frac{1}{n}\sum_{i\in[n]} \mathbb{E}\widetilde{\nabla} f_i\left(\mathbf{w}_{k,1}^{(i)}\right) - \nabla f\left(\beta_{k,1}\right)\right\|^2 \\
&= \left\|\frac{1}{n}\sum_{i\in[n]} \mathbb{E}\nabla f\left(\mathbf{w}_{k,1}^{(i)}\right) - \nabla f\left(\beta_{k,1}\right)\right\|^2 \\
&\leq \frac{1}{n}\sum_{i\in[n]} \mathbb{E}\left\|\nabla f\left(\mathbf{w}_{k,1}^{(i)}\right) - \nabla f\left(\beta_{k,1}\right)\right\|^2 \\
&\leq \frac{1}{n}L^2\sum_{i\in[n]} \mathbb{E}\left\|\mathbf{w}_{k,1}^{(i)} - \beta_{k,1}\right\|^2 \\
&= \frac{1}{n}L^2\sum_{i\in[n]} \mathbb{E}\left\|\left(\mathbf{w}_{k,0}^{(i)} - \eta_k\widetilde{\nabla} f_i\left(\mathbf{w}_{k,0}^{(i)}\right)\right) - \left(\beta_{k,0} - \eta_k \nabla f\left(\beta_{k,0}\right)\right)\right\|^2 \\
&= \frac{1}{n}L^2\eta_k^2\sum_{i\in[n]} \mathbb{E}\left\|\widetilde{\nabla} f_i\left(\mathbf{w}_k\right) - \nabla f\left(\mathbf{w}_k\right)\right\|^2 \\
&\leq L^2\sigma^2\eta_k^2. \qquad \text{(A.22)}
\end{aligned}$$

In general, for each $t = 1\cdots,\tau-1$ we can write

$$\left\|\mathbb{E}\mathbf{e}_{k,t}\right\|^2 = \left\|\frac{1}{n}\sum_{i\in[n]} \mathbb{E}\widetilde{\nabla} f_i\left(\mathbf{w}_{k,t}^{(i)}\right) - \nabla f\left(\beta_{k,t}\right)\right\|^2$$

$$
\begin{aligned}
&= \left\| \frac{1}{n} \sum_{i \in [n]} \mathbb{E} \nabla f\left(\mathbf{w}_{k,t}^{(i)}\right) - \nabla f\left(\beta_{k,t}\right) \right\|^2 \\
&\leq \frac{1}{n} \sum_{i \in [n]} \mathbb{E} \left\| \nabla f\left(\mathbf{w}_{k,t}^{(i)}\right) - \nabla f\left(\beta_{k,t}\right) \right\|^2 \\
&\leq \frac{1}{n} L^2 \sum_{i \in [n]} \mathbb{E} \left\| \mathbf{w}_{k,t}^{(i)} - \beta_{k,t} \right\|^2 . \qquad \text{(A.23)}
\end{aligned}
$$

Let us denote $a_{k,t} \coloneqq \frac{1}{n} \sum_{i \in [n]} \mathbb{E} \left\| \mathbf{w}_{k,t}^{(i)} - \beta_{k,t} \right\|^2$. In the following, we will derive a recursive bound on a_t. That is,

$$
\begin{aligned}
a_{k,t} &= \frac{1}{n} \sum_{i \in [n]} \mathbb{E} \left\| \mathbf{w}_{k,t}^{(i)} - \beta_{k,t} \right\|^2 \\
&= \frac{1}{n} \sum_{i \in [n]} \mathbb{E} \Bigg\| \left(\mathbf{w}_{k,0}^{(i)} - \eta_k \widetilde{\nabla} f_i\left(\mathbf{w}_{k,0}^{(i)}\right) - \cdots - \eta_k \widetilde{\nabla} f_i\left(\mathbf{w}_{k,t-1}^{(i)}\right) \right) \\
&\qquad\qquad - \left(\beta_{k,0} - \eta_k \nabla f\left(\beta_{k,0}\right) - \cdots - \eta_k \nabla f\left(\beta_{k,t-1}\right) \right) \Bigg\|^2 \\
&= \frac{1}{n} \eta_k^2 \sum_{i \in [n]} \mathbb{E} \left\| \widetilde{\nabla} f_i\left(\mathbf{w}_{k,0}^{(i)}\right) - \nabla f\left(\beta_{k,0}\right) + \cdots + \widetilde{\nabla} f_i\left(\mathbf{w}_{k,t-1}^{(i)}\right) - \nabla f\left(\beta_{k,t-1}\right) \right\|^2 \\
&\leq \eta_k^2 \sigma^2 + \frac{1}{n} \eta_k^2 \sum_{i \in [n]} \mathbb{E} \left\| \widetilde{\nabla} f_i\left(\mathbf{w}_{k,1}^{(i)}\right) - \nabla f\left(\beta_{k,1}\right) + \cdots + \widetilde{\nabla} f_i\left(\mathbf{w}_{k,t-1}^{(i)}\right) - \nabla f\left(\beta_{k,t-1}\right) \right\|^2 \\
&\leq \eta_k^2 \sigma^2 + \frac{1}{n} \eta_k^2 \sum_{i \in [n]} \mathbb{E} \Bigg\| \widetilde{\nabla} f_i\left(\mathbf{w}_{k,1}^{(i)}\right) - \nabla f\left(\mathbf{w}_{k,1}^{(i)}\right) + \nabla f\left(\mathbf{w}_{k,1}^{(i)}\right) - \nabla f\left(\beta_{k,1}\right) \\
&\qquad\qquad + \cdots + \widetilde{\nabla} f_i\left(\mathbf{w}_{k,t-1}^{(i)}\right) - \nabla f\left(\mathbf{w}_{k,t-1}^{(i)}\right) + \nabla f\left(\mathbf{w}_{k,t-1}^{(i)}\right) - \nabla f\left(\beta_{k,t-1}\right) \Bigg\|^2 \\
&\leq t \eta_k^2 \sigma^2 + \frac{1}{n} \eta_k^2 \sum_{i \in [n]} \mathbb{E} \left\| \nabla f\left(\mathbf{w}_{k,1}^{(i)}\right) - \nabla f\left(\beta_{k,1}\right) + \cdots \nabla f\left(\mathbf{w}_{k,t-1}^{(i)}\right) - \nabla f\left(\beta_{k,t-1}\right) \right\|^2 \\
&\leq t \eta_k^2 \sigma^2 + (t-1) L^2 \eta_k^2 \frac{1}{n} \sum_{i \in [n]} \mathbb{E} \left\| \mathbf{w}_{k,1}^{(i)} - \beta_{k,1} \right\|^2 + \cdots (t-1) L^2 \eta_k^2 \frac{1}{n} \sum_{i \in [n]} \mathbb{E} \left\| \mathbf{w}_{k,t-1}^{(i)} - \beta_{k,t-1} \right\|^2
\end{aligned}
$$

$$
\begin{aligned}
&= t\eta_k^2\sigma^2 + (t-1)L^2\eta_k^2\left(a_{k,1} + \cdots + a_{k,t-1}\right) \\
&\leq \tau\eta_k^2\sigma^2 + \tau L^2\eta_k^2\left(a_{k,1} + \cdots + a_{k,t-1}\right). \qquad \text{(A.24)}
\end{aligned}
$$

Therefore, for the sequence $\{a_{k,1}, \cdots, a_{k,\tau-1}\}$ we have shown that

$$
a_{k,t} \leq \tau\eta_k^2\sigma^2 + \tau L^2\eta_k^2\left(a_{k,1} + \cdots + a_{k,t-1}\right), \qquad \text{(A.25)}
$$

where $a_{k,1} \leq \sigma^2\eta_k^2$. We can show by induction, that such sequence satisfies the following inequality:

$$
a_{k,t} \leq \tau\eta_k^2\sigma^2\left(1 + \tau L^2\eta_k^2\right)^{t-1}. \qquad \text{(A.26)}
$$

See Section A.1.5 for the detailed proof. Therefore, we have

$$
\begin{aligned}
\left\|\mathbb{E}\mathbf{e}_k\right\|^2 &\leq (\tau-1)\left\|\mathbb{E}\mathbf{e}_{k,1}\right\|^2 + \cdots + (\tau-1)\left\|\mathbb{E}\mathbf{e}_{k,\tau-1}\right\|^2 \\
&\leq (\tau-1)L^2\left(a_1 + \cdots + a_{\tau-1}\right) \\
&\leq \tau(\tau-1)^2L^2\sigma^2\eta_k^2\left(1 + \tau L^2\eta_k^2\right)^{\tau}. \qquad \text{(A.27)}
\end{aligned}
$$

Now, we use the inequality $1 + x \leq e^x$ and conclude that

$$
\left\|\mathbb{E}\mathbf{e}_k\right\|^2 \leq \tau(\tau-1)^2L^2\sigma^2\eta_k^2 e^{\tau^2L^2\eta_k^2}. \qquad \text{(A.28)}
$$

Therefore, if $\tau^2L^2\eta_k^2 \leq 1$, we have

$$
\left\|\mathbb{E}\mathbf{e}_k\right\|^2 \leq \tau(\tau-1)^2L^2\sigma^2 e\eta_k^2. \qquad \text{(A.29)}
$$

Finally, we bound the third term in (A.19), that is $\mathbb{E}\|\mathbf{e}_k\|^2$. Using the definition, we know that $\mathbb{E}\|\mathbf{e}_k\|^2 \leq \tau\mathbb{E}\left\|\mathbf{e}_{k,0}\right\|^2 + \cdots + \tau\mathbb{E}\left\|\mathbf{e}_{k,\tau-1}\right\|^2$. Firstly, note that

$$\begin{aligned}
\mathbb{E}\left\|\mathbf{e}_{k,0}\right\|^2 &= \mathbb{E}\left\|\frac{1}{n}\sum_{i\in[n]}\widetilde{\nabla} f_i\left(\mathbf{w}_{k,0}^{(i)}\right) - \nabla f\left(\beta_{k,0}\right)\right\|^2 \\
&= \mathbb{E}\left\|\frac{1}{n}\sum_{i\in[n]}\widetilde{\nabla} f_i\left(\mathbf{w}_{k}\right) - \nabla f\left(\mathbf{w}_{k}\right)\right\|^2 \\
&\leq \frac{\sigma^2}{n}. \qquad \text{(A.30)}
\end{aligned}$$

For each $t = 1, \cdots, \tau - 1$ we have

$$\begin{aligned}
\mathbb{E}\left\|\mathbf{e}_{k,t}\right\|^2 &= \mathbb{E}\left\|\frac{1}{n}\sum_{i\in[n]}\widetilde{\nabla} f_i\left(\mathbf{w}_{k,t}^{(i)}\right) - \nabla f\left(\beta_{k,t}\right)\right\|^2 \\
&= \mathbb{E}\left\|\frac{1}{n}\sum_{i\in[n]}\widetilde{\nabla} f_i\left(\mathbf{w}_{k,t}^{(i)}\right) - \nabla f\left(\mathbf{w}_{k,t}^{(i)}\right) + \frac{1}{n}\sum_{i\in[n]}\nabla f\left(\mathbf{w}_{k,t}^{(i)}\right) - \nabla f\left(\beta_{k,t}\right)\right\|^2 \\
&\leq \frac{\sigma^2}{n} + L^2\frac{1}{n}\sum_{i\in[n]}\mathbb{E}\left\|\mathbf{w}_{k,t}^{(i)} - \beta_{k,t}\right\|^2 \\
&= \frac{\sigma^2}{n} + L^2 a_{k,t}. \qquad \text{(A.31)}
\end{aligned}$$

Summing over $t = 0, 1, \cdots, \tau - 1$ results in the following

$$\begin{aligned}
\mathbb{E}\|\mathbf{e}_k\|^2 &\leq \tau\mathbb{E}\left\|\mathbf{e}_{k,0}\right\|^2 + \cdots + \tau\mathbb{E}\left\|\mathbf{e}_{k,\tau-1}\right\|^2 \\
&\leq \tau^2\frac{\sigma^2}{n} + \tau L^2\left(a_1 + \cdots + a_{\tau-1}\right) \\
&\leq \tau^2\frac{\sigma^2}{n} + \tau^2(\tau-1)L^2\sigma^2\eta_k^2\left(1 + \tau L^2\eta_k^2\right)^\tau \\
&\leq \tau^2\frac{\sigma^2}{n} + \tau^2(\tau-1)L^2\sigma^2 e\eta_k^2. \qquad \text{(A.32)}
\end{aligned}$$

Now, we can put everything together and conclude Lemma A.2, as follows

$$\begin{aligned}\mathbb{E}\left\|\overline{\mathbf{w}}_{k,\tau}-\mathbf{w}^*\right\|^2 &= \left(1+n\eta_k^2\right)\mathbb{E}\|\mathbf{w}_k-\eta_k\mathbf{g}_k-\mathbf{w}^*\|^2+\frac{1}{m}\|\mathbb{E}\mathbf{e}_k\|^2+\eta_k^2\mathbb{E}\|\mathbf{e}_k\|^2\\ &\le \left(1+n\eta_k^2\right)\left(1-\mu\eta_k\right)^\tau\mathbb{E}\|\mathbf{w}_k-\mathbf{w}^*\|^2\\ &\quad+\tau(\tau-1)^2L^2\frac{\sigma^2}{n}e\eta_k^2+\tau^2\frac{\sigma^2}{n}\eta_k^2\\ &\quad+\tau^2(\tau-1)L^2\sigma^2e\eta_k^4. \end{aligned} \tag{A.33}$$

A.1.3 Proof of Lemma A.3

According to the notations defined on (A.1), we can write

$$\begin{aligned}\mathbb{E}\left\|\widehat{\mathbf{w}}_{k+1}-\overline{\mathbf{w}}_{k,\tau}\right\|^2 &= \mathbb{E}\left\|\mathbf{w}_k+\frac{1}{n}\sum_{i\in[n]}Q\left(\mathbf{w}_{k,\tau}^{(i)}-\mathbf{w}_k\right)-\frac{1}{n}\sum_{i\in[n]}\mathbf{w}_{k,\tau}^{(i)}\right\|^2\\ &= \mathbb{E}\left\|\frac{1}{n}\sum_{i\in[n]}Q\left(\mathbf{w}_{k,\tau}^{(i)}-\mathbf{w}_k\right)-\left(\mathbf{w}_{k,\tau}^{(i)}-\mathbf{w}_k\right)\right\|^2\\ &= \frac{1}{n^2}\sum_{i\in[n]}\mathbb{E}\left\|Q\left(\mathbf{w}_{k,\tau}^{(i)}-\mathbf{w}_k\right)-\left(\mathbf{w}_{k,\tau}^{(i)}-\mathbf{w}_k\right)\right\|^2\\ &\le q\frac{1}{n^2}\sum_{i\in[n]}\mathbb{E}\left\|\mathbf{w}_{k,\tau}^{(i)}-\mathbf{w}_k\right\|^2, \end{aligned} \tag{A.34}$$

where, we used Assumption 2.1. In particular, the last equality above follows from the fact that the random quatizer is unbiased and the quantizations are carried out independently in each iteration and each worker. Moreover, the last inequality in (A.34) simply relates the variance of the quantization to its argument. Next, we bound $\mathbb{E}\left\|\mathbf{w}_{k,\tau}^{(i)}-\mathbf{w}_k\right\|^2$ for each worker $i \in [n]$. From the update rule in Algorithm 2.1 we have

$$\begin{aligned}
\mathbf{w}_{k,\tau}^{(i)} &= \mathbf{w}_k - \eta_k \left(\widetilde{\nabla} f_i \left(\mathbf{w}_{k,0}^{(i)} \right) + \cdots + \widetilde{\nabla} f_i \left(\mathbf{w}_{k,\tau-1}^{(i)} \right) \right) \\
&= \mathbf{w}_k - \eta_k \left(\mathbf{g}_k + \mathbf{e}_k^{(i)} \right), \qquad \text{(A.35)}
\end{aligned}$$

where we denote

$$\mathbf{e}_k^{(i)} \coloneqq \widetilde{\nabla} f_i \left(\mathbf{w}_{k,0}^{(i)} \right) - \nabla f \left(\beta_{k,0} \right) + \cdots + \widetilde{\nabla} f_i \left(\mathbf{w}_{k,\tau-1}^{(i)} \right) - \nabla f \left(\beta_{k,\tau-1} \right), \qquad \text{(A.36)}$$

and $\mathbf{g}_k = \nabla f(\beta_{k,0}) + \cdots + \nabla f(\beta_{k,\tau-1})$ as defined before. Using these notations we have

$$\begin{aligned}
\mathbb{E}\left\| \mathbf{w}_{k,\tau}^{(i)} - \mathbf{w}_k \right\|^2 &= \eta_k^2 \mathbb{E}\left\| \mathbf{g}_k + \mathbf{e}_k^{(i)} \right\|^2 \\
&\le 2\eta_k^2 \|\mathbf{g}_k\|^2 + 2\eta_k^2 \mathbb{E}\left\| \mathbf{e}_k^{(i)} \right\|^2. \qquad \text{(A.37)}
\end{aligned}$$

Let us first bound the first term in (A.37), i.e. $\|\mathbf{g}_k\|^2$. That is,

$$\begin{aligned}
\|\mathbf{g}_k\|^2 &\le \tau \left\| \nabla f(\beta_{k,0}) \right\|^2 + \cdots + \tau \left\| \nabla f(\beta_{k,\tau-1}) \right\|^2 \\
&\overset{(a)}{\le} \tau L^2 \left(\|\mathbf{w}_k - \mathbf{w}^*\|^2 + \cdots + (1 - \mu\eta_k)^{\tau-1} \|\mathbf{w}_k - \mathbf{w}^*\|^2 \right) \\
&\le \tau^2 L^2 \|\mathbf{w}_k - \mathbf{w}^*\|^2, \qquad \text{(A.38)}
\end{aligned}$$

where we used the smoothness of the loss function f (Assumption 2.2) and the result in (A.20) to derive inequality (a). To bound the second term in (A.37), i.e. $\mathbb{E}\left\| \mathbf{e}_k^{(i)} \right\|^2$, we can employ our result in (A.32) for the special case $n = 1$. It yields that for $\eta_k \le \frac{1}{L\tau}$,

$$\mathbb{E}\left\| \mathbf{e}_k^{(i)} \right\|^2 \le \tau^2 \sigma^2 + \tau^2 (\tau - 1) L^2 \sigma^2 e \eta_k^2. \qquad \text{(A.39)}$$

Plugging (A.38) and (A.39) in (A.37) implies that

$$\mathbb{E}\left\|\mathbf{w}_{k,\tau}^{(i)}-\mathbf{w}_k\right\|^2 \leq 2\tau^2L^2\eta_k^2\mathbb{E}\|\mathbf{w}_k-\mathbf{w}^*\|^2+2\tau^2\sigma^2\eta_k^2+2(\tau-1)\tau^2L^2\sigma^2e\eta_k^4, \tag{A.40}$$

which together with (A.34) concludes Lemma A.3:

$$\mathbb{E}\left\|\widehat{\mathbf{w}}_{k+1}-\overline{\mathbf{w}}_{k,\tau}\right\|^2 \leq 2\frac{q}{n}\tau^2L^2\eta_k^2\mathbb{E}\|\mathbf{w}_k-\mathbf{w}^*\|^2+2q\tau^2\frac{\sigma^2}{n}\eta_k^2+2q(\tau-1)\tau^2L^2\frac{\sigma^2}{n}e\eta_k^4. \tag{A.41}$$

A.1.4 Proof of Lemma A.4

For each node $i \in [n]$ denote $\mathbf{z}_{k,\tau}^{(i)} = Q(\mathbf{w}_{k,\tau}^{(i)}-\mathbf{w}_k)$ and $\overline{\mathbf{z}}_{k,\tau} = \frac{1}{n}\sum_{i\in[n]}\mathbf{z}_{k,\tau}^{(i)}$. Then,

$$\begin{aligned}
\mathbb{E}_{\mathcal{S}_k}\|\mathbf{w}_{k+1}-\widehat{\mathbf{w}}_{k+1}\|^2 &= \mathbb{E}_{\mathcal{S}_k}\left\|\frac{1}{r}\sum_{i\in\mathcal{S}_k}\mathbf{z}_{k,\tau}^{(i)}-\overline{\mathbf{z}}_{k,\tau}\right\|^2\\
&= \frac{1}{r^2}\mathbb{E}_{\mathcal{S}_k}\left\|\sum_{i\in[n]}\mathbb{1}\{i\in\mathcal{S}_k\}\left(\mathbf{z}_{k,\tau}^{(i)}-\overline{\mathbf{z}}_{k,\tau}\right)\right\|^2\\
&= \frac{1}{r^2}\Bigg\{\sum_{i\in[n]}\Pr i\in\mathcal{S}_k\left\|\mathbf{z}_{k,\tau}^{(i)}-\overline{\mathbf{z}}_{k,\tau}\right\|^2\\
&\quad+\sum_{i\neq j}\Pr i,j\in\mathcal{S}_k\left\langle\mathbf{z}_{k,\tau}^{(i)}-\overline{\mathbf{z}}_{k,\tau},\mathbf{z}_{k,\tau}^{(j)}-\overline{\mathbf{z}}_{k,\tau}\right\rangle\Bigg\}\\
&= \frac{1}{nr}\sum_{i\in[n]}\left\|\mathbf{z}_{k,\tau}^{(i)}-\overline{\mathbf{z}}_{k,\tau}\right\|^2+\frac{r-1}{rn(n-1)}\sum_{i\neq j}\left\langle\mathbf{z}_{k,\tau}^{(i)}-\overline{\mathbf{z}}_{k,\tau},\mathbf{z}_{k,\tau}^{(j)}-\overline{\mathbf{z}}_{k,\tau}\right\rangle\\
&= \frac{1}{r(n-1)}\left(1-\frac{r}{n}\right)\sum_{i\in[n]}\left\|\mathbf{z}_{k,\tau}^{(i)}-\overline{\mathbf{z}}_{k,\tau}\right\|^2,
\end{aligned} \tag{A.42}$$

where we used the fact that $\left\|\mathbf{z}_{k,\tau}^{(i)}-\overline{\mathbf{z}}_{k,\tau}\right\|^2+\sum_{i\neq j}\left\langle \mathbf{z}_{k,\tau}^{(i)}-\overline{\mathbf{z}}_{k,\tau}, \mathbf{z}_{k,\tau}^{(j)}-\overline{\mathbf{z}}_{k,\tau}\right\rangle=0$. Further taking expectation with respect to the quantizer yields

$$\begin{aligned}\sum_{i\in[n]}\mathbb{E}_Q\left\|\mathbf{z}_{k,\tau}^{(i)}-\overline{\mathbf{z}}_{k,\tau}\right\|^2 &\leq 2\sum_{i\in[n]}\mathbb{E}_Q\left\|\mathbf{z}_{k,\tau}^{(i)}\right\|^2+2n\mathbb{E}_Q\left\|\overline{\mathbf{z}}_{k,\tau}\right\|^2\\ &\leq 4\sum_{i\in[n]}\mathbb{E}_Q\left\|\mathbf{z}_{k,\tau}^{(i)}\right\|^2\\ &= 4\sum_{i\in[n]}\mathbb{E}_Q\left\|Q\left(\mathbf{w}_{k,\tau}^{(i)}-\mathbf{w}_k\right)\right\|^2\\ &\leq 4(1+q)\sum_{i\in[n]}\left\|\mathbf{w}_{k,\tau}^{(i)}-\mathbf{w}_k\right\|^2. \qquad (A.43)\end{aligned}$$

In the above derivations, we used the fact that under Assumption 2.1 and for any $\mathbf{w}$ we have $\mathbb{E}\left\|Q(\mathbf{w})\right\|^2\leq(1+q)\|\mathbf{w}\|^2$. Therefore, (A.43) together with the equality derived in (A.42) yields that

$$\mathbb{E}\|\mathbf{w}_{k+1}-\widehat{\mathbf{w}}_{k+1}\|^2\leq\frac{1}{r(n-1)}\left(1-\frac{r}{n}\right)4(1+q)\sum_{i\in[n]}\mathbb{E}\left\|\mathbf{w}_{k,\tau}^{(i)}-\mathbf{w}_k\right\|^2. \qquad (A.44)$$

Finally, we substitute the bound in (A.40) into (A.44) and conclude Lemma A.4 as follows:

$$\mathbb{E}\|\mathbf{w}_{k+1}-\widehat{\mathbf{w}}_{k+1}\|^2\leq\frac{n-r}{r(n-1)}8(1+q)\left\{\tau^2L^2\eta^2\mathbb{E}\|\mathbf{w}_k-\mathbf{w}^*\|^2+\tau^2\sigma^2\eta^2+(\tau-1)\tau^2L^2\sigma^2e\eta^4\right\}. \qquad (A.45)$$

A.1.5 Proof of Equation (A.26)

Let us fix the period k and for simplicity of the notations in this proof, let us take $a_t=a_{k,t}$ and $\eta=\eta_k$. We showed that $a_t\leq\tau\eta^2\sigma^2+\tau L^2\eta^2\left(a_1+\cdots+a_{t-1}\right)$ for every $t=2,\cdots,\tau-1$ and also $a_1\leq\eta^2\sigma^2$. For $t=1$, (A.26) holds. Assume that (A.26) holds also for $\{a_1, \cdots, a_{t-1}\}$. Now, for a_t we have

$$\begin{aligned}
a_t &\le \tau\eta^2\sigma^2 + \tau L^2\eta^2 \left(a_1 + \cdots + a_{t-1}\right) \\
&\le \tau\eta^2\sigma^2 + \tau L^2\eta^2 \sum_{i=0}^{t-2} \tau\eta^2\sigma^2 \left(1 + \tau L^2\eta^2\right)^i \\
&= \tau\eta^2\sigma^2 + \tau\eta^2\sigma^2 \cdot \tau L^2\eta^2 \cdot \frac{\left(1 + \tau L^2\eta^2\right)^{t-1} - 1}{\tau L^2\eta^2} \\
&= \tau\eta^2\sigma^2 \left(1 + \tau L^2\eta^2\right)^{t-1}, \qquad \text{(A.46)}
\end{aligned}$$

as desired. Therefore, (A.26) holds for every $t = 1, \cdots, \tau - 1$.

A.1.6 Discussion on stepsize η_k

Here we show that for any $k \ge k_0$ we have $C_0 \le 1 - \frac{1}{2}\mu\tau\eta_k$, where k_0 satisfies the condition in Theorem 2.1, that is

$$k_0 \ge 4 \max\left\{\frac{L}{\mu}, 4\left(\frac{B_1}{\mu^2} + 1\right), \frac{1}{\tau}, \frac{4n}{\mu^2\tau}\right\}. \qquad \text{(A.47)}$$

First note that this condition on k_0 implies the following conditions on the stepsize $\eta_k = \frac{4\mu^{-1}}{k\tau+1}$ for $k \ge k_0$:

$$\eta_k\tau \le \min\left\{\frac{1}{L}, \frac{\mu}{4\left(\mu^2 + B_1\right)}\right\}, \quad \text{and} \quad \eta_k \le \min\left\{\frac{\mu}{L^2}, \frac{\mu}{4n}\right\}. \qquad \text{(A.48)}$$

Now consider the term $(1 - \mu\eta_k)^\tau$ in C_0. We have

$$\begin{aligned}
(1 - \mu\eta_k)^\tau &= \left(1 - \frac{\mu\tau\eta_k}{\tau}\right)^\tau \\
&\le e^{-\mu\tau\eta_k}
\end{aligned}$$

$$\leq 1 - \mu\tau\eta_k + \mu^2\tau^2\eta_k^2, \tag{A.49}$$

where the first inequality follows from the assumption $\eta_k \leq \frac{1}{\mu}$ and the second inequality uses the fact that $e^x \leq 1 + x + x^2$ for $x \leq 0$. Therefore,

$$\begin{aligned} C_0 &\leq \left(1 + n\eta_k^2\right)\left(1 - \mu\tau\eta_k + \mu^2\tau^2\eta_k^2\right) + B_1\tau^2\eta_k^2 \\ &= 1 - \mu\tau\eta_k + \tau^2\eta_k^2(B_1 + \mu^2) + n\eta_k^2\left(1 - \mu\tau\eta_k + \mu^2\tau^2\eta_k^2\right). \end{aligned} \tag{A.50}$$

Note that from the assumption $\eta_k \leq \frac{1}{L\tau}$ we have $0 \leq \mu\tau\eta_k \leq \frac{\mu}{L} \leq 1$. This implies that $1 - \mu\tau\eta_k + \mu^2\tau^2\eta_k^2 \leq 1$. Hence,

$$C_0 \leq 1 - \mu\tau\eta_k + \tau^2\eta_k^2(B_1 + \mu^2) + n\eta_k^2. \tag{A.51}$$

Now from the condition $\eta_k\tau \leq \frac{\mu}{4(B_1+\mu^2)}$ we have

$$\tau^2\eta_k^2(B_1 + \mu^2) \leq \frac{1}{4}\mu\tau\eta_k, \tag{A.52}$$

and from $\eta_k \leq \frac{\mu}{4n}$ we have

$$n\eta_k^2 \leq \frac{1}{4}\mu\tau\eta_k, \tag{A.53}$$

sine $\tau \geq 1$. Plugging (A.52) and (A.53) in (A.51) yields that for any $k \geq k_0$ we have $C_0 \leq 1 - \frac{1}{2}\mu\tau\eta_k$.

A.1.7 Skipped lemmas and proofs

Lemma A.5 *Let a non-negative sequence δ_k satisfy the following*

$$\delta_{k+1} \leq \left(1 - \frac{2}{k+k_1}\right)\delta_k + \frac{a}{(k+k_1)^2} + \frac{b}{(k+k_1)^4}, \tag{A.54}$$

for every $k \geq k_0$, where a, b, c, k_1 are positive reals and k_0 is a positive integer. Then for every $k \geq k_0$ we have

$$\delta_k \leq \frac{(k_0+k_1)^2}{(k+k_1)^2}\delta_{k_0} + \frac{a}{k+k_1} + \frac{b}{(k+k_1)^2}. \tag{A.55}$$

Proof: We prove by induction on $k \geq k_0$. The claim in (A.55) is trivial for $k = k_0$. Let (A.55) hold for $s \geq k_0$, that is

$$\delta_s \leq \frac{(k_0+k_1)^2}{(s+k_1)^2}\delta_{k_0} + \frac{a}{s+k_1} + \frac{b}{(s+k_1)^2}. \tag{A.56}$$

We can then write

$$\begin{aligned}
\delta_{s+1} &\leq \left(1 - \frac{2}{s+k_1}\right)\delta_s + \frac{a}{s+k_1} + \frac{b}{(s+k_1)^2} \\
&\leq \left(1 - \frac{2}{s+k_1}\right)\left(\frac{(k_0+k_1)^2}{(s+k_1)^2}\delta_{k_0} + \frac{a}{s+k_1} + \frac{b}{(s+k_1)^2}\right) + \frac{a}{(s+k_1)^2} + \frac{b}{(s+k_1)^4} \\
&= \frac{s+k_1-2}{(s+k_1)^3}(k_0+k_1)^2\delta_{k_0} + \frac{s+k_1-1}{(s+k_1)^2}a + \frac{(s+k_1-1)^2}{(s+k_1)^4}b.
\end{aligned} \tag{A.57}$$

Now, take $s' = s + k_1$. We have for $s' \geq 1$ that

$$\frac{s'-2}{s'^3} \leq \frac{1}{(s'+1)^2}, \qquad \frac{s'-1}{s'^2} \leq \frac{1}{s'+1}, \qquad \frac{(s'-1)^2}{s'^4} \leq \frac{1}{(s'+1)^2}. \tag{A.58}$$

Plugging (A.58) in (A.57) yields that the claim in (A.55) holds for $s+1$ and hence for any $k \geq k_0$. ■

A.2 Proof of Theorem 2.2

We begin the proof of Theorem 2.2 by noting the following property for any smooth loss function.

Lemma A.6 *Consider the sequences of updates $\{\mathbf{w}_{k+1}, \widehat{\mathbf{w}}_{k+1}, \overline{\mathbf{w}}_{k,\tau}\}$ generated by* `FedPAQ` *method in Algorithm 2.1. If Assumptions 2.1 and 2.2 hold, then*

$$\mathbb{E}f(\mathbf{w}_{k+1}) \leq \mathbb{E}f(\overline{\mathbf{w}}_{k,\tau}) + \frac{L}{2}\mathbb{E}\left\|\widehat{\mathbf{w}}_{k+1} - \overline{\mathbf{w}}_{k,\tau}\right\|^2 + \frac{L}{2}\mathbb{E}\|\widehat{\mathbf{w}}_{k+1} - \mathbf{w}_{k+1}\|^2, \tag{A.59}$$

for any period $k = 0, \cdots, K-1$.

Proof: See Section A.2.2. ■

In the following three lemmas, we bound each of the three terms in the RHS of (A.59).

Lemma A.7 *Let Assumptions 2.2 and 2.3 hold and consider the sequence of updates in* `FedPAQ` *method with stepsize η. Then, for every period $k = 0, \cdots, K-1$ we have*

$$\begin{aligned}\mathbb{E}f(\overline{\mathbf{w}}_{k,\tau}) \leq \mathbb{E}f(\mathbf{w}_k) &- \frac{1}{2}\eta\sum_{t=0}^{\tau-1}\mathbb{E}\left\|\nabla f(\overline{\mathbf{w}}_{k,t})\right\|^2 \\ &- \eta\left(\frac{1}{2n} - \frac{1}{2n}L\eta - \frac{1}{n}L^2\tau(\tau-1)\eta^2\right)\sum_{t=0}^{\tau-1}\sum_{i\in[n]}\mathbb{E}\left\|\nabla f\left(\mathbf{w}_{k,t}^{(i)}\right)\right\|^2 \\ &+ \eta^2\frac{L}{2}\frac{\sigma^2}{n}\tau + \eta^3\frac{\sigma^2}{n}(n+1)\frac{\tau(\tau-1)}{2}L^2.\end{aligned} \tag{A.60}$$

Proof: See Section A.2.3. ■

Lemma A.8 *If Assumptions 2.1 and 2.3 hold, then for sequences* $\{\widehat{\mathbf{w}}_{k+1}, \overline{\mathbf{w}}_{k,\tau}\}$ *defined in* (A.1) *we have*

$$\mathbb{E}\left\|\widehat{\mathbf{w}}_{k+1} - \overline{\mathbf{w}}_{k,\tau}\right\|^2 \leq q\frac{\sigma^2}{n}\tau\eta^2 + q\frac{1}{n^2}\tau\eta^2 \sum_{i\in[n]}\sum_{t=0}^{\tau-1}\left\|\nabla f\left(\mathbf{w}_{k,t}^{(i)}\right)\right\|^2. \tag{A.61}$$

Proof: See Section A.2.4. ■

Lemma A.9 *Under Assumptions 2.1 and 2.3, for the sequence of averages* $\{\widehat{\mathbf{w}}_{k+1}\}$ *defined in* (A.1) *we have*

$$\mathbb{E}\|\widehat{\mathbf{w}}_{k+1} - \mathbf{w}_{k+1}\|^2 \leq \frac{1}{r(n-1)}\left(1-\frac{r}{n}\right)4(1+q)\left\{n\sigma^2\tau\eta^2 + \tau\eta^2\sum_{i\in[n]}\sum_{t=0}^{\tau-1}\left\|\nabla f\left(\mathbf{w}_{k,t}^{(i)}\right)\right\|^2\right\}. \tag{A.62}$$

Proof: See Section A.2.5. ■

After establishing the main building modules in the above lemmas, we now proceed to prove the convergence rate in Theorem 2.2. In particular, we combine the results in Lemmas A.6–A.9 to derive the following recursive inequality on the expected function value on the models updated at the parameter servers, i.e. $\{\mathbf{w}_k : k = 1, \cdots, K\}$:

$$\begin{aligned}\mathbb{E}f(\mathbf{w}_{k+1}) \leq \mathbb{E}f(\mathbf{w}_k) &- \frac{1}{2}\eta\sum_{t=0}^{\tau-1}\mathbb{E}\left\|\nabla f(\overline{\mathbf{w}}_{k,t})\right\|^2 \\ &- \eta\frac{1}{2n}\left(1 - L\left(1+\frac{1}{n}q\tau + 4\frac{n-r}{r(n-1)}(1+q)\tau\right)\eta - 2L^2\tau(\tau-1)\eta^2\right) \\ &\times \sum_{t=0}^{\tau-1}\sum_{i\in[n]}\mathbb{E}\left\|\nabla f\left(\mathbf{w}_{k,t}^{(i)}\right)\right\|^2 \\ &+ \eta^2\frac{L}{2}(1+q)\tau\left(\frac{\sigma^2}{m} + 4\frac{\sigma^2}{r}\frac{n-r}{n-1}\right) + \eta^3\frac{\sigma^2}{m}(m+1)\frac{\tau(\tau-1)}{2}L^2.\end{aligned} \tag{A.63}$$

For sufficiently small η, such that

$$1 - L\eta - L\left(\frac{1}{n}q + 4\frac{n-r}{r(n-1)}(1+q)\right)\tau\eta - 2L^2\tau(\tau-1)\eta^2 \geq 0, \tag{A.64}$$

we have

$$\begin{aligned}\mathbb{E}f(\mathbf{w}_{k+1}) \leq \mathbb{E}f(\mathbf{w}_k) - \frac{1}{2}\eta\sum_{t=0}^{\tau-1}\mathbb{E}\left\|\nabla f(\overline{\mathbf{w}}_{k,t})\right\|^2 \\ + \eta^2\frac{L}{2}(1+q)\tau\left(\frac{\sigma^2}{n} + 4\frac{\sigma^2}{r}\frac{n-r}{n-1}\right) + \eta^3\frac{\sigma^2}{n}(n+1)\frac{\tau(\tau-1)}{2}L^2.\end{aligned} \tag{A.65}$$

In Section A.2.1 we show that if the stepsize is picked as $\eta = {}^1\!/\!_{L\sqrt{T}}$ and the T ans τ satisfy the condition (2.16) in Theorem 2.2, then (A.64) also holds. Now summing (A.65) over $k = 0, \cdots, K-1$ and rearranging the terms yield that

$$\begin{aligned}&\frac{1}{2}\eta\sum_{k=0}^{K-1}\sum_{t=0}^{\tau-1}\mathbb{E}\left\|\nabla f(\overline{\mathbf{w}}_{k,t})\right\|^2 \\ &\leq f(\mathbf{w}_0) - f^* + K\eta^2\frac{L}{2}(1+q)\tau\left(\frac{\sigma^2}{n} + 4\frac{\sigma^2}{r}\frac{n-r}{n-1}\right) + K\eta^3\frac{\sigma^2}{n}(n+1)\frac{\tau(\tau-1)}{2}L^2,\end{aligned} \tag{A.66}$$

or

$$\begin{aligned}&\frac{1}{K\tau}\sum_{k=0}^{K-1}\sum_{t=0}^{\tau-1}\mathbb{E}\left\|\nabla f(\overline{\mathbf{w}}_{k,t})\right\|^2 \\ &\leq \frac{2(f(\mathbf{w}_0) - f^*)}{\eta K\tau} + \eta L(1+q)\left(\frac{\sigma^2}{n} + 4\frac{\sigma^2}{r}\frac{n-r}{n-1}\right) + \eta^2\frac{\sigma^2}{n}(n+1)(\tau-1)L^2.\end{aligned} \tag{A.67}$$

Picking the stepsize $\eta = {}^1\!/\!_{L\sqrt{T}} = {}^1\!/\!_{L\sqrt{K\tau}}$ results in the following convergence rate:

$$\frac{1}{T}\sum_{k=0}^{K-1}\sum_{t=0}^{\tau-1}\mathbb{E}\left\|\nabla f(\overline{\mathbf{w}}_{k,t})\right\|^2$$

$$\leq \frac{2L(f(\mathbf{w}_0) - f^*)}{\sqrt{T}} + (1+q)\left(\frac{\sigma^2}{n} + \frac{\sigma^2}{r}\frac{n-r}{n-1}\right)\frac{1}{\sqrt{T}} + \frac{\sigma^2}{n}(n+1)\frac{\tau-1}{T}, \tag{A.68}$$

which completes the proof of Theorem 2.2.

A.2.1 Discussion on stepsize η

Here, we consider the constraint on the stepsize derived in (A.64) and show that if η is picked according to Theorem 2.2, then it also satisfies (A.64). First, let the stepsize satisfy $1 - L\eta \geq 0.1$. Now, if the following holds

$$L\left(\frac{1}{n}q + 4\frac{n-r}{r(n-1)}(1+q)\right)\tau\eta + 2L^2(\tau\eta)^2 \leq 0.1, \tag{A.69}$$

the condition in (A.64) also holds. It is straightforward to see when (A.69) holds. To do so, consider the following quadratic inequality in terms of $y = \eta\tau$:

$$2L^2y^2 + LB_2y - 0.1 \leq 0, \tag{A.70}$$

where

$$B_2 \coloneqq \frac{1}{n}q + 4\frac{n-r}{r(n-1)}(1+q). \tag{A.71}$$

We can solve the quadratic form in (A.70) for $y = \eta\tau$ which yields

$$\eta\tau \leq \frac{\sqrt{B_2^2 + 0.8} - B_2}{4L}. \tag{A.72}$$

This implies that if the parameter τ and the stepsize η satisfy (A.72) and $\eta \leq {}^{0.9}/_{L}$, then the condition (A.64) is satisfied. In particular, for our pick of $\eta = {}^{1}/_{L\sqrt{T}}$, the condition

$\eta \leq {}^{0.9}/_{L}$ holds if $T \geq 2$; and the constraint in (A.72) is equivalent to having

$$\tau \leq \frac{\sqrt{B_2^2 + 0.8} - B_2}{8} \sqrt{T}. \tag{A.73}$$

A.2.2 Proof of Lemma A.6

Recall that for any L-smooth function f and variables $\mathbf{w}, \mathbf{w}'$ we have

$$f(\mathbf{w}) \leq f(\mathbf{w}') + \left\langle \nabla f(\mathbf{w}'), \mathbf{w} - \mathbf{w}' \right\rangle + \frac{L}{2} \left\| \mathbf{w} - \mathbf{w}' \right\|^2. \tag{A.74}$$

Therefore, we can write

$$\begin{aligned} f(\mathbf{w}_{k+1}) &= f(\widehat{\mathbf{w}}_{k+1} + \mathbf{w}_{k+1} - \widehat{\mathbf{w}}_{k+1}) \\ &\leq f(\widehat{\mathbf{w}}_{k+1}) + \left\langle \nabla f(\widehat{\mathbf{w}}_{k+1}), \mathbf{w}_{k+1} - \widehat{\mathbf{w}}_{k+1} \right\rangle + \frac{L}{2} \left\| \mathbf{w}_{k+1} - \widehat{\mathbf{w}}_{k+1} \right\|^2. \end{aligned} \tag{A.75}$$

We take expectation of both sides of (A.75) and since $\widehat{\mathbf{w}}_{k+1}$ is unbiased for $\mathbf{w}_{k+1}$, that is $\mathbb{E}_{\mathcal{S}_k} \mathbf{w}_{k+1} = \widehat{\mathbf{w}}_{k+1}$ (See (A.12)), it yields that

$$\mathbb{E} f(\mathbf{w}_{k+1}) \leq \mathbb{E} f(\widehat{\mathbf{w}}_{k+1}) + \frac{L}{2} \mathbb{E} \left\| \widehat{\mathbf{w}}_{k+1} - \mathbf{w}_{k+1} \right\|^2. \tag{A.76}$$

Moreover, $\widehat{\mathbf{w}}_{k+1}$ is also unbiased for $\overline{\mathbf{w}}_{k,\tau}$, i.e. $\mathbb{E}_Q \widehat{\mathbf{w}}_{k+1} = \overline{\mathbf{w}}_{k,\tau}$ (See (A.13)), and since f is L-smooth, we can write

$$\mathbb{E} f(\widehat{\mathbf{w}}_{k+1}) \leq \mathbb{E} f(\overline{\mathbf{w}}_{k,\tau}) + \frac{L}{2} \mathbb{E} \left\| \widehat{\mathbf{w}}_{k+1} - \overline{\mathbf{w}}_{k,\tau} \right\|^2, \tag{A.77}$$

which together with (A.76) concludes the lemma.

A.2.3 Proof of Lemma A.7

According to the update rule in Algorithm 2.1, for every $t = 0, \cdots, \tau - 1$ the average model is

$$\overline{\mathbf{w}}_{k,t+1} = \overline{\mathbf{w}}_{k,t} - \eta \frac{1}{n} \sum_{i \in [n]} \widetilde{\nabla} f_i \left(\mathbf{w}_{k,t}^{(i)} \right). \tag{A.78}$$

Since f is L-smooth, we can write

$$f(\overline{\mathbf{w}}_{k,t+1}) \leq f(\overline{\mathbf{w}}_{k,t}) - \eta \left\langle \nabla f(\overline{\mathbf{w}}_{k,t}), \frac{1}{n} \sum_{i \in [n]} \widetilde{\nabla} f_i \left(\mathbf{w}_{k,t}^{(i)} \right) \right\rangle + \eta^2 \frac{L}{2} \left\| \frac{1}{n} \sum_{i \in [n]} \widetilde{\nabla} f_i \left(\mathbf{w}_{k,t}^{(i)} \right) \right\|^2. \tag{A.79}$$

The inner product term above can be written in expectation as follows:

$$\begin{aligned} 2\mathbb{E} \left\langle \nabla f(\overline{\mathbf{w}}_{k,t}), \frac{1}{n} \sum_{i \in [n]} \widetilde{\nabla} f_i \left(\mathbf{w}_{k,t}^{(i)} \right) \right\rangle &= \frac{1}{n} \sum_{i \in [n]} 2\mathbb{E} \left\langle \nabla f(\overline{\mathbf{w}}_{k,t}), \nabla f \left(\mathbf{w}_{k,t}^{(i)} \right) \right\rangle \\ &= \mathbb{E} \left\| \nabla f(\overline{\mathbf{w}}_{k,t}) \right\|^2 + \frac{1}{n} \sum_{i \in [n]} \mathbb{E} \left\| \nabla f \left(\mathbf{w}_{k,t}^{(i)} \right) \right\|^2 \\ &\quad - \frac{1}{n} \sum_{i \in [n]} \mathbb{E} \left\| \nabla f(\overline{\mathbf{w}}_{k,t}) - \nabla f \left(\mathbf{w}_{k,t}^{(i)} \right) \right\|^2, \end{aligned} \tag{A.80}$$

where we used the identity $2\langle \mathbf{a}, \mathbf{b} \rangle = \|\mathbf{a}\|^2 + \|\mathbf{b}\|^2 - \|\mathbf{a} - \mathbf{b}\|^2$ for any two vectors $\mathbf{a}, \mathbf{b}$. In the following, we bound each of the three terms in the RHS of (A.80). Starting with the third term, we use the smoothness assumption to write

$$\left\| \nabla f(\overline{\mathbf{w}}_{k,t}) - \nabla f \left(\mathbf{w}_{k,t}^{(i)} \right) \right\|^2 \leq L^2 \left\| \overline{\mathbf{w}}_{k,t} - \mathbf{w}_{k,t}^{(i)} \right\|^2. \tag{A.81}$$

Moreover, local models $\mathbf{w}_{k,t}^{(i)}$ and average model $\overline{\mathbf{w}}_{k,t}$ are respectively

$$\mathbf{w}_{k,t}^{(i)} = \mathbf{w}_k - \eta\left(\widetilde{\nabla} f_i(\mathbf{w}_k) + \widetilde{\nabla} f_i\left(\mathbf{w}_{k,1}^{(i)}\right) + \cdots + \widetilde{\nabla} f_i\left(\mathbf{w}_{k,t-1}^{(i)}\right)\right), \tag{A.82}$$

and

$$\overline{\mathbf{w}}_{k,t} = \mathbf{w}_{k\tau} - \eta\left(\frac{1}{n}\sum_{j\in[n]}\widetilde{\nabla} f_j(\mathbf{w}_k) + \frac{1}{n}\sum_{j\in[n]}\widetilde{\nabla} f_j\left(\mathbf{w}_{k,1}^{(j)}\right) + \cdots + \frac{1}{n}\sum_{j\in[n]}\widetilde{\nabla} f_j\left(\mathbf{w}_{k,t-1}^{(j)}\right)\right). \tag{A.83}$$

Therefore, the expected deviation of each local model form the average model can be written as

$$\begin{aligned}
&\mathbb{E}\left\|\overline{\mathbf{w}}_{k,t} - \mathbf{w}_{k,t}^{(i)}\right\|^2 \\
&\le 2\eta^2\mathbb{E}\left\|\frac{1}{n}\sum_{j\in[n]}\widetilde{\nabla} f_j(\mathbf{w}_k) + \frac{1}{n}\sum_{j\in[n]}\widetilde{\nabla} f_j\left(\mathbf{w}_{k,1}^{(j)}\right) + \cdots + \frac{1}{n}\sum_{j\in[n]}\widetilde{\nabla} f_j\left(\mathbf{w}_{k,t-1}^{(j)}\right)\right\|^2 \\
&\quad + 2\eta^2\mathbb{E}\left\|\widetilde{\nabla} f_i(\mathbf{w}_k) + \widetilde{\nabla} f_i\left(\mathbf{w}_{k,1}^{(i)}\right) + \cdots + \widetilde{\nabla} f_i\left(\mathbf{w}_{k,t-1}^{(i)}\right)\right\|^2 \\
&\le 2\eta^2\left(t\frac{\sigma^2}{n} + \left\|\frac{1}{n}\sum_{j\in[n]}\nabla f(\mathbf{w}_k) + \frac{1}{n}\sum_{j\in[n]}\nabla f\left(\mathbf{w}_{k,1}^{(j)}\right) + \cdots + \frac{1}{n}\sum_{j\in[n]}\nabla f\left(\mathbf{w}_{k,t-1}^{(j)}\right)\right\|^2\right) \\
&\quad + 2\eta^2\left(t\sigma^2 + \left\|\nabla f(\mathbf{w}_k) + \nabla f\left(\mathbf{w}_{k,1}^{(i)}\right) + \cdots + \nabla f\left(\mathbf{w}_{k,t-1}^{(i)}\right)\right\|^2\right) \\
&\le 2\eta^2 t\left(\frac{1}{n}\sum_{j\in[n]}\left\|\nabla f(\mathbf{w}_k)\right\|^2 + \frac{1}{n}\sum_{j\in[n]}\left\|\nabla f\left(\mathbf{w}_{k,1}^{(j)}\right)\right\|^2 + \cdots + \frac{1}{n}\sum_{j\in[n]}\left\|\nabla f\left(\mathbf{w}_{k,t-1}^{(j)}\right)\right\|^2\right) \\
&\quad + 2\eta^2 t\sigma^2 + 2\eta^2 t\left(\left\|\nabla f(\mathbf{w}_k)\right\|^2 + \left\|\nabla f\left(\mathbf{w}_{k,1}^{(i)}\right)\right\|^2 + \cdots + \left\|\nabla f\left(\mathbf{w}_{k,t-1}^{(i)}\right)\right\|^2\right) + 2\eta^2 t\frac{\sigma^2}{n}.
\end{aligned} \tag{A.84}$$

Summing (A.84) over all the workers $i \in [n]$ yields

$$\begin{aligned}
&\sum_{i\in[n]} \mathbb{E}\left\|\overline{\mathbf{w}}_{k,t} - \mathbf{w}_{k,t}^{(i)}\right\|^2 \\
&\leq 2\eta^2 t\sigma^2 + 2\eta^2 t \left(\sum_{j\in[n]} \left\|\nabla f(\mathbf{w}_k)\right\|^2 + \sum_{j\in[n]} \left\|\nabla f\left(\mathbf{w}_{k,1}^{(j)}\right)\right\|^2 + \cdots + \sum_{j\in[n]} \left\|\nabla f\left(\mathbf{w}_{k,t-1}^{(j)}\right)\right\|^2\right) \\
&\quad + 2\eta^2 t\sigma^2 n + 2\eta^2 t \left(\sum_{i\in[n]} \left\|\nabla f(\mathbf{w}_k)\right\|^2 + \sum_{i\in[n]} \left\|\nabla f\left(\mathbf{w}_{k,1}^{(i)}\right)\right\|^2 + \cdots + \sum_{i\in[n]} \left\|\nabla f\left(\mathbf{w}_{k,t-1}^{(i)}\right)\right\|^2\right) \\
&= 2\eta^2 t\sigma^2 (n+1) \\
&\quad + 4\eta^2 t \left(\sum_{j\in[n]} \left\|\nabla f(\mathbf{w}_k)\right\|^2 + \sum_{j\in[n]} \left\|\nabla f\left(\mathbf{w}_{k,1}^{(j)}\right)\right\|^2 + \cdots + \sum_{j\in[n]} \left\|\nabla f\left(\mathbf{w}_{k,t-1}^{(j)}\right)\right\|^2\right). \qquad \text{(A.85)}
\end{aligned}$$

Finally, summing (A.85) over $t = 0, \cdots, \tau - 1$ results in the following:

$$\begin{aligned}
&\sum_{t=0}^{\tau-1} \sum_{i\in[n]} \mathbb{E}\left\|\overline{\mathbf{w}}_{k,t} - \mathbf{w}_{k,t}^{(i)}\right\|^2 \\
&\leq 2\eta^2 \sigma^2 (n+1) \sum_{t=0}^{\tau-1} t \\
&\quad + 4\eta^2 \sum_{t=0}^{\tau-1} t \left(\sum_{j\in[n]} \left\|\nabla f(\mathbf{w}_k)\right\|^2 + \sum_{j\in[n]} \left\|\nabla f\left(\mathbf{w}_{k,1}^{(j)}\right)\right\|^2 + \cdots + \sum_{j\in[n]} \left\|\nabla f\left(\mathbf{w}_{k,t-1}^{(j)}\right)\right\|^2\right) \\
&\leq \eta^2\sigma^2(n+1)\tau(\tau-1) + 2\eta^2\tau(\tau-1) \sum_{t=0}^{\tau-2} \sum_{i\in[n]} \left\|\nabla f\left(\mathbf{w}_{k,t}^{(i)}\right)\right\|^2. \qquad \text{(A.86)}
\end{aligned}$$

Next, we bound the third term in (D.1). Using Assumption 2.3 we have

$$\begin{aligned}
\mathbb{E}\left\|\frac{1}{n}\sum_{i\in[n]} \widetilde{\nabla} f_i\left(\mathbf{w}_{k,t}^{(i)}\right)\right\|^2 &= \mathbb{E}\left\|\frac{1}{n}\sum_{i\in[n]} \nabla f\left(\mathbf{w}_{k,t}^{(i)}\right)\right\|^2 + \mathbb{E}\left\|\frac{1}{n}\sum_{i\in[n]} \widetilde{\nabla} f_i\left(\mathbf{w}_{k,t}^{(i)}\right) - \nabla f\left(\mathbf{w}_{k,t}^{(i)}\right)\right\|^2 \\
&\leq \frac{1}{n}\sum_{i\in[n]} \mathbb{E}\left\|\nabla f\left(\mathbf{w}_{k,t}^{(i)}\right)\right\|^2 + \frac{\sigma^2}{n}. \qquad \text{(A.87)}
\end{aligned}$$

Summing (A.87) over iterations $t = 0, \cdots, \tau - 1$ yields

$$\sum_{t=0}^{\tau-1} \eta^2 \frac{L}{2} \mathbb{E}\left\| \frac{1}{n} \sum_{i\in[n]} \widetilde{\nabla} f_i \left(\mathbf{w}_{k,t}^{(i)}\right) \right\|^2 \leq \eta^2 \frac{L}{2} \frac{1}{n} \sum_{t=0}^{\tau-1} \sum_{i\in[n]} \mathbb{E}\left\| \nabla f \left(\mathbf{w}_{k,t}^{(i)}\right)\right\|^2 + \eta^2 \frac{L}{2} \frac{\sigma^2}{n} \tau. \tag{A.88}$$

Now we can sum (D.1) for $t = 0, \cdots, \tau - 1$ and use the results in (A.86) and (A.88) to conclude:

$$\begin{aligned}
\mathbb{E} f(\overline{\mathbf{w}}_{k,\tau}) \leq{}& \mathbb{E} f(\mathbf{w}_k) - \frac{1}{2}\eta \sum_{t=0}^{\tau-1} \mathbb{E}\left\|\nabla f(\overline{\mathbf{w}}_{k,t})\right\|^2 - \frac{1}{2n}\eta \sum_{t=0}^{\tau-1}\sum_{i\in[n]} \mathbb{E}\left\|\nabla f\left(\mathbf{w}_{k,t}^{(i)}\right)\right\|^2 \\
&+ \frac{1}{2n}\eta \sum_{t=0}^{\tau-1}\sum_{i\in[n]} \mathbb{E}\left\|\nabla f(\overline{\mathbf{w}}_{k,t}) - \nabla f\left(\mathbf{w}_{k,t}^{(i)}\right)\right\|^2 + \sum_{t=0}^{\tau-1} \eta^2 \frac{L}{2} \mathbb{E}\left\| \frac{1}{n}\sum_{i\in[n]} \widetilde{\nabla} f_i\left(\mathbf{w}_{k,t}^{(i)}\right)\right\|^2 \\
\leq{}& \mathbb{E} f(\mathbf{w}_k) - \frac{1}{2}\eta \sum_{t=0}^{\tau-1} \mathbb{E}\left\|\nabla f(\overline{\mathbf{w}}_{k,t})\right\|^2 \\
&- \eta\left(\frac{1}{2n} - \frac{1}{2n}L\eta - \frac{1}{n}L^2\tau(\tau-1)\eta^2\right) \sum_{t=0}^{\tau-1}\sum_{i\in[n]} \mathbb{E}\left\|\nabla f\left(\mathbf{w}_{k,t}^{(i)}\right)\right\|^2 \\
&+ \eta^2 \frac{L}{2}\frac{\sigma^2}{n}\tau + \eta^3 \frac{\sigma^2}{n}(n+1)\frac{\tau(\tau-1)}{2}L^2.
\end{aligned} \tag{A.89}$$

A.2.4 Proof of Lemma A.8

According to definitions in (A.1) and using Assumption 2.1 we have

$$\mathbb{E}\left\|\widehat{\mathbf{w}}_{k+1} - \overline{\mathbf{w}}_{k,\tau}\right\|^2 \leq \frac{1}{n^2}\sum_{i\in[n]} q\, \mathbb{E}\left\|\mathbf{w}_{k,\tau}^{(i)} - \mathbf{w}_k\right\|^2. \tag{A.90}$$

Using the model update in (A.82) and Assumption 2.3, we can write

$$\begin{aligned}
\mathbb{E}\left\|\mathbf{w}_{k,\tau}^{(i)} - \mathbf{w}_k\right\|^2 &= \eta^2 \mathbb{E}\left\|\widetilde{\nabla} f_i(\mathbf{w}_k) + \widetilde{\nabla} f_i\left(\mathbf{w}_{k,1}^{(i)}\right) + \cdots + \widetilde{\nabla} f_i\left(\mathbf{w}_{k,\tau-1}^{(i)}\right)\right\|^2 \\
&= \eta^2 \mathbb{E}\left\|\widetilde{\nabla} f_i(\mathbf{w}_k) - \nabla f(\mathbf{w}_k) + \cdots + \widetilde{\nabla} f_i\left(\mathbf{w}_{k,\tau-1}^{(i)}\right) - \nabla f\left(\mathbf{w}_{k,\tau-1}^{(i)}\right)\right\|^2
\end{aligned}$$

$$+\eta^2\left\|\nabla f(\mathbf{w}_{k\tau}) + \cdots + \nabla f\left(\mathbf{w}_{k,\tau-1}^{(i)}\right)\right\|^2$$

$$\leq \eta^2\sigma^2\tau + \eta^2\tau\sum_{t=0}^{\tau-1}\left\|\nabla f\left(\mathbf{w}_{k,t}^{(i)}\right)\right\|^2. \tag{A.91}$$

Summing (A.91) over all workers $i \in [n]$ and using (A.90) yields

$$\mathbb{E}\left\|\widehat{\mathbf{w}}_{k+1} - \overline{\mathbf{w}}_{k,\tau}\right\|^2 \leq q\frac{\sigma^2}{n}\tau\eta^2 + q\frac{1}{n^2}\tau\eta^2\sum_{i\in[n]}\sum_{t=0}^{\tau-1}\left\|\nabla f\left(\mathbf{w}_{k,t}^{(i)}\right)\right\|^2, \tag{A.92}$$

as desired in Lemma A.8.

A.2.5 Proof of Lemma A.9

The steps to prove the bound in (A.44) for strongly convex losses in Lemma A.4 can also be applied for non-convex losses. That is, we can use (A.44) and together with (A.91) conclude the following:

$$\begin{aligned}\mathbb{E}\|\widehat{\mathbf{w}}_{k+1} - \mathbf{w}_{k+1}\|^2 &\leq \frac{1}{r(n-1)}\left(1-\frac{r}{n}\right)4(1+q)\sum_{i\in[n]}\left\|\mathbf{w}_{k,\tau}^{(i)} - \mathbf{w}_k\right\|^2 \\ &\leq \frac{1}{r(n-1)}\left(1-\frac{r}{n}\right)4(1+q)\left\{n\sigma^2\tau\eta^2 + \tau\eta^2\sum_{i\in[n]}\sum_{t=0}^{\tau-1}\left\|\nabla f\left(\mathbf{w}_{k,t}^{(i)}\right)\right\|^2\right\}.\end{aligned} \tag{A.93}$$

Appendix B

Supplements to Chapter 3

B.1 Proof of Proposition 3.1

Let us present and prove the following lemma which includes the claim in Proposition 3.1.

Lemma B.1 *Consider two subsets of nodes $\mathcal{N}_m \subseteq \mathcal{N}_n$ and assume that model $\mathbf{w}_m$ attains the statistical accuracy for the empirical risk associated with nodes in $\mathcal{N}_m$, that is, $\|\nabla L_m(\mathbf{w}_m)\|^2 \leq 2\mu V_{ms}$ where the loss function ℓ is μ-strongly convex. Then the suboptimality of $\mathbf{w}_m$ for risk L_n, i.e., $L_n(\mathbf{w}_m) - L_n(\mathbf{w}_n^*)$ is w.h.p. bounded above as follows:*

$$L_n(\mathbf{w}_m) - L_n(\mathbf{w}_n^*) \leq \frac{2(n-m)}{n}\left(V_{(n-m)s} + V_{ms}\right) + V_{ms}. \tag{B.1}$$

Moreover, norms of local and global gradients are upper-bounded w.h.p. as follows:

$$\|\nabla L_n(\mathbf{w}_m)\|^2 \leq 2\left(\frac{n-m}{n}\right)^2\left(V_{(n-m)s}^{1/2} + V_{ms}^{1/2}\right)^2 + 4\mu V_{ms}, \tag{B.2}$$

and

$$\|\nabla L^i(\mathbf{w}_m)\|^2 \leq 3(2\mu+1)V_{ms} + 3V_s. \tag{B.3}$$

Proof: We begin the proof of Lemma B.1 by proving the inequality in (B.1). Let us decompose the sub-otpimiality error $L_n(\mathbf{w}_m) - L_n(\mathbf{w}_n^*)$ to four difference terms as follows:

$$\begin{aligned} L_n(\mathbf{w}_m) - L_n(\mathbf{w}_n^*) &= L_n(\mathbf{w}_m) - L_m(\mathbf{w}_m) + L_m(\mathbf{w}_m) - L_m(\mathbf{w}_m^*) \\ &\quad + L_m(\mathbf{w}_m^*) - L_m(\mathbf{w}_n^*) + L_m(\mathbf{w}_n^*) - L_n(\mathbf{w}_n^*). \end{aligned} \tag{B.4}$$

From definition of local empirical risks in (3.1), the difference of local risks $L_n(\mathbf{w})$ and $L_m(\mathbf{w})$ for any $\mathbf{w}$ can be bounded w.h.p. as follows:

$$\begin{aligned} L_n(\mathbf{w}) - L_m(\mathbf{w}) &\leq \left| L_n(\mathbf{w}) - L_m(\mathbf{w}) \right| \\ &= \left| \frac{1}{n} \sum_{i\in\mathcal{N}_n} L^i(\mathbf{w}) - \frac{1}{m} \sum_{i\in\mathcal{N}_m} L^i(\mathbf{w}) \right| \\ &= \left| \frac{1}{n} \sum_{i\in\mathcal{N}_n\backslash\mathcal{N}_m} L^i(\mathbf{w}) - \frac{n-m}{n} \cdot \frac{1}{m} \sum_{i\in\mathcal{N}_m} L^i(\mathbf{w}) \right| \\ &= \frac{n-m}{n} \left| \frac{1}{n-m} \sum_{i\in\mathcal{N}_n\backslash\mathcal{N}_m} L^i(\mathbf{w}) - \frac{1}{m} \sum_{i\in\mathcal{N}_m} L^i(\mathbf{w}) \right| \\ &\leq \frac{n-m}{n} \left| \frac{1}{n-m} \sum_{i\in\mathcal{N}_n\backslash\mathcal{N}_m} L^i(\mathbf{w}) - L(\mathbf{w}) \right| + \frac{n-m}{n} \left| \frac{1}{m} \sum_{i\in\mathcal{N}_m} L^i(\mathbf{w}) - L(\mathbf{w}) \right| \\ &\leq \frac{n-m}{n} \left(V_{(n-m)s} + V_{ms} \right), \end{aligned} \tag{B.5}$$

where the last inequality is implied from Assumption 3.2 when applied to empirical risks $\frac{1}{n-m}\sum_{i\in\mathcal{N}_n\backslash\mathcal{N}_m} L^i(\mathbf{w})$ and $\frac{1}{m}\sum_{i\in\mathcal{N}_m} L^i(\mathbf{w})$ with $(n-m)s$ and ms samples, respectfully. We now proceed to bound the next term in (B.4), that is the optimality gap $L_m(\mathbf{w}_m) -$

$L_m(\mathbf{w}_m^*)$. Using the strong convexity assumption in Assumption 3.1 and the condition $\|\nabla L_m(\mathbf{w}_m)\|^2 \leq 2\mu V_{ms}$ assumed to hold in the statement of the lemma, we can write

$$L_m(\mathbf{w}_m) - L_m(\mathbf{w}_m^*) \leq \frac{1}{2\mu}\left\|\nabla L_m(\mathbf{w}_m)\right\|^2 \leq \frac{2\mu V_{ms}}{2\mu} = V_{ms}. \tag{B.6}$$

Next, the term $L_m(\mathbf{w}_m^*) - L_m(\mathbf{w}_n^*)$ in (B.4) can be simply bounded as $L_m(\mathbf{w}_m^*) - L_m(\mathbf{w}_n^*) \leq 0$, since $\mathbf{w}_m^*$ is the minimizer of $L_m(\mathbf{w})$. Finally, to bound $L_m(\mathbf{w}_n^*) - L_n(\mathbf{w}_n^*)$ in (B.4), we use the result in (B.5) which holds for any $\mathbf{w}$ and here we pick $\mathbf{w} = \mathbf{w}_n^*$ to conclude

$$L_m(\mathbf{w}_n^*) - L_n(\mathbf{w}_n^*) \leq \frac{n-m}{n}\left(V_{(n-m)s} + V_{ms}\right). \tag{B.7}$$

Putting the upper bounds for the four terms in (B.4) together proves inequality (B.1) which is the same claim as in Proposition 3.1.

Next we prove inequality (B.2) by first noting the following:

$$\left\|\nabla L_n(\mathbf{w}_m)\right\|^2 \leq 2\left\|\nabla L_n(\mathbf{w}_m) - \nabla L_m(\mathbf{w}_m)\right\|^2 + 2\left\|\nabla L_m(\mathbf{w}_m)\right\|^2. \tag{B.8}$$

The first term $\|\nabla L_n(\mathbf{w}_m) - \nabla L_m(\mathbf{w}_m)\|$ can be bounded as follows:

$$\begin{aligned}
\left\|\nabla L_n(\mathbf{w}_m) - \nabla L_m(\mathbf{w}_m)\right\| &= \left\|\frac{1}{n}\sum_{i\in\mathcal{N}_n}\nabla L^i(\mathbf{w}) - \frac{1}{m}\sum_{i\in\mathcal{N}_m}\nabla L^i(\mathbf{w})\right\| \\
&= \left\|\frac{1}{n}\sum_{i\in\mathcal{N}_n\setminus\mathcal{N}_m}\nabla L^i(\mathbf{w}) - \frac{n-m}{n}\cdot\frac{1}{m}\sum_{i\in\mathcal{N}_m}\nabla L^i(\mathbf{w})\right\| \\
&= \frac{n-m}{n}\left\|\frac{1}{n-m}\sum_{i\in\mathcal{N}_n\setminus\mathcal{N}_m}\nabla L^i(\mathbf{w}) - \frac{1}{m}\sum_{i\in\mathcal{N}_m}\nabla L^i(\mathbf{w})\right\| \\
&\leq \frac{n-m}{n}\left\|\frac{1}{n-m}\sum_{i\in\mathcal{N}_n\setminus\mathcal{N}_m}\nabla L^i(\mathbf{w}) - \nabla L(\mathbf{w})\right\|
\end{aligned}$$

$$
\begin{aligned}
&+ \frac{n-m}{n}\left\|\frac{1}{m}\sum_{i\in\mathcal{N}_m}\nabla L^i(\mathbf{w}) - \nabla L(\mathbf{w})\right\| \\
&\leq \frac{n-m}{n}\left(V_{(n-m)s}^{1/2} + V_{ms}^{1/2}\right). \qquad \text{(B.9)}
\end{aligned}
$$

In the last inequality above, we used Assumption 3.2 to upper-bound the approximation of empirical gradients for $(n-m)s$ and ms samples. Together with (B.8) and the assumption of the lemma, that is $\left\|\nabla L_m(\mathbf{w}_m)\right\|^2 \leq 2\mu V_{ms}$, the claim in (B.2) is concluded:

$$
\|\nabla L_n(\mathbf{w}_m)\|^2 \leq 2\left(\frac{n-m}{n}\right)^2\left(V_{(n-m)s}^{1/2} + V_{ms}^{1/2}\right)^2 + 4\mu V_{ms}. \qquad \text{(B.10)}
$$

Finally, we prove the claim in inequality (B.3) by bounding node i's local gradient $\nabla L^i(\mathbf{w}_m)$ as follows:

$$
\begin{aligned}
\left\|\nabla L^i(\mathbf{w}_m)\right\|^2 &\leq 3\left\|\nabla L^i(\mathbf{w}_m) - \nabla L(\mathbf{w}_m)\right\|^2 + 3\left\|\nabla L_m(\mathbf{w}_m) - \nabla L(\mathbf{w}_m)\right\|^2 + 3\left\|\nabla L_m(\mathbf{w}_m)\right\|^2 \\
&\leq 3V_s + 3V_{ms} + 6\mu V_{ms} \\
&= 3(2\mu+1)V_{ms} + 3V_s, \qquad \text{(B.11)}
\end{aligned}
$$

where we used Assumption 3.2 to upper-bound the approximation error of empirical gradients for node i with s samples and m nodes with ms samples.

■

B.2 Proof of Theorem 3.1

Consider a stage of Algorithm 3.2 running with n participating nodes. More precisely, n nodes in $\{1, \cdots, n\}$ begin a sequence of local and global model updates according to `FedGATE` initialized with $\mathbf{w}_m$ obtained from the previous stage ($n = 2m$). After R_n

communication rounds each with τ_n local updates, the final sub-optimality error is upper-bounded as follows: (refer to Algorithm 2 and Theorem E.6 in [56] with no quantization)

$$\begin{aligned}\mathbb{E}[L_n(\mathbf{w}) - L_n(\mathbf{w}_n^*)] \le & \left(1 - \frac{1}{3}\mu\eta_n\gamma_n\tau_n\right)^{R_n} \big(L_n(\mathbf{w}_m) - L_n(\mathbf{w}_n^*)\big) \\ & + 24\kappa^3 L\tau_n^2\eta_n^2 \frac{1}{n}\sum_{i=1}^{n}\left\|\nabla L^i(\mathbf{w}_m)\right\|^2 + 24\kappa L\tau_n^2\eta_n^2\left\|\nabla L_n(\mathbf{w}_m)\right\|^2 \\ & + 24\kappa^2 L^2\tau_n^2\eta_n^3\sigma^2 + 15\kappa L^3\tau_n^3\eta_n^2(\eta_n\gamma_n)^2\frac{\sigma^2}{n} + \frac{L}{2}\eta_n\gamma_n\frac{\sigma^2}{n}, \qquad \text{(B.12)}\end{aligned}$$

where two stepsizes η_n, γ_n satisfy the following conditions:

$$1 - L\eta_n\gamma_n\tau_n + \frac{10\eta_n^2\tau_n^4 L^4(\eta_n\gamma_n)^2}{1 - \mu\tau_n\gamma_n\eta_n + 20\mu\gamma_n\eta_n^3 L^2\eta_n^3} \le 1 \qquad \& \qquad 30\eta_n^2 L^2\tau_n^2 \le 1. \qquad \text{(B.13)}$$

To satisfy the two conditions in (B.13), we can pick stepsizes η_n, γ_n such that

$$2\eta_n\gamma_n\tau_n L = 1 \qquad \& \qquad 30\eta_n^2 L^2\tau_n^2 \le 1. \qquad \text{(B.14)}$$

Now we use the result in Lemma B.1 and put $n = 2m$ to conclude that

$$\begin{aligned} L_n(\mathbf{w}_m) - L_n(\mathbf{w}_n^*) &\le 3V_{ms}, \\ \|\nabla L_n(\mathbf{w}_m)\|^2 &\le 2(2\mu+1)V_{ms}, \\ \|\nabla L^i(\mathbf{w}_m)\|^2 &\le 3(2\mu+1)V_{ms} + 3V_s. \qquad \text{(B.15)}\end{aligned}$$

Substituting the three inequalities (B.15) in the sub-optimality error (B.12) yields that

$$\begin{aligned}\mathbb{E}[L_n(\mathbf{w}) - L_n(\mathbf{w}_n^*)] \le & \, 3\left(1 - \frac{1}{3}\mu\eta_n\gamma_n\tau_n\right)^{R_n} V_{ms} \\ & + 72\kappa^3 L\tau_n^2\eta_n^2\big((2\mu+1)V_{ms} + V_s\big) + 48(2\mu+1)\kappa L\tau_n^2\eta_n^2 V_{ms}\end{aligned}$$

$$+ 24\kappa^2 L^2 \tau_n^2 \eta_n^3 \sigma^2 + 15\kappa L^3 \tau_n^3 \eta_n^2 (\eta_n \gamma_n)^2 \frac{\sigma^2}{n} + \frac{L}{2}\eta_n\gamma_n \frac{\sigma^2}{n}. \tag{B.16}$$

We use the fact that $2\eta_n\gamma_n\tau_n L = 1$ and rearrange the terms in (B.16) and rewrite it as follows:

$$\begin{aligned}\mathbb{E}[L_n(\mathbf{w}) - L_n(\mathbf{w}_n^*)] \leq 3\left(1 - \frac{1}{6\kappa}\right)^{R_n} V_{ms} \\ + 72\kappa^3 L\tau_n^2\eta_n^2 V_s + 24(3\kappa^2+2)(2\mu+1)\kappa L\tau_n^2\eta_n^2 V_{ms} \\ + 24\kappa^2 L^2\tau_n^2\eta_n^3\sigma^2 + \frac{15}{4}\kappa L\tau_n\eta_n^2\frac{\sigma^2}{n} + \frac{L}{2}\eta_n\gamma_n\frac{\sigma^2}{n}.\end{aligned} \tag{B.17}$$

To ensure that a model $\mathbf{w} = \mathbf{w}_n$ attains the statistical accuracy of $L_n(\mathbf{w})$, i.s. $\mathbb{E}[L_n(\mathbf{w}_n) - L_n(\mathbf{w}_n^*)] \leq V_{ns}$, it suffices to have each of the six terms in RHS of (B.17) less than or equal to $V_{ns}/6$. That is,

$$\begin{aligned} 3\left(1 - \frac{1}{6\kappa}\right)^{R_n} V_{ms} &\leq \frac{V_{ns}}{6}, \\ 72\kappa^3 L\tau_n^2\eta_n^2 V_s &\leq \frac{V_{ns}}{6}, \\ 24(3\kappa^2+2)(2\mu+1)\kappa L\tau_n^2\eta_n^2 V_{ms} &\leq \frac{V_{ns}}{6}, \\ 24\kappa^2 L^2\tau_n^2\eta_n^3\sigma^2 &\leq \frac{V_{ns}}{6}, \\ \frac{15}{4}\kappa L\tau_n\eta_n^2\frac{\sigma^2}{n} &\leq \frac{V_{ns}}{6}, \\ \frac{L}{2}\eta_n\gamma_n\frac{\sigma^2}{n} &\leq \frac{V_{ns}}{6}, \end{aligned} \tag{B.18}$$

where $n = 2m$ and $V_{ns} = \frac{c}{ns}$ for any n. One can check that the following picks for the stepsizes η_n, γ_n satisfies all the conditions in (B.13) and (B.18):

$$\eta_n = \frac{\alpha_n}{\tau_n\sqrt{n}},$$

$$\gamma_n = \frac{\sqrt{n}}{2\alpha_n L}, \tag{B.19}$$

where

$$\alpha_n \leq \min\left\{ \frac{1}{12\sqrt{3}\kappa\sqrt{\kappa L}}, \frac{\sqrt{n}}{12\sqrt{2(3\kappa^2+2)(2\mu+1)\kappa L}}, \left(\frac{\sqrt{n}}{96\kappa^2 L^2}\right)^{1/3}, \frac{\sqrt{n}}{\sqrt{15c\kappa L}}, \frac{\sqrt{n}}{L\sqrt{30}} \right\}. \tag{B.20}$$

Moreover, the first and the last conditions in (B.18) yield that the number of local updates and the number of communication rounds for the stage with n participating nodes are

$$\begin{aligned} \tau_n &= \frac{3}{2}\frac{\sigma^2 s}{c}, \\ R_n &= 12\kappa \ln(6). \end{aligned} \tag{B.21}$$

B.3 Proof of Proposition 3.2

In order to characterize the runtime of FedGATE, we first need to determine its two major parameters τ and R. More precisely, we run FedGATE algorithm with all the N available nodes while initialized with arbitrary model $\mathbf{w}_0$ and look for τ, R after which the global model $\tilde{\mathbf{w}}$ attains the statistical accuracy of $L_N(\mathbf{w})$, i.e. $\mathbb{E}[L_N(\tilde{\mathbf{w}}) - L_N(\mathbf{w}_N^*)] \leq V_{Ns}$. We use the convergence guarantee of FedGATE [56] in (B.12) with $n = N$ nodes, that is,

$$\begin{aligned} \mathbb{E}[L_N(\mathbf{w}) - L_N(\mathbf{w}_N^*)] \leq &\left(1 - \frac{1}{3}\mu\eta\gamma\tau\right)^R \left(L_N(\mathbf{w}_0) - L_N(\mathbf{w}_N^*)\right) \\ &+ 24\kappa^3 L\tau^2\eta^2 \frac{1}{N}\sum_{i=1}^{N} \left\|\nabla L^i(\mathbf{w}_0)\right\|^2 + 24\kappa L\tau^2\eta^2 \left\|\nabla L_N(\mathbf{w}_0)\right\|^2 \\ &+ 24\kappa^2 L^2\tau^2\eta^3\sigma^2 + 15\kappa L^3\tau^3\eta^2(\eta\gamma)^2\frac{\sigma^2}{N} + \frac{L}{2}\eta\gamma\frac{\sigma^2}{N}, \end{aligned} \tag{B.22}$$

where the stepsizes η, γ satisfy the following conditions:

$$1 - L\eta\gamma\tau + \frac{10\eta^2\tau^4 L^4(\eta\gamma)^2}{1 - \mu\tau\gamma\eta + 20\mu\gamma\eta^3 L^2\eta^3} \leq 1 \qquad \text{and} \qquad 30\eta^2 L^2\tau^2 \leq 1. \tag{B.23}$$

Note that the initial model $\mathbf{w}_0$ is arbitrary and therefore the initial sub-optimality error can be treated as a constant (and not scaling with N), that is, $L_N(\mathbf{w}_0) - L_N(\mathbf{w}_N^*) = \Delta_0$ for a constant $\Delta_0 = \mathcal{O}(1)$. Similarly, we can assume that $\frac{1}{N}\sum_{i=1}^{N}\left\|\nabla L^i(\mathbf{w}_0)\right\|^2 = \Delta_0'$ for a constant $\Delta_0' = \mathcal{O}(1)$ which also yields that $\left\|\nabla L_N(\mathbf{w}_0)\right\|^2 \leq \Delta_0'$. We can therefore further simplify (B.22) and write

$$\begin{aligned}\mathbb{E}[L_N(\mathbf{w}) - L_N(\mathbf{w}_N^*)] \leq \left(1 - \frac{1}{3}\mu\eta\gamma\tau\right)^R \Delta_0 + 24\kappa(\kappa^2+1)L\tau^2\eta^2\Delta_0' \\ + 24\kappa^2 L^2\tau^2\eta^3\sigma^2 + 15\kappa L^3\tau^3\eta^2(\eta\gamma)^2\frac{\sigma^2}{N} + \frac{L}{2}\eta\gamma\frac{\sigma^2}{N}.\end{aligned} \tag{B.24}$$

We furthermore pick the parameters such that $2\eta\gamma\tau L = 1$ which further simplifies (B.24) as follows:

$$\begin{aligned}\mathbb{E}[L_N(\mathbf{w}) - L_N(\mathbf{w}_N^*)] \leq \left(1 - \frac{1}{6\kappa}\right)^R \Delta_0 + 24\kappa(\kappa^2+1)L\tau^2\eta^2\Delta_0' \\ + 24\kappa^2 L^2\tau^2\eta^3\sigma^2 + \frac{15}{4}\kappa L\tau\eta^2\frac{\sigma^2}{N} + \frac{L}{2}\eta\gamma\frac{\sigma^2}{N}.\end{aligned} \tag{B.25}$$

Now to ensure that $\mathbb{E}[L_N(\tilde{\mathbf{w}}) - L_N(\mathbf{w}_N^*)] \leq V_{Ns}$ holds for a model $\tilde{\mathbf{w}}$ in (B.25), it suffices to satisfy the following inequalities:

$$\begin{aligned}\left(1 - \frac{1}{6\kappa}\right)^R \Delta_0 &\leq \frac{V_{Ns}}{5}, \\ 24\kappa(\kappa^2+1)L\tau^2\eta^2\Delta_0' &\leq \frac{V_{Ns}}{5}, \\ 24\kappa^2 L^2\tau^2\eta^3\sigma^2 &\leq \frac{V_{Ns}}{5},\end{aligned}$$

$$\begin{aligned}\frac{15}{4}\kappa L\tau\eta^2\frac{\sigma^2}{N} &\leq \frac{V_{Ns}}{5},\\ \frac{L}{2}\eta\gamma\frac{\sigma^2}{N} &\leq \frac{V_{Ns}}{5},\end{aligned}\tag{B.26}$$

with $V_{Ns} = \frac{c}{Ns}$. The following picks for the stepsizes satisfy the aforementioned conditions

$$\begin{aligned}\eta &= \frac{\alpha}{\tau\sqrt{Ns}},\\ \gamma &= \frac{\sqrt{Ns}}{2\alpha L},\end{aligned}\tag{B.27}$$

where

$$\alpha \leq \min\left\{\frac{\sqrt{c}}{2\sqrt{30}\sqrt{\kappa(\kappa^2+1)L\Delta_0'}}, \left(\frac{\sqrt{Ns}}{96\kappa^2L^2}\right)^{1/3}, \frac{\sqrt{Ns}}{\sqrt{15\kappa L}}, \frac{\sqrt{Ns}}{L\sqrt{30}}\right\}.\tag{B.28}$$

Moreover, the number of local updates and the number of communication rounds to reach the final statistical accuracy are as follows:

$$\begin{aligned}\tau &= \frac{5}{4}\frac{\sigma^2 s}{c} = \mathcal{O}(s),\\ R &= 6\kappa\ln\left(\frac{5\Delta_0 Ns}{c}\right) = \mathcal{O}(\kappa\ln(Ns)).\end{aligned}\tag{B.29}$$

Now note that the expected runtime of each communication round of `FedGATE` is τT_N as the server has to wait for the slowest node that is node N with processing time T_N. Therefore, the total expected wall-clock time of `FedGATE` to reach the final statistical accuracy of all the samples of the N nodes in $L_N(\mathbf{w})$ is

$$\bar{T}_{\texttt{FedGATE}} = R\tau T_N = \mathcal{O}(\kappa s\sigma^2\ln(Ns)T_N),$$

as claimed in Proposition 3.2.

B.4 Proof of Theorem 3.2

Recall the result in Proposition 3.2 and the following discussion in (3.3). As discussed, the average runtime for the proposed `FLANP` with `FedGATE` in Algorithm 3.2 is as follows:

$$\bar{T}_{\texttt{FLANP}} = R_{\texttt{FLANP}}\, \tau_{\texttt{FLANP}} \sum_{i = n_0, 2n_0, 4n_0, \cdots, N} T_i = \frac{18\ln(6)}{c}\kappa s\sigma^2 \left(T_{n_0} + T_{2n_0} + \cdots + T_N\right), \quad \text{(B.30)}$$

where $R_{\texttt{FLANP}} = 12\kappa\ln(6)$ and $\tau_{\texttt{FLANP}} = 1.5s\sigma^2/c$ per Theorem 3.1. Moreover, we showed in Proposition 3.2 that the expected runtime for `FedGATE` is

$$\mathbb{E}[\bar{T}_{\texttt{FedGATE}}] = R_{\texttt{FedGATE}}\, \tau_{\texttt{FedGATE}}\, T_N = \frac{15}{2c}\kappa s\sigma^2 \ln\left(\frac{5\Delta_0 N s}{c}\right) T_N. \quad \text{(B.31)}$$

In the case that clients' computation times T_is are random, the expected runtimes are

$$\begin{aligned}\mathbb{E}[\bar{T}_{\texttt{FLANP}}] &= \frac{18\ln(6)}{c}\kappa s\sigma^2 \left(\mathbb{E}[T_{n_0}] + \mathbb{E}[T_{2n_0}] + \cdots + \mathbb{E}[T_N]\right), \\ \mathbb{E}[\bar{T}_{\texttt{FedGATE}}] &= \frac{15}{2c}\kappa s\sigma^2 \ln\left(\frac{5\Delta_0 N s}{c}\right) \mathbb{E}[T_N]. \end{aligned} \quad \text{(B.32)}$$

Therefore, in order to derive the runtime gain $\frac{\mathbb{E}[T_{\texttt{FLANP}}]}{\mathbb{E}[T_{\texttt{FedGATE}}]}$, we first characterize the ratio

$$\frac{\mathbb{E}[T_{n_0}] + \mathbb{E}[T_{2n_0}] + \cdots + \mathbb{E}[T_N]}{\mathbb{E}[T_N]}, \quad \text{(B.33)}$$

where the clients runtimes T_i are i.i.d. with random exponential distribution $\exp(\lambda)$ with rate λ. Note that we assumed that the clients are sorted with respect to their processing speeds from fastest to slowest. Here, since the computation times T_is are random, we first sort them as $T_{(1)} \leq T_{(2)} \leq T_{(3)} \leq \cdots T_{(N)}$. Without loss of generality and for simplification, let us take $n_0 = 1$ and $\lambda = 1$ and proceed to bound the ratio

$$\frac{\mathbb{E}[T_{(1)}] + \mathbb{E}[T_{(2)}] + \mathbb{E}[T_{(4)}] + \cdots + \mathbb{E}[T_{(N)}]}{\mathbb{E}[T_{(N)}]}. \tag{B.34}$$

We first provide the following facts about i.i.d. random exponential variables. If $T_i \sim \exp(1)$ are i.i.d. random variables from exponential distribution with mean value of 1, then the order statistics $T_{(i)}$ have the following properties:

$$T_{(1)} \sim \exp\left(\frac{1}{N}\right), \quad T_{(i)} - T_{(i-1)} \sim \exp\left(\frac{1}{N-i+1}\right). \tag{B.35}$$

Therefore, the expected value of client i's computation speed $T_{(i)}$ can be written as

$$\begin{aligned}
\mathbb{E}[T_{(i)}] &= \mathbb{E}[T_{(i)} - T_{(i-1)}] + \mathbb{E}[T_{(i-1)} - T_{(i-2)}] + \cdots + \mathbb{E}[T_{(2)} - T_{(1)}] + \mathbb{E}[T_{(1)}] \\
&= \frac{1}{N-i+1} + \frac{1}{N-i+2} + \cdots + \frac{1}{N-1} + \frac{1}{N} \\
&= H_N - H_{N-i},
\end{aligned} \tag{B.36}$$

for any $1 \leq n \leq N$. In above, $H_n = 1 + \frac{1}{2} + \frac{1}{3} + \cdots + \frac{1}{n}$ denotes the nth harmonic number. Now we use the bounds $\ln(n) + \gamma \leq H_n \leq \ln(n+1) + \gamma$ for each $n \geq 2$ where $\gamma \approx 0.577$ is the Euler-Mascheroni constant. For further simplification, we assume that N is a power of 2, that is $N = 2^K$ for some integer K. Therefore, we can write

$$\begin{aligned}
\mathbb{E}[T_{(1)}] &= H_N - H_{N-1} \leq \ln(N+1) - \ln(N-1), \\
\mathbb{E}[T_{(2)}] &= H_N - H_{N-2} \leq \ln(N+1) - \ln(N-2), \\
\mathbb{E}[T_{(4)}] &= H_N - H_{N-4} \leq \ln(N+1) - \ln(N-4), \\
\mathbb{E}[T_{(8)}] &= H_N - H_{N-8} \leq \ln(N+1) - \ln(N-8), \\
&\vdots
\end{aligned}$$

$$\mathbb{E}[T_{(N/2)}] = H_N - H_{N/2} \leq \ln(N+1) - \ln(N/2),$$
$$\mathbb{E}[T_{(N)}] = H_N \leq \ln(N+1) + \gamma. \tag{B.37}$$

Therefore, we can bound the numerator of the ratio in (B.34) as follows:

$$\begin{aligned}
&\mathbb{E}[T_{(1)}] + \mathbb{E}[T_{(2)}] + \mathbb{E}[T_{(4)}] + \cdots + \mathbb{E}[T_{(N)}] \\
&\quad \leq (K+1)\ln\left(2^K+1\right) + \gamma - \ln\left(\left(2^K-1\right)\left(2^K-2\right)\left(2^K-4\right)\cdots\left(2^{K-1}\right)\right) \\
&\quad \leq (K+1)\ln\left(2^K+1\right) + \gamma - (K^2-K)\ln(2) \\
&\quad \leq (K+1)\left(K\ln(2) + \frac{1}{2^K}\right) + \gamma - (K^2-K)\ln(2) \\
&\quad = K\left(2\ln(2) + \frac{1}{2^K}\right) + \frac{1}{2^K} + \gamma
\end{aligned} \tag{B.38}$$

Moreover, the denominator of the ratio in (B.34) can be bounded as follows:

$$\mathbb{E}[T_{(N)}] = H_N \geq \ln(N) + \gamma = K\ln(2) + \gamma. \tag{B.39}$$

Putting (B.38) and (B.39) together, we can bound the ratio in (B.34) as follows:

$$\frac{\mathbb{E}[T_{(1)}] + \mathbb{E}[T_{(2)}] + \mathbb{E}[T_{(4)}] + \cdots + \mathbb{E}[T_{(N)}]}{\mathbb{E}[T_{(N)}]} \leq \frac{K\left(2\ln(2) + \frac{1}{2^K}\right) + \frac{1}{2^K} + \gamma}{K\ln(2) + \gamma} \leq 2 + \frac{1}{N}. \tag{B.40}$$

Now, we are able to precisely characterize the speedup gain of `FLANP` compared to `FedGATE` according to the expressions in (B.32) and the ratio in (B.40) to conclude that

$$\begin{aligned}
\frac{\mathbb{E}[\bar{T}_{\texttt{FLANP}}]}{\mathbb{E}[\bar{T}_{\texttt{FedGATE}}]} &= \frac{12\ln(6)}{5\ln\left(5c^{-1}\Delta_0 N s\right)} \frac{\mathbb{E}[T_{(1)}] + \mathbb{E}[T_{(2)}] + \mathbb{E}[T_{(4)}] + \cdots + \mathbb{E}[T_{(N)}]}{\mathbb{E}[T_{(N)}]} \\
&\leq \frac{12\ln(6)}{5\ln\left(5c^{-1}\Delta_0 N s\right)}\left(2 + \frac{1}{N}\right)
\end{aligned}$$

$$= \mathcal{O}\left(\frac{1}{\ln(Ns)}\right), \tag{B.41}$$

which completes the proof of Theorem 3.2.

Appendix C

Supplements to Chapter 4

C.1 Preliminaries and Useful Lemmas

In this section, we provide preliminary and useful results in order to prove Theorems 4.1 and 4.2. For notational convenience, we use the following short-hand notations:

Now, we present a set of useful lemmas and observations which we will invoke to prove the convergence results for both PL-PL and nonconvex-PL loss cases. The following lemma establishes the Lipschitz gradient parameter for the global function given those of the local objectives.

Lemma C.1 *If the local functions f^is have Lipschits gradients with parameters stated in Assumption 4.3, then the global function f has also Lipschitz gradients as follows: for any $\mathbf{w}, \mathbf{w}', \Psi, \Psi'$ it holds that*

$$\begin{aligned}
\left\|\nabla_{\boldsymbol{w}} f(\mathbf{w}, \Psi) - \nabla_{\boldsymbol{w}} f(\mathbf{w}', \Psi)\right\| &\leq L_1 \left\|\mathbf{w} - \mathbf{w}'\right\|, \\
\left\|\nabla_{\boldsymbol{w}} f(\mathbf{w}, \Psi) - \nabla_{\boldsymbol{w}} f(\mathbf{w}, \Psi')\right\| &\leq \frac{L_{12}}{\sqrt{n}} \left\|\Psi - \Psi'\right\|_F, \\
\left\|\nabla_{\Psi} f(\mathbf{w}, \Psi) - \nabla_{\Psi} f(\mathbf{w}', \Psi)\right\|_F &\leq \frac{L_{21}}{\sqrt{n}} \left\|\mathbf{w} - \mathbf{w}'\right\|,
\end{aligned}$$

Notation	Description
$\boldsymbol{\psi}_t^i = \left(\Lambda_t^i, \delta_t^i\right)$	maximization variables of node i iteration t
$\Psi_t = \left(\boldsymbol{\psi}_t^1; \cdots; \boldsymbol{\psi}_t^n\right)$	concatenation of all nodes' maximization models at iteration t
$\overline{\mathbf{w}}_t = \frac{1}{n}\sum_{i\in[n]} \mathbf{w}_t^i$	average model at iteration t
$a_t = \mathbb{E}[\Phi(\overline{\mathbf{w}}_t)] - \Phi^*$	optimality gap measure between $\Phi(\overline{\mathbf{w}}_t)$ and $\min_{\mathbf{w}} \Phi(\mathbf{w})$
$b_t = \mathbb{E}[\Phi(\overline{\mathbf{w}}_t) - f(\overline{\mathbf{w}}_t, \Psi_t)]$	optimality gap measure between $f(\overline{\mathbf{w}}_t, \Psi_t)$ and $\max_\Psi f(\overline{\mathbf{w}}_t, \Psi)$
$e_t = \frac{1}{n}\sum_{i\in[n]} \mathbb{E}\left\|\mathbf{w}_t^i - \overline{\mathbf{w}}_t\right\|^2$	average deviation of the local models from the average model at iteration t
$g_t = \mathbb{E}\left\|\frac{1}{n}\sum_{i\in[n]} \nabla_{\boldsymbol{w}} f^i(\mathbf{w}_t^i, \boldsymbol{\psi}_t^i)\right\|^2$	norm squared of local gradients w.r.t $\mathbf{w}$ at iteration t
$h_t = \mathbb{E}\left\|\nabla\Phi(\overline{\mathbf{w}}_t) - \frac{1}{n}\sum_{i\in[n]} \nabla_{\boldsymbol{w}} f^i(\mathbf{w}_t^i, \boldsymbol{\psi}_t^i)\right\|^2$	norm squared of deviation in gradients w.r.t $\mathbf{w}$ of $\max_\Psi f(\overline{\mathbf{w}}_t, \Psi)$ and local functions $f^i(\mathbf{w}_t^i, \boldsymbol{\psi}_t^i)$

Table C.1: Table of notations.

$$\left\|\nabla_\Psi f(\mathbf{w}, \Psi) - \nabla_\Psi f(\mathbf{w}, \Psi')\right\|_F \leq \frac{L_2}{n}\left\|\Psi - \Psi'\right\|_F . \tag{C.1}$$

Proof: We defer the proof to Section C.4.1. ■

Recall the definition of the function $\Phi(\cdot)$, that is,

$$\Phi(\mathbf{w}) \coloneqq \max_\Psi f(\mathbf{w}, \Psi) = \max_{\boldsymbol{\psi}^1,\cdots,\boldsymbol{\psi}^n} \frac{1}{n}\sum_{i\in[n]} f^i(\mathbf{w}, \boldsymbol{\psi}^i) = \max_{(\Lambda^1,\delta^1),\cdots,(\Lambda^n,\delta^n)} \frac{1}{n}\sum_{i\in[n]} f^i(\mathbf{w}, \Lambda^i, \delta^i). \tag{C.2}$$

Next lemma shows that Φ has Lipschitz gradients and characterizes its parameter.

Lemma C.2 ([99]) *If Assumptions 4.3 and 4.4 (ii) hold, that is, the local objectives* *have Lipschitz gradients and $-f(\mathbf{w},\cdot)$ is μ_2-PL, then we have*

$$\nabla\Phi(\mathbf{w}) = \nabla_{\boldsymbol{w}} f(\mathbf{w}, \Psi^*(\mathbf{w})), \tag{C.3}$$

where $\Psi^*(\mathbf{w}) \in \arg\max_{\Psi} f(\mathbf{w},\Psi)$ for any $\mathbf{w}$. Moreover, Φ has Lipschitz gradients with parameter $L_\Phi = L_1 + \frac{L_{12}L_{21}}{2n\mu_2}$.

Proof: We defer the proof to Section C.4.2. ■

Next lemma shows the contraction of the sequence $\{\mathbb{E}[\Phi(\overline{\mathbf{w}}_t)]\}_{t\geq 0}$ when running the update rule of `FedRobust` method in Algorithm 4.1. Please refer to Table C.1 to recall the definition of h_t and g_t.

Lemma C.3 *If Assumptions 4.2 and 4.3 hold, then the iterates of* `FedRobust` *satisfy the following contraction inequality for any iteration $t \geq 0$*

$$\mathbb{E}[\Phi(\overline{\mathbf{w}}_{t+1})] - \mathbb{E}[\Phi(\overline{\mathbf{w}}_t)] \leq -\frac{\eta_1}{2}\mathbb{E}\left\|\nabla\Phi(\overline{\mathbf{w}}_t)\right\|^2 + \frac{\eta_1}{2}h_t - \frac{\eta_1}{2}\left(1-\eta_1 L_\Phi\right)g_t + \eta_1^2\frac{L_\Phi}{2}\frac{\sigma_{\boldsymbol{w}}^2}{n}. \tag{C.4}$$

Proof: We defer the proof to Section C.4.3. ■

Next lemma further bounds h_t w.r.t. the two sequences b_t and e_t.

Lemma C.4 *If Assumptions 4.3 and 4.4 (ii) hold, that is, the local objectives have Lipschitz gradients and $-f(\mathbf{w},\cdot)$ is μ_2-PL, then we have*

$$h_t \leq \frac{4L_{12}^2}{\mu_2 n}b_t + 2L_1^2 e_t. \tag{C.5}$$

Proof: We defer the proof to Section C.4.4. ■

Next lemma establishes a contraction bound on the sequence b_t.

Lemma C.5 *If Assumptions 4.2, 4.3 and 4.4 (ii) hold, then the sequence of $\{b_t\}_{t\geq 0}$ generated by the* `FedRobust` *iterations with $\eta_2 \leq 1/L_2$ satisfies the following contraction bound:*

$$b_{t+1} \leq (1-\mu_2\eta_2 n)\left(1+\eta_1\frac{4L_{12}^2}{\mu_2 n}\right) b_t + \frac{\eta_1}{2}\mathbb{E}\left\|\nabla\Phi(\overline{\mathbf{w}}_t)\right\|^2 + \frac{\eta_1^2}{2}\left(L_1+L_\Phi+2\eta_2 L_{21}^2\right) g_t$$
$$+\left(\eta_1 L_1^2+\eta_2 L_{21}^2\right) e_t + \frac{\eta_1^2}{2}\left(L_1+L_\Phi+2\eta_2 L_{21}^2\right)\frac{\sigma_{\boldsymbol{w}}^2}{n} + \frac{\eta_2^2}{2}L_2\sigma_\psi^2, \tag{C.6}$$

where L_Φ is the Lipschitz gradient parameter of the function $\Phi(\cdot)$ characterized in Lemma C.2.

Proof: We defer the proof to Section C.4.5. ■

Next lemma bounds e_t, that is the average deviation of local parameter models from their average.

Lemma C.6 *If Assumptions 4.1, 4.2 and 4.3 hold and the step-size η_1 satisfies $32\eta_1^2(\tau-1)^2L_1^2 \leq 1$, then the sequence $e_t = \frac{1}{n}\sum_{i\in[n]}\mathbb{E}\left\|\mathbf{w}_t^i - \overline{\mathbf{w}}_t\right\|^2$ is bounded as follows*

$$e_t \leq 16\eta_1^2(\tau-1)^2\rho^2 + 4\eta_1^2(\tau-1)(n+1)\frac{\sigma_{\boldsymbol{w}}^2}{n} + 20\eta_1^2(\tau-1)\sum_{l=t_c+1}^{t-1} g_l, \tag{C.7}$$

where t_c denotes the index of the most recent server-worker communication, i.e. $t_c = \left\lfloor\frac{t}{\tau}\right\rfloor\tau$ and we also denote $\rho^2 := 3\rho_f^2 + 6L_{12}^2(\epsilon_1^2+\epsilon_2^2)$.

Proof: We defer the proof to Section C.4.6. ■

Next generic lemma is adopted form [70].

Lemma C.7 *Assume that two non-negative sequences $\{P_t\}_{t\geq 0}$ and $\{g_t\}_{t\geq 0}$ satisfy the following inequality for each iteration $t \geq 0$ and some constants $0 < \Upsilon < 1$, $L \geq 0$,*

$B \geq 0$, and $\Gamma \geq 0$:

$$P_{t+1} \leq \Upsilon P_t - \frac{\eta_1}{2}\left(1-\eta_1 L\right) g_t + \eta_1^2 B \sum_{l=t_c+1}^{t-1} g_l + \Gamma, \tag{C.8}$$

where $t_c = \left\lfloor \frac{t}{\tau} \right\rfloor \tau$. Then, for each $t \geq 0$ we have

$$P_t \leq \Upsilon^t P_0 + \frac{\Gamma}{1-\Upsilon}, \tag{C.9}$$

if η_1 satisfies the following condition

$$\eta_1 \left(L + \frac{2B}{\Upsilon^{\tau-1}(1-\Upsilon)} \right) \leq 1. \tag{C.10}$$

Proof: We defer the proof to Section C.4.7. ■

Next lemma bounds the overall optimality gap b_t averaged over T iterations.

Lemma C.8 *If Assumptions 4.2, 4.3 and 4.4 (ii) hold and the step-sizes satisfy the conditions $\eta_2 \leq 1/L_2$ and $\frac{\eta_2}{\eta_1} \geq \frac{8L_{12}^2}{\mu_2^2 n^2}$, then the average of the sequence $\{b_t\}_{t=0}^{T-1}$ generated from the* `FedRobust` *can be bounded as follows:*

$$\begin{aligned}\frac{1}{T}\sum_{t=0}^{T-1} b_t \leq{}& \frac{4L_2^2}{\mu_2^2 n^2}\frac{\epsilon_1^2+\epsilon_2^2}{\eta_2 T} + \frac{\eta_1}{\eta_2}\frac{1}{\mu_2 n}\frac{1}{T}\sum_{t=0}^{T-1}\mathbb{E}\left\|\nabla\Phi(\overline{\mathbf{w}}_t)\right\|^2 \\ &+ \frac{\eta_1^2}{\eta_2}\frac{1}{\mu_2 n}\left(L_1 + L_\Phi + 2\eta_2 L_{21}^2\right)\frac{1}{T}\sum_{t=0}^{T-1} g_t + \frac{1}{\eta_2}\frac{2}{\mu_2 n}\left(\eta_1 L_1^2 + \eta_2 L_{21}^2\right)\frac{1}{T}\sum_{t=0}^{T-1} e_t \\ &+ \frac{\eta_1^2}{\eta_2}\frac{1}{\mu_2 n}\left(L_1 + L_\Phi + 2\eta_2 L_{21}^2\right)\frac{\sigma_{\boldsymbol{w}}^2}{n} + \eta_2\frac{L_2}{\mu_2 n}\sigma_\psi^2,\end{aligned} \tag{C.11}$$

where L_Φ is the Lipschitz gradient parameter of the function $\Phi(\cdot)$ characterized in Lemma C.2 and ϵ_1, ϵ_2 represent the radius of the affine perturbation balls, i.e. $\|\Lambda^i - I\| \leq \epsilon_1$ and $\|\delta^i\| \leq \epsilon_2$ for each node $i \in [n]$.

Proof: We defer the proof to Section C.4.8. ■

Next lemma bounds the averaged local model deviations e_t over T iterations.

Lemma C.9 *If Assumptions 4.1, 4.2 and 4.3 hold and the step-size η_1 satisfies $32\eta_1^2(\tau-1)^2L_1^2 \leq 1$, then the average of the sequence e_t over $t = 0, \cdots, T-1$ is bounded as follows*

$$\frac{1}{T}\sum_{t=0}^{T-1} e_t \leq 20\eta_1^2(\tau-1)^2\frac{1}{T}\sum_{t=0}^{T-1} g_t + 16\eta_1^2(\tau-1)^2\rho^2 + 8\eta_1^2(\tau-1)(n+1)\frac{\sigma_{\boldsymbol{w}}^2}{n}. \tag{C.12}$$

Proof: We defer the proof to Section C.4.9. ■

C.2 Proof of Theorem 4.1

Having established the key lemmas, now we proceed to prove Theorem 4.1 for any $\beta \leq 1/2$. To show the convergence of the sequence $P_t = a_t + \beta b_t$, we firstly need to establish a contraction inequality on P_{t+1} with respect to P_t. We begin by the following bound on the sequence $a_t = \mathbb{E}[\Phi(\overline{\mathbf{w}}_t)] - \Phi^*$ which is directly implied from Lemma C.3:

$$a_{t+1} \leq a_t - \frac{\eta_1}{2}\mathbb{E}\left\|\nabla\Phi(\overline{\mathbf{w}}_t)\right\|^2 + \frac{\eta_1}{2}h_t - \frac{\eta_1}{2}\left(1-\eta_1 L_\Phi\right)g_t + \eta_1^2\frac{L_\Phi}{2}\frac{\sigma_{\boldsymbol{w}}^2}{n}. \tag{C.13}$$

Using Lemma C.4 that shows $h_t \leq 4L_{12}^2 b_t/(\mu_2 n) + 2L_1^2 e_t$, the bound in (C.13) yields that

$$a_{t+1} \leq a_t - \frac{\eta_1}{2}\mathbb{E}\left\|\nabla\Phi(\overline{\mathbf{w}}_t)\right\|^2 + \eta_1\frac{2L_{12}^2}{\mu_2 n}b_t + \eta_1 L_1^2 e_t - \frac{\eta_1}{2}\left(1-\eta_1 L_\Phi\right)g_t + \eta_1^2\frac{L_\Phi}{2}\frac{\sigma_{\boldsymbol{w}}^2}{n}. \tag{C.14}$$

Next, we employ the result of Lemma C.5 which establishes a contraction bound on the b_t sequence. Putting together with (C.14) implies that

$$\begin{aligned}P_{t+1} &= a_{t+1} + \beta b_{t+1}\\ &\leq a_t - \frac{\eta_1}{2}\left(1-\beta\right)\mathbb{E}\left\|\nabla\Phi(\overline{\mathbf{w}}_t)\right\|^2\end{aligned}$$

$$
\begin{aligned}
&+ \beta \left(\eta_1 \frac{2L_{12}^2}{\beta \mu_2 n} + (1 - \mu_2 \eta_2 n) \left(1 + \eta_1 \frac{4L_{12}^2}{\mu_2 n} \right) \right) b_t \\
&- \left(\frac{\eta_1}{2} (1 - \eta_1 L_\Phi) - \eta_1^2 \frac{\beta}{2} \left(L_1 + L_\Phi + 2\eta_2 L_{21}^2 \right) \right) g_t \\
&+ \left(\eta_1 L_1^2 + \beta \left(\eta_1 L_1^2 + \eta_2 L_{21}^2 \right) \right) e_t \\
&+ \frac{\eta_1^2}{2} \left(L_\Phi + \beta \left(L_1 + L_\Phi + 2\eta_2 L_{21}^2 \right) \right) \frac{\sigma_{\boldsymbol{w}}^2}{n} + \eta_2^2 L_2 \frac{\beta}{2} \sigma_\psi^2. \qquad \text{(C.15)}
\end{aligned}
$$

We begin simplifying the above bound by first considering the first two terms in RHS of (C.15). We can show that the function $\Phi(\cdot)$ is μ_1-PL [98], which implies that

$$
\mathbb{E}\left\| \nabla \Phi(\overline{\mathbf{w}}_t) \right\|^2 \geq 2\mu_1 \mathbb{E}[\Phi(\overline{\mathbf{w}}_t)] - \Phi^* = 2\mu_1 a_t. \qquad \text{(C.16)}
$$

Therefore, for any $\beta \leq 1/2$ we have

$$
a_t - \frac{\eta_1}{2} (1 - \beta) \mathbb{E}\left\| \nabla \Phi(\overline{\mathbf{w}}_t) \right\|^2 \leq \left(1 - \frac{1}{2} \mu_1 \eta_1 \right) a_t, \qquad \text{(C.17)}
$$

which implies the coefficient of a_t in (C.15) is bounded by $1 - \frac{1}{2}\mu_1 \eta_1$. Next, the coefficient of βb_t in (C.15) can be bounded as follows:

$$
\begin{aligned}
&\eta_1 \frac{2L_{12}^2}{\beta \mu_2 n} + (1 - \mu_2 \eta_2 n) \left(1 + \eta_1 \frac{4L_{12}^2}{\mu_2 n} \right) \\
&= 1 - \eta_1 \frac{L_1 L_2}{\mu_2 n} \left(\frac{\mu_2^2 \eta_2 n}{\eta_1 L_1 L_2} - \frac{2L_{21}^2}{\beta L_1 L_2} - 4(1 - \mu_2 \eta_2 n) \frac{L_{21}^2}{L_1 L_2} \right) \\
&\overset{(a)}{\leq} 1 - \eta_1 \frac{L_1 L_2}{\mu_2 n} \\
&\overset{(b)}{\leq} 1 - \frac{1}{2} \mu_1 \eta_1, \qquad \text{(C.18)}
\end{aligned}
$$

where (a) holds for our choice of β and assuming $\frac{\mu_2^2\eta_2 n}{\eta_1 L_1 L_2} \geq 1 + (4+\frac{2}{\beta})\frac{L_{12}^2}{L_1 L_2}$ and (b) is implies from the fact that

$$\frac{\eta_1 \frac{L_1 L_2}{\mu_2 n}}{\frac{1}{2}\mu_1\eta_1} = 2\left(\frac{L_1}{\mu_1}\right)\left(\frac{L_2}{\mu_2 n}\right) \geq 1. \tag{C.19}$$

Now that we have bounded the coefficients of a_t and βb_t in (C.15), rearranging the terms and using the assumption $\eta_2 \leq 1/L_2$ simplifies the contraction on P_t as follows

$$P_{t+1} \leq \left(1 - \frac{1}{2}\mu_1\eta_1\right)P_t - \frac{\eta_1}{2}\left(1 - \eta_1\hat{L}_\beta\right)g_t + \tilde{L}_\beta e_t + \eta_1^2\frac{\hat{L}_\beta}{2}\frac{\sigma_{\boldsymbol{w}}^2}{n} + \eta_2^2\frac{L_2}{2}\beta\sigma_\psi^2, \tag{C.20}$$

where we picked the following notations for convenient of the exposition

$$\tilde{L}_\beta = (1+\beta)\eta_1 L_1^2 + \beta\eta_2 L_{21}^2, \quad \hat{L}_\beta = (1+\beta)L_\Phi + \beta L_1 + 2\beta\frac{L_{21}^2}{L_2}. \tag{C.21}$$

Next, we use Lemma C.6 which for $32\eta_1^2(\tau-1)^2L_1^2 \leq 1$ provides an upper bound on e_t with respect to g_t. We can write

$$\begin{aligned} P_{t+1} \leq& \left(1 - \frac{1}{2}\mu_1\eta_1\right)P_t - \frac{\eta_1}{2}\left(1 - \eta_1\hat{L}_\beta\right)g_t + 20\eta_1^2\tilde{L}_\beta(\tau-1)\sum_{l=t_c+1}^{t-1} g_l \\ &+ 16\eta_1^2\tilde{L}_\beta(\tau-1)^2\rho^2 + 4\eta_1^2\tilde{L}_\beta(\tau-1)(n+1)\frac{\sigma_{\boldsymbol{w}}^2}{n} + \eta_1^2\frac{\hat{L}_\beta}{2}\frac{\sigma_{\boldsymbol{w}}^2}{n} + \eta_2^2\frac{L_2}{2}\beta\sigma_\psi^2. \end{aligned} \tag{C.22}$$

We have shown in Lemma C.7 that how a such contraction sequence converges. In particular, let us pick the following notations and apply the result of Lemma C.7 to contraction in (C.22)

$$L = \hat{L}_\beta,$$

$$\Upsilon = 1 - \frac{1}{2}\mu_1\eta_1,$$

$$B = 20\tilde{L}_\beta(\tau - 1),$$

$$\Gamma = 16\eta_1^2\tilde{L}_\beta(\tau-1)^2\rho^2 + 4\eta_1^2\tilde{L}_\beta(\tau-1)(n+1)\frac{\sigma_{\boldsymbol{w}}^2}{n} + \eta_1^2\frac{\hat{L}_\beta}{2}\frac{\sigma_{\boldsymbol{w}}^2}{n} + \eta_2^2\frac{L_2}{2}\beta\sigma_\psi^2. \tag{C.23}$$

It implies that if the step-sizes satisfy the following condition

$$\eta_1\left(\hat{L}_\beta + \frac{80\tilde{L}_\beta(\tau-1)}{\eta_1\mu_1\left(1-\frac{1}{2}\mu_1\eta_1\right)^{\tau-1}}\right) \leq 1, \tag{C.24}$$

then we have

$$\begin{aligned} P_t \leq \left(1-\frac{1}{2}\mu_1\eta_1\right)^t P_0 &+ 32\eta_1\frac{\tilde{L}_\beta}{\mu_1}(\tau-1)^2\rho^2 \\ &+ 8\eta_1\frac{\tilde{L}_\beta}{\mu_1}(\tau-1)(n+1)\frac{\sigma_{\boldsymbol{w}}^2}{n} + \eta_1\frac{\hat{L}_\beta}{\mu_1}\frac{\sigma_{\boldsymbol{w}}^2}{n} + \frac{\eta_2^2}{\eta_1}\frac{L_2}{\mu_1}\beta\sigma_\psi^2, \end{aligned} \tag{C.25}$$

which concludes the proof of Theorem 4.1. Note to hold this result, in addition to condition (C.24), we have assumed the following constraints on the step-sizes as well

$$\eta_2 L_2 \leq 1, \quad 32\eta_1^2(\tau-1)^2L_1^2 \leq 1, \quad \frac{\mu_2^2\eta_2 n}{\eta_1 L_1 L_2} \geq 1 + \left(4+\frac{2}{\beta}\right)\frac{L_{12}^2}{L_1L_2}. \tag{C.26}$$

C.3 Proof of Theorem 4.2

We begin the proof by combining the results of Lemmas C.3 and C.4 which yields that for every iteration $t = 0, \cdots, T-1$ we have

$$\begin{aligned} \mathbb{E}[\Phi(\overline{\mathbf{w}}_{t+1})] - \mathbb{E}[\Phi(\overline{\mathbf{w}}_t)] \leq -\frac{\eta_1}{2}\mathbb{E}\left\|\nabla\Phi(\overline{\mathbf{w}}_t)\right\|^2 &- \frac{\eta_1}{2}\left(1-\eta_1 L_\Phi\right)g_t \\ &+ \eta_1\frac{2L_{12}^2}{\mu_2 n}b_t + \eta_1 L_1^2 e_t + \eta_1^2\frac{L_\Phi}{2}\frac{\sigma_{\boldsymbol{w}}^2}{n}. \end{aligned} \tag{C.27}$$

Summing up all the T inequalities in (C.27) for $t = 0, \cdots, T-1$ and dividing by T yields the following

$$\frac{1}{T}\left(\mathbb{E}[\Phi(\overline{\mathbf{w}}_T)] - \Phi(\overline{\mathbf{w}}_0)\right) \leq -\frac{\eta_1}{2}\frac{1}{T}\sum_{t=0}^{T-1}\mathbb{E}\left\|\nabla\Phi(\overline{\mathbf{w}}_t)\right\|^2 - \frac{\eta_1}{2}\left(1-\eta_1 L_\Phi\right)\frac{1}{T}\sum_{t=0}^{T-1} g_t$$
$$+ \eta_1\frac{2L_{12}^2}{\mu_2 n}\frac{1}{T}\sum_{t=0}^{T-1} b_t + \eta_1 L_1^2\frac{1}{T}\sum_{t=0}^{T-1} e_t + \eta_1^2\frac{L_\Phi}{2}\frac{\sigma_{\boldsymbol{w}}^2}{n}. \tag{C.28}$$

Next we use Lemmas C.8 and then Lemma C.9 to replace the terms $\frac{1}{T}\sum_{t=0}^{T-1} b_t$ and $\frac{1}{T}\sum_{t=0}^{T-1} e_t$ and rewrite the above bound in terms of $\frac{1}{T}\sum_{t=0}^{T-1} g_t$. It yields that

$$\frac{1}{T}\left(\mathbb{E}[\Phi(\overline{\mathbf{w}}_T)] - \Phi(\overline{\mathbf{w}}_0)\right) \leq -\frac{\eta_1}{2}\left(1 - \eta_1\frac{4L_{12}^2 L_2}{\mu_2^2 n^2}\right)\frac{1}{T}\sum_{t=0}^{T-1}\mathbb{E}\left\|\nabla\Phi(\overline{\mathbf{w}}_t)\right\|^2$$
$$- \frac{\eta_1}{2}\left(1 - \eta_1\left(\hat{L} + 40\tilde{L}(\tau-1)^2\right)\right)\frac{1}{T}\sum_{t=0}^{T-1} g_t$$
$$+ \frac{\eta_1}{\eta_2}\frac{8L_{12}^2 L_2^2}{\mu_2^3 n^3}\frac{\epsilon_1^2+\epsilon_2^2}{T} + 16\eta_1^2\tilde{L}(\tau-1)^2\rho^2$$
$$+ \frac{\eta_1^2}{2}\hat{L}\frac{\sigma_{\boldsymbol{w}}^2}{n} + \eta_1\eta_2\frac{4L_{12}^2}{\mu_2^2 n^2}\hat{L}\sigma_\psi^2, \tag{C.29}$$

where we adopt the following short-hand notations

$$\tilde{L} = \frac{3}{2}\eta_1 L_1^2 + \frac{1}{2}\eta_2 L_{21}^2, \quad \hat{L} = \frac{3}{2}L_\Phi + \frac{1}{2}L_1 + \frac{L_{21}^2}{L_2}. \tag{C.30}$$

Finally, we use the assumption $\eta_1(\hat{L} + 40\tilde{L}(\tau-1)^2) \leq 1$ to remove the term $\frac{1}{T}\sum_{t=0}^{T-1} g_t$ and apply $\frac{\eta_1}{\eta_2} \leq \frac{\mu_2^2 n^2}{8L_{12}^2}$ to simply the bound and conclude the proof:

$$\frac{1}{T}\sum_{t=0}^{T-1}\mathbb{E}\left\|\nabla\Phi(\overline{\mathbf{w}}_t)\right\|^2 \leq \frac{4\Delta_\Phi}{\eta_1 T} + \frac{4L_2^2}{\mu_2^2 n^2}\frac{\epsilon_1^2+\epsilon_2^2}{\eta_1 T} + 64\eta_1\tilde{L}(\tau-1)^2\rho^2$$
$$+ 16\eta_1\tilde{L}(\tau-1)(n+1)\frac{\sigma_{\boldsymbol{w}}^2}{n} + 2\eta_1\hat{L}\frac{\sigma_{\boldsymbol{w}}^2}{n} + \frac{\eta_2^2}{\eta_1}L_2\sigma_\psi^2. \tag{C.31}$$

C.4 Proof of Useful Lemmas

C.4.1 Proof of Lemma C.1

Proof of all four cases in the claim is simple. We derive the proof for the fourth one as an instance. Recall definition of the global function f, that is

$$f(\mathbf{w}, \Psi) = \frac{1}{n} \sum_{i \in [n]} f^i(\mathbf{w}, \boldsymbol{\psi}^i). \tag{C.32}$$

Therefore, the gradient of f with respect to Ψ is

$$\nabla_\Psi f(\mathbf{w}, \Psi) = \begin{pmatrix} \frac{\partial}{\partial \boldsymbol{\psi}^1} f(\mathbf{w}, \Psi) \\ \vdots \\ \frac{\partial}{\partial \boldsymbol{\psi}^n} f(\mathbf{w}, \Psi) \end{pmatrix} = \frac{1}{n} \begin{pmatrix} \nabla_{\boldsymbol{\psi}} f^1(\mathbf{w}, \boldsymbol{\psi}^1) \\ \vdots \\ \nabla_{\boldsymbol{\psi}} f^n(\mathbf{w}, \boldsymbol{\psi}^n) \end{pmatrix}. \tag{C.33}$$

We can then write for any $\mathbf{w}, \Psi = (\boldsymbol{\psi}^1; \cdots; \boldsymbol{\psi}^n), \Psi' = (\boldsymbol{\psi}'^1; \cdots; \boldsymbol{\psi}'^n)$ and using Assumption 4.3 that

$$\begin{aligned} \left\|\nabla_\Psi f(\mathbf{w}, \Psi) - \nabla_\Psi f(\mathbf{w}, \Psi')\right\|_F^2 &= \frac{1}{n^2} \sum_{i \in [n]} \left\|\nabla_{\boldsymbol{\psi}} f^i(\mathbf{w}, \boldsymbol{\psi}^i) - \nabla_{\boldsymbol{\psi}} f^i(\mathbf{w}, \boldsymbol{\psi}'^i)\right\|_F^2 \\ &\leq \frac{L_2^2}{n^2} \sum_{i \in [n]} \left\|\boldsymbol{\psi}^i - \boldsymbol{\psi}'^i\right\|_F^2 \\ &= \frac{L_2^2}{n^2} \left\|\Psi - \Psi'\right\|_F^2. \end{aligned} \tag{C.34}$$

C.4.2 Proof of Lemma C.2

The detailed proof can be found in [99], Lemma A.5. Note that in our case, according to Lemma C.1 the function f has Lipschitz gradients with constants $L_1, L_{12}/\sqrt{n}, L_{21}/\sqrt{n}$,

L_2/n; implying the Lipschitz gradient parameter of the function Φ to be

$$L_\Phi = L_1 + \frac{(L_{12}/\sqrt{n})(L_{21}/\sqrt{n})}{2\mu_2} = L_1 + \frac{L_{12}L_{21}}{2n\mu_2}. \tag{C.35}$$

C.4.3 Proof of Lemma C.3

We invoke Lemma C.2 which shows that the gradient of the function $\Phi(\cdot)$ is L_Φ-Lipschitz. We can write

$$\begin{aligned}\Phi(\overline{\mathbf{w}}_{t+1}) - \Phi(\overline{\mathbf{w}}_t) &\leq \left\langle \nabla\Phi(\overline{\mathbf{w}}_t), \overline{\mathbf{w}}_{t+1} - \overline{\mathbf{w}}_t \right\rangle + \frac{L_\Phi}{2}\|\overline{\mathbf{w}}_{t+1} - \overline{\mathbf{w}}_t\|^2 \\ &= -\eta_1 \left\langle \nabla\Phi(\overline{\mathbf{w}}_t), \frac{1}{n}\sum_{i\in[n]} \widetilde{\nabla}_{\mathbf{w}} f^i(w_t^i, \boldsymbol{\psi}_t^i) \right\rangle + \eta_1^2 \frac{L_\Phi}{2}\left\| \frac{1}{n}\sum_{i\in[n]} \widetilde{\nabla}_{\mathbf{w}} f^i(w_t^i, \boldsymbol{\psi}_t^i) \right\|^2,\end{aligned} \tag{C.36}$$

where we use the update rule of `FedRobust` and note that the difference of averaged models can be written as $\overline{\mathbf{w}}_{t+1} - \overline{\mathbf{w}}_t = -\eta_1 \frac{1}{n}\sum_{i\in[n]} \widetilde{\nabla}_{\mathbf{w}} f^i(w_t^i, \boldsymbol{\psi}_t^i)$. Moreover, since the stochastic gradients $\widetilde{\nabla}_{\mathbf{w}} f^i$ are unbiased and variance-bounded by $\sigma_{\boldsymbol{w}}^2$, we can take expectation from both sides of (C.36) and further simplify it as follows

$$\mathbb{E}[\Phi(\overline{\mathbf{w}}_{t+1}) - \mathbb{E}[\Phi(\overline{\mathbf{w}}_t)] \leq -\frac{\eta_1}{2}\mathbb{E}\left\|\nabla\Phi(\overline{\mathbf{w}}_t)\right\|^2 + \frac{\eta_1}{2}h_t - \frac{\eta_1}{2}\left(1 - \eta_1 L_\Phi\right) g_t + \eta_1^2 \frac{L_\Phi}{2}\frac{\sigma_{\boldsymbol{w}}^2}{n}. \tag{C.37}$$

In above, we used the inequality $2\langle \mathbf{a}, \mathbf{b}\rangle = \|\mathbf{a}\|^2 + \|\mathbf{b}\|^2 - \|\mathbf{a} - \mathbf{b}\|^2$ as well as the notations for g_t and h_t as defined in Table C.1.

C.4.4 Proof of Lemma C.4

We begin bounding h_t by adding/subtracting the term $\nabla_{\boldsymbol{w}} f(\overline{\mathbf{w}}_t, \Psi_t)$ and use the inequality $\|\mathbf{a}+\mathbf{b}\|^2 \leq 2\|\mathbf{a}\|^2 + 2\|\mathbf{b}\|^2$ to write

$$\begin{aligned} h_t &= \mathbb{E}\left\| \nabla \Phi(\overline{\mathbf{w}}_t) - \frac{1}{n}\sum_{i\in[n]} \nabla_{\boldsymbol{w}} f^i(\mathbf{w}_t^i, \boldsymbol{\psi}_t^i) \right\|^2 \\ &\leq 2\mathbb{E}\left\| \nabla \Phi(\overline{\mathbf{w}}_t) - \nabla_{\boldsymbol{w}} f(\overline{\mathbf{w}}_t, \Psi_t) \right\|^2 + 2\mathbb{E}\left\| \nabla_{\boldsymbol{w}} f(\overline{\mathbf{w}}_t, \Psi_t) - \frac{1}{n}\sum_{i\in[n]} \nabla_{\boldsymbol{w}} f^i(\mathbf{w}_t^i, \boldsymbol{\psi}_t^i) \right\|^2. \end{aligned} \tag{C.38}$$

The first term in RHS of (C.38) can be bounded as follows:

$$\begin{aligned} \mathbb{E}\left\| \nabla \Phi(\overline{\mathbf{w}}_t) - \nabla_{\boldsymbol{w}} f(\overline{\mathbf{w}}_t, \Psi_t) \right\|^2 &= \mathbb{E}\left\| \nabla_{\boldsymbol{w}} f(\overline{\mathbf{w}}_t, \Psi^*(\overline{\mathbf{w}}_t)) - \nabla_{\boldsymbol{w}} f(\overline{\mathbf{w}}_t, \Psi_t) \right\|^2 \\ &\overset{(a)}{\leq} \frac{L_{12}^2}{n}\mathbb{E}\left\| \Psi^*(\overline{\mathbf{w}}_t) - \Psi_t \right\|_F^2 \\ &\overset{(b)}{\leq} \frac{2L_{12}^2}{\mu_2 n}\mathbb{E}\left[\Phi(\overline{\mathbf{w}}_t) - f(\overline{\mathbf{w}}_t, \Psi_t) \right] \\ &\overset{(c)}{=} \frac{2L_{12}^2}{\mu_2 n} b_t. \end{aligned} \tag{C.39}$$

In above and to derive (a), we employ the result of Lemma C.1 which shows that given Assumption 4.3, the gradient function $\nabla_{\boldsymbol{w}} f(\mathbf{w}, \cdot)$ is $L_{12}/\sqrt{n}$ Lipschitz. To derive (b), we use Assumption 4.4 (ii) and lastly, (c) is implied from the definition of b_t. The second term in RHS of (C.38) can be bounded by noting that the local gradients $\nabla_{\boldsymbol{w}} f^i(\cdot, \boldsymbol{\psi}^i)$ are L_1-Lipschitz, which we can write

$$\mathbb{E}\left\| \nabla_{\boldsymbol{w}} f(\overline{\mathbf{w}}_t, \Psi_t) - \frac{1}{n}\sum_{i\in[n]} \nabla_{\boldsymbol{w}} f^i(\mathbf{w}_t^i, \boldsymbol{\psi}_t^i) \right\|^2$$

$$
\begin{aligned}
&= \mathbb{E}\left\| \frac{1}{n}\sum_{i\in[n]} \nabla_{\boldsymbol{w}} f^i(\overline{\mathbf{w}}_t, \boldsymbol{\psi}_t^i) - \frac{1}{n}\sum_{i\in[n]} \nabla_{\boldsymbol{w}} f^i(\mathbf{w}_t^i, \boldsymbol{\psi}_t^i) \right\|^2 \\
&\le \frac{L_1^2}{n}\sum_{i\in[n]} \mathbb{E}\left\| \mathbf{w}_t^i - \overline{\mathbf{w}}_t \right\|^2 \\
&= L_1^2 e_t. \qquad \text{(C.40)}
\end{aligned}
$$

Finally, plugging (C.39) and (C.40) back in (C.38) implies the claim of the lemma, that is

$$
h_t \le \frac{4L_{12}^2}{\mu_2 n} b_t + 2L_1^2 e_t. \qquad \text{(C.41)}
$$

C.4.5 Proof of Lemma C.5

We begin the proof by noting the definition of b_t and use the fact that the gradients $\nabla_\Psi f(\mathbf{w}, \cdot)$ are $\frac{L_2}{n}$-Lipschitz (Refer to Lemma C.1). We can accordingly write

$$
\begin{aligned}
\Phi(\overline{\mathbf{w}}_{t+1}) - f(\overline{\mathbf{w}}_{t+1}, \Psi_{t+1}) \le \Phi(\overline{\mathbf{w}}_{t+1}) - f(\overline{\mathbf{w}}_{t+1}, \Psi_t) - \langle \nabla_\Psi f(\overline{\mathbf{w}}_{t+1}, \Psi_t), \Psi_{t+1} - \Psi_t \rangle \\
+ \frac{L_2}{2n} \|\Psi_{t+1} - \Psi_t\|_F^2 . \qquad \text{(C.42)}
\end{aligned}
$$

In this work, we define the inner product for any two matrices A, B as follows

$$
\langle A, B \rangle \coloneqq \mathrm{Tr}(A^\top B). \qquad \text{(C.43)}
$$

Note that according to the ascent update rule of `FedRobust` in Algorithm 4.1, we can write

$$
\Psi_{t+1} - \Psi_t = \eta_2 \tilde{\partial}_t f, \qquad \text{(C.44)}
$$

where we adopt the following short-hand notation for the stochastic gradients at iteration t with respect to the maximization variables $\boldsymbol{\psi}_t^i = (\Lambda_t^i, \delta_t^i)$

$$\tilde{\partial}_t f = \begin{pmatrix} \widetilde{\nabla}_\psi f^1(\mathbf{w}_t^1, \boldsymbol{\psi}_t^1) \\ \vdots \\ \widetilde{\nabla}_\psi f^n(\mathbf{w}_t^n, \boldsymbol{\psi}_t^n) \end{pmatrix} = \begin{pmatrix} \widetilde{\nabla}_\Lambda f^1(\mathbf{w}_t^1, \Lambda_t^1, \delta_t^1) & \widetilde{\nabla}_\delta f^1(\mathbf{w}_t^1, \Lambda_t^1, \delta_t^1) \\ \vdots & \vdots \\ \widetilde{\nabla}_\Lambda f^n(\mathbf{w}_t^n, \Lambda_t^n, \delta_t^n) & \widetilde{\nabla}_\delta f^n(\mathbf{w}_t^n, \Lambda_t^n, \delta_t^n) \end{pmatrix}. \tag{C.45}$$

We also denote the gradients by $\partial_t f = \mathbb{E}[\tilde{\partial}_t f]$ where the expectation is with respect to the randomness in stochastic gradients $\widetilde{\nabla}_\psi f^i$. According to Assumption 4.2, each of the local stochastic gradients $\widetilde{\nabla}_\psi f^i(\mathbf{w}_t^i, \boldsymbol{\psi}_t^i)$ are variance-bounded by σ_ψ^2. Therefore, we can bound the variance of $\tilde{\partial}_t f$ as $\mathbb{E}\|\tilde{\partial}_t f - \partial_t f\|_F^2 \leq n\sigma_\psi^2$. Now, we can plug these back in (C.42) which implies

$$\begin{aligned}\Phi(\overline{\mathbf{w}}_{t+1}) - \mathbb{E} f(\overline{\mathbf{w}}_{t+1}, \Psi_{t+1}) \leq \Phi(\overline{\mathbf{w}}_{t+1}) - f(\overline{\mathbf{w}}_{t+1}, \Psi_t) - \eta_2 \frac{n}{2} \left\| \nabla_\Psi f(\overline{\mathbf{w}}_{t+1}, \Psi_t) \right\|_F^2 + \eta_2^2 \frac{L_2}{2} \sigma_\psi^2 \\ + \eta_2 \frac{n}{2} \left\| \nabla_\Psi f(\overline{\mathbf{w}}_{t+1}, \Psi_t) - \frac{1}{n} \partial_t f \right\|_F^2 - \frac{\eta_2}{2n} (1 - \eta_2 L_2) \|\partial_t f\|_F^2,\end{aligned} \tag{C.46}$$

where the expectation is with respect to the randomness of the stochastic gradients $\tilde{\partial}_t f$ while conditioning on all the randomness history. Now recall from Assumption 4.4 (ii) that $-f(\overline{\mathbf{w}}_{t+1}, \cdot)$ is μ_2-PL implying that $\|\nabla_\Psi f(\overline{\mathbf{w}}_{t+1}, \Psi_t)\|_F^2 \geq 2\mu_2(\Phi(\overline{\mathbf{w}}_{t+1}) - f(\overline{\mathbf{w}}_{t+1}, \Psi_t))$. Moreover, assume that $\eta_2 \leq 1/L_2$ to remove the last term in (C.46). Putting altogether implies that

$$\begin{aligned}\Phi(\overline{\mathbf{w}}_{t+1}) - \mathbb{E} f(\overline{\mathbf{w}}_{t+1}, \Psi_{t+1}) \leq (1 - \mu_2 \eta_2 n) \left(\Phi(\overline{\mathbf{w}}_{t+1}) - f(\overline{\mathbf{w}}_{t+1}, \Psi_t) \right) + \eta_2^2 \frac{L_2}{2} \sigma_\psi^2 \\ + \eta_2 \frac{n}{2} \left\| \nabla_\Psi f(\overline{\mathbf{w}}_{t+1}, \Psi_t) - \frac{1}{n} \partial_t f \right\|_F^2.\end{aligned} \tag{C.47}$$

Next, we continue to bound the last term in RHS of (C.47). We can write

$$
\begin{aligned}
\left\|\nabla_{\Psi} f(\overline{\mathbf{w}}_{t+1}, \Psi_t) - \frac{1}{n}\partial_t f\right\|_F^2 &= \frac{1}{n^2}\sum_{i\in[n]}\left\|\nabla_{\boldsymbol{\psi}} f^i(\overline{\mathbf{w}}_{t+1}, \boldsymbol{\psi}_t^i) - \nabla_{\boldsymbol{\psi}} f^i(\mathbf{w}_t^i, \boldsymbol{\psi}_t^i)\right\|_F^2 \\
&\leq \frac{L_{21}^2}{n^2}\sum_{i\in[n]}\left\|\overline{\mathbf{w}}_{t+1} - \mathbf{w}_t^i\right\|^2 \\
&\leq \frac{2L_{21}^2}{n^2}\sum_{i\in[n]}\left\|\mathbf{w}_t^i - \overline{\mathbf{w}}_t\right\|^2 + \frac{2L_{21}^2}{n}\left\|\overline{\mathbf{w}}_{t+1} - \overline{\mathbf{w}}_t\right\|^2, \qquad \text{(C.48)}
\end{aligned}
$$

where the first inequality above uses Assumption 4.3 on Lipschitz continuity of local gradients and the second inequality simply uses the inequality $\|\mathbf{a}+\mathbf{b}\|^2 \leq 2\|\mathbf{a}\|^2 + 2\|\mathbf{b}\|^2$. Next, let us bound the term $\|\overline{\mathbf{w}}_{t+1} - \overline{\mathbf{w}}_t\|^2$ in expectation as follows. Using the descent update rule in Algorithm 4.1 and considering Assumption 4.2 on variance of the stochastic gradients $\widetilde{\nabla}_{\mathbf{w}} f^i$ we can write

$$
\begin{aligned}
\mathbb{E}\left\|\overline{\mathbf{w}}_{t+1} - \overline{\mathbf{w}}_t\right\|^2 &= \eta_1^2 \mathbb{E}\left\|\frac{1}{n}\sum_{i\in[n]}\widetilde{\nabla}_{\mathbf{w}} f^i(\mathbf{w}_t^i, \boldsymbol{\psi}_t^i)\right\|^2 \\
&\leq \eta_1^2 \mathbb{E}\left\|\frac{1}{n}\sum_{i\in[n]}\nabla_{\boldsymbol{w}} f^i(\mathbf{w}_t^i, \boldsymbol{\psi}_t^i)\right\|^2 + \eta_1^2\frac{\sigma_{\boldsymbol{w}}^2}{n} \\
&= \eta_1^2 g_t + \eta_1^2\frac{\sigma_{\boldsymbol{w}}^2}{n}, \qquad \text{(C.49)}
\end{aligned}
$$

where we use the short-hand notation of g_t also listed in Table C.1. Plugging (C.49) back in (C.48) and noting the notation $e_t = \frac{1}{n}\sum_{i\in[n]}\mathbb{E}\left\|\mathbf{w}_t^i - \overline{\mathbf{w}}_t\right\|^2$ implies that

$$
\mathbb{E}\left\|\nabla_{\Psi} f(\overline{\mathbf{w}}_{t+1}, \Psi_t) - \frac{1}{n}\partial_t f\right\|_F^2 \leq \frac{2L_{21}^2}{n} e_t + \eta_1^2\frac{2L_{21}^2}{n} g_t + \eta_1^2\frac{2L_{21}^2}{n}\frac{\sigma_{\boldsymbol{w}}^2}{n}. \qquad \text{(C.50)}
$$

Before proceeding to bound more terms, let us recall what we have shown till this point. We plug (C.50) back in (C.47), take the expectation with respect to all the sources of randomness and use the notation $b_t = \mathbb{E}[\Phi(\overline{\mathbf{w}}_t) - f(\overline{\mathbf{w}}_t, \Psi_t)]$ to conclude

$$\begin{aligned} b_{t+1} \leq (1 - \mu_2\eta_2 n)\mathbb{E}\left[\Phi(\overline{\mathbf{w}}_{t+1}) - f(\overline{\mathbf{w}}_{t+1}, \Psi_t)\right] \\ + \eta_2 L_{21}^2 e_t + \eta_1^2\eta_2 L_{21}^2 g_t + \eta_1^2\eta_2 L_{21}^2 \frac{\sigma_{\boldsymbol{w}}^2}{n} + \eta_2^2 \frac{L_2}{2}\sigma_{\psi}^2. \end{aligned} \tag{C.51}$$

To bound the term $\mathbb{E}\left[\Phi(\overline{\mathbf{w}}_{t+1}) - f(\overline{\mathbf{w}}_{t+1}, \Psi_t)\right]$, we can decompose it to the following three terms:

$$\Phi(\overline{\mathbf{w}}_{t+1}) - f(\overline{\mathbf{w}}_{t+1}, \Psi_t) = \Phi(\overline{\mathbf{w}}_t) - f(\overline{\mathbf{w}}_t, \Psi_t) + f(\overline{\mathbf{w}}_t, \Psi_t) - f(\overline{\mathbf{w}}_{t+1}, \Psi_t) + \Phi(\overline{\mathbf{w}}_{t+1}) - \Phi(\overline{\mathbf{w}}_t). \tag{C.52}$$

Given the Lipschitz gradient assumption for the local functions in Assumption 4.3 and using Lemma C.1 on Lipschitz gradient for the global function, we can write

$$f(\overline{\mathbf{w}}_t, \Psi_t) - f(\overline{\mathbf{w}}_{t+1}, \Psi_t) \leq -\langle \nabla_{\boldsymbol{w}} f(\overline{\mathbf{w}}_t, \Psi_t), \overline{\mathbf{w}}_{t+1} - \overline{\mathbf{w}}_t\rangle + \frac{L_1}{2}\left\|\overline{\mathbf{w}}_{t+1} - \overline{\mathbf{w}}_t\right\|^2, \tag{C.53}$$

where $\overline{\mathbf{w}}_{t+1} - \overline{\mathbf{w}}_t = -\eta_1 \frac{1}{n}\sum_{i\in[n]} \widetilde{\nabla}_{\mathbf{w}} f^i(\mathbf{w}_t^i, \boldsymbol{\psi}_t^i)$. Taking expectation from both sides of (C.53) implies that

$$\begin{aligned} \mathbb{E}\left[f(\overline{\mathbf{w}}_t, \Psi_t) - f(\overline{\mathbf{w}}_{t+1}, \Psi_t)\right] &\overset{(a)}{\leq} \eta_1 \mathbb{E}\left\|\nabla_{\boldsymbol{w}} f(\overline{\mathbf{w}}_t, \Psi_t) - \nabla\Phi(\overline{\mathbf{w}}_t)\right\|^2 + \eta_1\mathbb{E}\left\|\nabla\Phi(\overline{\mathbf{w}}_t)\right\|^2 \\ &\quad + \left(\frac{\eta_1}{2} + \eta_1^2\frac{L_1}{2}\right) g_t + \eta_1^2 \frac{L_1}{2}\frac{\sigma_{\boldsymbol{w}}^2}{n} \\ &\overset{(b)}{\leq} \eta_1 \frac{2L_{12}^2}{\mu_2 n} b_t + \eta_1 \mathbb{E}\left\|\nabla\Phi(\overline{\mathbf{w}}_t)\right\|^2 + \left(\frac{\eta_1}{2} + \eta_1^2\frac{L_1}{2}\right) g_t + \eta_1^2\frac{L_1}{2}\frac{\sigma_{\boldsymbol{w}}^2}{n}, \end{aligned} \tag{C.54}$$

where in inequality (a) we use the inequality $2\langle \mathbf{a}, \mathbf{b}\rangle \leq \|\mathbf{a}\|^2 + \|\mathbf{b}\|^2$ and also the result in (C.49). To derive (b), we use Assumptions 4.3 and 4.4 (ii), result of Lemma C.1 and the notation $b_t = \mathbb{E}[\Phi(\overline{\mathbf{w}}_t) - f(\overline{\mathbf{w}}_t, \Psi_t)]$ to write

$$\begin{aligned}
\mathbb{E}\left\|\nabla\Phi(\overline{\mathbf{w}}_t) - \nabla_{\boldsymbol{w}} f(\overline{\mathbf{w}}_t, \Psi_t)\right\|^2 &= \mathbb{E}\left\|\nabla_{\boldsymbol{w}} f(\overline{\mathbf{w}}_t, \Psi^*(\overline{\mathbf{w}}_t)) - \nabla_{\boldsymbol{w}} f(\overline{\mathbf{w}}_t, \Psi_t)\right\|^2 \\
&\leq \frac{L_{12}^2}{n}\mathbb{E}\left\|\Psi^*(\overline{\mathbf{w}}_t) - \Psi_t\right\|_F^2 \\
&\leq \frac{2L_{12}^2}{\mu_2 n}\mathbb{E}\left[\Phi(\overline{\mathbf{w}}_t) - f(\overline{\mathbf{w}}_t, \Psi_t)\right] \\
&= \frac{2L_{12}^2}{\mu_2 n} b_t. \qquad \text{(C.55)}
\end{aligned}$$

We now have all the ingredients to conclude the claim of Lemma C.5. To do so, we combine the result of Lemma C.3 which bounds the term $\mathbb{E}[\Phi(\overline{\mathbf{w}}_{t+1})] - \mathbb{E}[\Phi(\overline{\mathbf{w}}_t)]$, Lemma C.4 that shows $h_t \leq 4L_{12}^2 b_t/(\mu_2 n) + 2L_1^2 e_t$, and the bound (C.54); plug back in (C.52) and then in (C.51) and conclude the claim of the lemma, that is

$$\begin{aligned}
b_{t+1} \leq (1 - \mu_2\eta_2 n)\left(1 + \eta_1\frac{4L_{12}^2}{\mu_2 n}\right) b_t + \frac{\eta_1}{2}\mathbb{E}\left\|\nabla\Phi(\overline{\mathbf{w}}_t)\right\|^2 + \frac{\eta_1^2}{2}\left(L_1 + L_\Phi + 2\eta_2 L_{21}^2\right) g_t \\
+ \left(\eta_1 L_1^2 + \eta_2 L_{21}^2\right) e_t + \frac{\eta_1^2}{2}\left(L_1 + L_\Phi + 2\eta_2 L_{21}^2\right)\frac{\sigma_{\boldsymbol{w}}^2}{n} + \frac{\eta_2^2}{2} L_2 \sigma_\psi^2, \qquad \text{(C.56)}
\end{aligned}$$

C.4.6 Proof of Lemma C.6

To prove this lemma, we first need to establish an intermediate step, which is stated in the following.

Proposition C.1 *If Assumptions 4.1, 4.2 and 4.3 hold, then*

$$e_t \leq 16\eta_1^2(\tau - 1)L_1^2 \sum_{l=t_c+1}^{t-1} e_l + 10\eta_1^2(\tau - 1)\sum_{l=t_c+1}^{t-1} g_l + 8\eta_1^2(\tau - 1)^2\rho^2 + 4\eta_1^2(\tau - 1)(n+1)\frac{\sigma_{\boldsymbol{w}}^2}{n}. \qquad \text{(C.57)}$$

Proof: [Proof of Proposition C.1] Consider an iteration $t \geq 1$ and let t_c denote the index of the most recent communication between the workers and the server, i.e. $t_c = \left\lfloor \frac{t}{\tau} \right\rfloor \tau$. Therefore, all the workers share the same local minimization model at iteration $t_c + 1$, i.e. $\mathbf{w}^1_{t_c+1} = \cdots = \mathbf{w}^n_{t_c+1} = \overline{\mathbf{w}}_{t_c+1}$. According to the update rule of `FedRobust`, we can write for each node i that

$$\begin{aligned}\mathbf{w}^i_{t_c+2} &= \mathbf{w}^i_{t_c+1} - \eta_1 \widetilde{\nabla}_{\mathbf{w}} f^i(\mathbf{w}^i_{t_c+1}, \boldsymbol{\psi}^i_{t_c+1}), \\ &\vdots \\ \mathbf{w}^i_t &= \mathbf{w}^i_{t-1} - \eta_1 \widetilde{\nabla}_{\mathbf{w}} f^i(\mathbf{w}^i_{t-1}, \boldsymbol{\psi}^i_{t-1}).\end{aligned} \tag{C.58}$$

Summing up all the equalities in (C.58) yields that

$$\mathbf{w}^i_t = \mathbf{w}^i_{t_c+1} - \eta_1 \sum_{l=t_c+1}^{t-1} \widetilde{\nabla}_{\mathbf{w}} f^i(\mathbf{w}^i_l, \boldsymbol{\psi}^i_l). \tag{C.59}$$

Therefore, the difference of the local models $\mathbf{w}^i_t$ and their average $\overline{\mathbf{w}}_t$ can be written as

$$\begin{aligned}\mathbf{w}^i_t - \overline{\mathbf{w}}_t &= \mathbf{w}^i_{t_c+1} - \eta_1 \sum_{l=t_c+1}^{t-1} \widetilde{\nabla}_{\mathbf{w}} f^i(\mathbf{w}^i_l, \boldsymbol{\psi}^i_l) - \left(\overline{\mathbf{w}}_{t_c+1} - \eta_1 \frac{1}{n} \sum_{j\in[n]} \sum_{l=t_c+1}^{t-1} \widetilde{\nabla}_{\mathbf{w}} f^j(\mathbf{w}^j_l, \boldsymbol{\psi}^j_l) \right) \\ &= -\eta_1 \left(\sum_{l=t_c+1}^{t-1} \widetilde{\nabla}_{\mathbf{w}} f^i(\mathbf{w}^i_l, \boldsymbol{\psi}^i_l) - \frac{1}{n} \sum_{j\in[n]} \sum_{l=t_c+1}^{t-1} \widetilde{\nabla}_{\mathbf{w}} f^j(\mathbf{w}^j_l, \boldsymbol{\psi}^j_l) \right).\end{aligned} \tag{C.60}$$

This yields the following bound on each local deviation from the average $\mathbb{E}\|\mathbf{w}^i_t - \overline{\mathbf{w}}_t\|^2$:

$$\begin{aligned}\mathbb{E}\left\|\mathbf{w}^i_t - \overline{\mathbf{w}}_t\right\|^2 &= \eta_1^2 \mathbb{E}\left\| \sum_{l=t_c+1}^{t-1} \widetilde{\nabla}_{\mathbf{w}} f^i(\mathbf{w}^i_l, \boldsymbol{\psi}^i_l) - \frac{1}{n} \sum_{j\in[n]} \sum_{l=t_c+1}^{t-1} \widetilde{\nabla}_{\mathbf{w}} f^j(\mathbf{w}^j_l, \boldsymbol{\psi}^j_l) \right\|^2 \\ &\leq 2\eta_1^2 \mathbb{E}\left\| \sum_{l=t_c+1}^{t-1} \widetilde{\nabla}_{\mathbf{w}} f^i(\mathbf{w}^i_l, \boldsymbol{\psi}^i_l) \right\|^2 + 2\eta_1^2 \mathbb{E}\left\| \frac{1}{n} \sum_{j\in[n]} \sum_{l=t_c+1}^{t-1} \widetilde{\nabla}_{\mathbf{w}} f^j(\mathbf{w}^j_l, \boldsymbol{\psi}^j_l) \right\|^2\end{aligned}$$

$$\overset{(a)}{\leq} 2\eta_1^2\,\underbrace{\mathbb{E}\left\|\sum_{l=t_c+1}^{t-1}\nabla_{\boldsymbol{w}} f^i(\mathbf{w}_l^i,\boldsymbol{\psi}_l^i)\right\|^2}_{T_3} + 2\eta_1^2\,\underbrace{\mathbb{E}\left\|\frac{1}{n}\sum_{j\in[n]}\sum_{l=t_c+1}^{t-1}\nabla_{\boldsymbol{w}} f^j(\mathbf{w}_l^j,\boldsymbol{\psi}_l^j)\right\|^2}_{T_4}$$
$$+ 2\eta_1^2(t-t_c-1)(n+1)\frac{\sigma_{\boldsymbol{w}}^2}{n}, \tag{C.61}$$

where we used Assumption 4.2 to bound the variance of the stochastic gradients and derive (a). The term T_4 in (C.61) can simply be bounded as

$$T_4 \leq \mathbb{E}\left\|\frac{1}{n}\sum_{j\in[n]}\sum_{l=t_c+1}^{t-1}\nabla_{\boldsymbol{w}} f^j(\mathbf{w}_l^j,\boldsymbol{\psi}_l^j)\right\|^2 \leq (t-t_c-1)\sum_{l=t_c+1}^{t-1}\mathbb{E}\left\|\frac{1}{n}\sum_{j\in[n]}\nabla_{\boldsymbol{w}} f^j(\mathbf{w}_l^j,\boldsymbol{\psi}_l^j)\right\|^2 \tag{C.62}$$

Note that t_c denotes the latest server-worker communication before iteration t, hence $t-t_c\leq\tau$ where τ is the duration of local updates in each round. Therefore, we have

$$T_4 \leq (\tau-1)\sum_{l=t_c+1}^{t-1}\mathbb{E}\left\|\frac{1}{n}\sum_{j\in[n]}\nabla_{\boldsymbol{w}} f^j(\mathbf{w}_l^j,\boldsymbol{\psi}_l^j)\right\|^2 \leq (\tau-1)\sum_{l=t_c+1}^{t-1} g_l \tag{C.63}$$

Now we proceed to bound the term T_3 in (C.61) as follows:

$$\begin{aligned}
T_3 &= \mathbb{E}\left\|\sum_{l=t_c+1}^{t-1}\nabla_{\boldsymbol{w}} f^i(\mathbf{w}_l^i,\boldsymbol{\psi}_l^i)\right\|^2 \\
&\leq (\tau-1)\sum_{l=t_c+1}^{t-1}\mathbb{E}\left\|\nabla_{\boldsymbol{w}} f^i(\mathbf{w}_l^i,\boldsymbol{\psi}_l^i)\right\|^2 \\
&\leq 4(\tau-1)\sum_{l=t_c+1}^{t-1}\mathbb{E}\left\|\nabla_{\boldsymbol{w}} f^i(\mathbf{w}_l^i,\boldsymbol{\psi}_l^i)-\nabla_{\boldsymbol{w}} f^i(\overline{\mathbf{w}}_l,\boldsymbol{\psi}_l^i)\right\|^2 \\
&\quad + 4(\tau-1)\sum_{l=t_c+1}^{t-1}\mathbb{E}\left\|\nabla_{\boldsymbol{w}} f^i(\overline{\mathbf{w}}_l,\boldsymbol{\psi}_l^i)-\frac{1}{n}\sum_{j\in[n]}\nabla_{\boldsymbol{w}} f^j(\overline{\mathbf{w}}_l,\boldsymbol{\psi}_l^j)\right\|^2
\end{aligned}$$

$$+ 4(\tau-1)\sum_{l=t_c+1}^{t-1}\mathbb{E}\left\|\frac{1}{n}\sum_{j\in[n]}\nabla_{\boldsymbol{w}} f^j(\overline{\mathbf{w}}_l,\boldsymbol{\psi}_l^j)-\frac{1}{n}\sum_{j\in[n]}\nabla_{\boldsymbol{w}} f^j(\mathbf{w}_l^j,\boldsymbol{\psi}_l^j)\right\|^2$$

$$+ 4(\tau-1)\sum_{l=t_c+1}^{t-1}\mathbb{E}\left\|\frac{1}{n}\sum_{j\in[n]}\nabla_{\boldsymbol{w}} f^j(\mathbf{w}_l^j,\boldsymbol{\psi}_l^j)\right\|^2 \tag{C.64}$$

We can simply this bound by using Assumption 4.3 on Lipschitz gradients for the local objectives f^is and applying the notations for e_l and g_l to derive

$$T_3 \le 4(\tau-1)L_1^2\sum_{l=t_c+1}^{t-1}\mathbb{E}\left\|\mathbf{w}_l^i-\overline{\mathbf{w}}_l\right\|^2 + 4(\tau-1)\sum_{l=t_c+1}^{t-1}\mathbb{E}\left\|\nabla_{\boldsymbol{w}} f^i(\overline{\mathbf{w}}_l,\boldsymbol{\psi}_l^i)-\nabla_{\boldsymbol{w}} f(\overline{\mathbf{w}}_l,\Psi_l)\right\|^2$$

$$+ 4(\tau-1)L_1^2\sum_{l=t_c+1}^{t-1} e_l + 4(\tau-1)\sum_{l=t_c+1}^{t-1} g_l \tag{C.65}$$

We can plug (C.63) and (C.65) into (C.61) and take the average of the both sides over $i = 1, \cdots, n$. This implies that

$$e_t \le 16\eta_1^2(\tau-1)L_1^2\sum_{l=t_c+1}^{t-1} e_l + 10\eta_1^2(\tau-1)\sum_{l=t_c+1}^{t-1} g_l + 8\eta_1^2(\tau-1)^2\rho^2 + 4\eta_1^2(\tau-1)(n+1)\frac{\sigma_{\boldsymbol{w}}^2}{n}. \tag{C.66}$$

In above, we used the result of Proposition C.2 that given Assumption 4.1, bounds the gradient diversity $\frac{1}{n}\sum_{i\in[n]}\|\nabla_{\boldsymbol{w}} f^i(\mathbf{w},\boldsymbol{\psi}^i)-\nabla_{\boldsymbol{w}} f(\mathbf{w},\Psi)\|^2 \le \rho^2$, where $\rho^2 = 3\rho_f^2 + 6L_{12}^2(\epsilon_1^2+\epsilon_2^2)$. We defer the proof this proposition to the end of this section. This concludes the proof of Proposition C.1. ■

Having set the required intermediate steps, we resume the proof of Lemma C.6. According to Proposition C.1, we can write the term e_t as follows

$$e_t \le C_1\sum_{l=t_c+1}^{t-1} e_l + C_2\sum_{l=t_c+1}^{t-1} g_l + C_3 \tag{C.67}$$

where we use the following short-hand coefficients

$$\begin{aligned}
C_1 &:= 16\eta_1^2(\tau-1)L_1^2 \\
C_2 &:= 10\eta_1^2(\tau-1) \\
C_3 &:= 8\eta_1^2(\tau-1)^2\rho^2 + 4\eta_1^2(\tau-1)(n+1)\frac{\sigma_{\boldsymbol{w}}^2}{n}.
\end{aligned} \tag{C.68}$$

We can then write this bound for every iteration in $[t_c+1:t]$, that is

$$\begin{aligned}
e_{t_c+1} &= 0 \\
e_{t_c+2} &\leq C_1 e_{t_c+1} + C_2 g_{t_c+1} + C_3 \\
&\vdots \\
e_t &\leq C_1\left(e_{t_c+1} + \cdots + e_{t-1}\right) + C_2\left(g_{t_c+1} + \cdots + g_{t-1}\right) + C_3.
\end{aligned} \tag{C.69}$$

Summing all of the inequalities results in the following

$$\sum_{l=t_c+1}^{t-1} e_l \leq C_1(\tau-1)\sum_{l=t_c+1}^{t-1} e_l + C_2(\tau-1)\sum_{l=t_c+1}^{t-1} g_l + C_3(\tau-1). \tag{C.70}$$

We can further rearrange the terms above and write

$$\sum_{l=t_c+1}^{t-1} e_l \leq \frac{C_2(\tau-1)}{1-C_1(\tau-1)}\sum_{l=t_c+1}^{t-1} g_l + \frac{C_3(\tau-1)}{1-C_1(\tau-1)}. \tag{C.71}$$

Now, if we assume that $C_1(\tau-1) \leq 1/2$, then we get the following bound on $\sum_{l=t_c+1}^{t-1} e_l$

$$\sum_{l=t_c+1}^{t-1} e_l \leq 2C_2(\tau-1)\sum_{l=t_c+1}^{t-1} g_l + 2C_3(\tau-1) \tag{C.72}$$

Plugging back in (C.90) and using the assumption $C_1(\tau-1) \le 1/2$ yields that

$$\begin{aligned} e_t &\le C_1 \left(2C_2(\tau-1) \sum_{l=t_c+1}^{t-1} g_l + 2C_3(\tau-1) \right) + C_2 \sum_{l=t_c+1}^{t-1} g_l + C_3 \\ &\le 2C_2 \sum_{l=t_c+1}^{t-1} g_l + 2C_3, \end{aligned} \tag{C.73}$$

which concludes the proof of Lemma C.6. Lastly, we present the following proposition along with its proof which we used this result to prove Proposition C.1.

Proposition C.2 *An immediate implication of Assumptions 4.1 and 4.3 is that for any* $\mathbf{w}, \Psi$, *the diversity of the local gradients is bounded in the following sense*

$$\frac{1}{n} \sum_{i\in[n]} \left\| \nabla_{\boldsymbol{w}} f^i(\mathbf{w}, \boldsymbol{\psi}^i) - \nabla_{\boldsymbol{w}} f(\mathbf{w}, \Psi) \right\|^2 \le \rho^2, \tag{C.74}$$

where we denote $\rho^2 = 3\rho_f^2 + 6L_{12}^2(\epsilon_1^2 + \epsilon_2^2)$.

Proof: [Proof of Proposition C.2] The proof is simply implied from Assumptions 4.1 and 4.3 by writing

$$\begin{aligned} \frac{1}{n} \sum_{i\in[n]} \left\| \nabla_{\boldsymbol{w}} f^i(\mathbf{w}, \boldsymbol{\psi}^i) - \nabla_{\boldsymbol{w}} f(\mathbf{w}, \Psi) \right\|^2 &\le 3\frac{1}{n} \sum_{i\in[n]} \left\| \nabla_{\boldsymbol{w}} f^i(\mathbf{w}, \Lambda^i, \delta^i) - \nabla_{\boldsymbol{w}} f^i(\mathbf{w}, I, 0) \right\|^2 \\ &\quad + 3\frac{1}{n} \sum_{i\in[n]} \left\| \nabla_{\boldsymbol{w}} f^i(\mathbf{w}) - \nabla_{\boldsymbol{w}} f(\mathbf{w}) \right\|^2 \\ &\quad + 3\frac{1}{n} \sum_{i\in[n]} \left\| \nabla_{\boldsymbol{w}} f(\mathbf{w}, I, 0) - \nabla_{\boldsymbol{w}} f(\mathbf{w}, \Psi) \right\|^2 \\ &\le 3\rho_f^2 + 6L_{12}^2(\epsilon_1^2 + \epsilon_2^2). \end{aligned} \tag{C.75}$$

■

C.4.7 Proof of Lemma C.7

[70] proves a similar claim for $\Gamma = 0$. For completeness, we provide the proof for general case when $\Gamma \neq 0$. Let t_c denote the index of the most recent communication round, i.e. $t_c = \left\lfloor \frac{t}{\tau} \right\rfloor \tau$. We can write $t = t_c + r$ where $1 \leq r \leq \tau$. Starting from $r = 1$, we can write

$$\begin{aligned} P_{t_c+2} &\leq \Upsilon P_{t_c+1} - \frac{\eta_1}{2}\left(1 - \eta_1 L\right) g_{t_c+1} + \Gamma \\ &\leq \Upsilon P_{t_c+1} + \Gamma, \end{aligned} \tag{C.76}$$

where the last inequality holds if

$$\eta_1 L \leq 1. \tag{C.77}$$

We can continue for $r = 2$ as follows

$$\begin{aligned} P_{t_c+3} &\leq \Upsilon P_{t_c+2} - \frac{\eta_1}{2}\left(1 - \eta_1 L\right) g_{t_c+2} + \eta_1^2 B g_{t_c+1} + \Gamma \\ &\overset{(a)}{\leq} \Upsilon^2 P_{t_c+1} - \frac{\eta_1}{2}\Upsilon\left(1 - \eta_1 L - \eta_1 \frac{2B}{\Upsilon}\right) g_{t_c+1} + \Gamma(1+\Upsilon) \\ &\overset{(b)}{\leq} \Upsilon^2 P_{t_c+1} + \Gamma(1+\Upsilon) \end{aligned} \tag{C.78}$$

where (a) is due to the inequality $P_{t_c+2} \leq \Upsilon P_{t_c+1} - \frac{\eta_1}{2}(1-\eta_1 L)g_{t_c+1} + \Gamma$ and (b) holds if

$$1 - \eta_1 L - \eta_1 \frac{2B}{\Upsilon} \geq 0, \tag{C.79}$$

or equivalently

$$\eta_1\left(L + \frac{2B}{\Upsilon}\right) \leq 1. \tag{C.80}$$

We can continue the same argument up to $r+1$ and write

$$P_{t_c+r+1} \leq \Upsilon^r P_{t_c+1} + \Gamma(1+\Upsilon+\cdots+\Upsilon^{r-1}), \tag{C.81}$$

if the step-size is as small as follows

$$\eta_1 \left(L + \frac{2B}{\Upsilon^{r-1}} \left(1+\Upsilon+\cdots+\Upsilon^{r-2}\right)\right) \leq 1. \tag{C.82}$$

Since $1+\Upsilon+\cdots+\Upsilon^{r-2} \leq \frac{1}{1-\Upsilon}$, then the following condition implies all the previous ones on η

$$\eta_1 \left(L + \frac{2B}{\Upsilon^{r-1}(1-\Upsilon)} \right). \tag{C.83}$$

Moreover, since $\Upsilon < 1$, then the strongest condition on η is (C.83) when we put the largest possible value for r which is τ, yielding

$$\eta_1 \left(L + \frac{2B}{\Upsilon^{\tau-1}(1-\Upsilon)} \right). \tag{C.84}$$

Lastly, we note that $1+\Upsilon+\cdots+\Upsilon^{r-1} \leq \frac{1}{1-\Upsilon}$ in (C.81), and the claim is concluded.

C.4.8 Proof of Lemma C.8

Recall the result of Lemma C.5 in which we showed that if $\eta_2 \leq 1/L_2$, then the following contraction bound on the sequence $\{b_t\}_{t\geq 0}$ holds:

$$\begin{aligned} b_{t+1} \leq (1-\mu_2\eta_2 n)\left(1+\eta_1\frac{4L_{12}^2}{\mu_2 n}\right) b_t + \frac{\eta_1}{2}\mathbb{E}\left\|\nabla\Phi(\overline{\mathbf{w}}_t)\right\|^2 + \frac{\eta_1^2}{2}\left(L_1+L_\Phi+2\eta_2 L_{21}^2\right) g_t \\ + \left(\eta_1 L_1^2 + \eta_2 L_{21}^2\right) e_t + \frac{\eta_1^2}{2}\left(L_1+L_\Phi+2\eta_2 L_{21}^2\right)\frac{\sigma_{\boldsymbol{w}}^2}{n} + \frac{\eta_2^2}{2} L_2 \sigma_\psi^2, \end{aligned} \tag{C.85}$$

and consider the coefficient of b_t in above. A simple calculation yields that if the step-sizes satisfy the condition $\frac{\eta_2}{\eta_1} \geq \frac{8L_{12}^2}{\mu_2^2 n^2}$, then we have

$$(1 - \mu_2\eta_2 n)\left(1 + \eta_1 \frac{4L_{12}^2}{\mu_2 n}\right) \leq 1 - \frac{1}{2}\mu_2\eta_2 n. \tag{C.86}$$

Now, we denote $\gamma = 1 - \frac{1}{2}\mu_2\eta_2 n$ and apply (C.85) to all iterations $t = 0, \cdots, T-1$, which yields that

$$\begin{aligned}
b_0 &\leq \frac{2L_2^2}{\mu_2 n}\left(\epsilon_1^2 + \epsilon_2^2\right), \\
b_1 &\leq \gamma b_0 + \frac{\eta_1}{2}\mathbb{E}\left\|\nabla\Phi(\overline{\mathbf{w}}_t)\right\|^2 + \frac{\eta_1^2}{2}\left(L_1 + L_\Phi + 2\eta_2 L_{21}^2\right) g_0 + \left(\eta_1 L_1^2 + \eta_2 L_{21}^2\right) e_0 \\
&\quad + \frac{\eta_1^2}{2}\left(L_1 + L_\Phi + 2\eta_2 L_{21}^2\right)\frac{\sigma_{\boldsymbol{w}}^2}{n} + \frac{\eta_2^2}{2}L_2\sigma_\psi^2, \\
&\vdots \\
b_{T-1} &\leq \gamma b_{T-2} + \frac{\eta_1}{2}\mathbb{E}\left\|\nabla\Phi(\overline{\mathbf{w}}_t)\right\|^2 + \frac{\eta_1^2}{2}\left(L_1 + L_\Phi + 2\eta_2 L_{21}^2\right) g_{T-2} + \left(\eta_1 L_1^2 + \eta_2 L_{21}^2\right) e_{T-2} \\
&\quad + \frac{\eta_1^2}{2}\left(L_1 + L_\Phi + 2\eta_2 L_{21}^2\right)\frac{\sigma_{\boldsymbol{w}}^2}{n} + \frac{\eta_2^2}{2}L_2\sigma_\psi^2.
\end{aligned} \tag{C.87}$$

Taking the average of the T inequalities above yields that

$$\begin{aligned}
(1-\gamma)\frac{1}{T}\sum_{t=0}^{T-1} b_t \leq{}& \frac{2L_2^2}{\mu_2 n}\frac{\epsilon_1^2 + \epsilon_2^2}{T} + \frac{\eta_1}{2}\frac{1}{T}\sum_{t=0}^{T-1}\mathbb{E}\left\|\nabla\Phi(\overline{\mathbf{w}}_t)\right\|^2 \\
&+ \frac{\eta_1^2}{2}\left(L_1 + L_\Phi + 2\eta_2 L_{21}^2\right)\frac{1}{T}\sum_{t=0}^{T-1} g_t + \left(\eta_1 L_1^2 + \eta_2 L_{21}^2\right)\frac{1}{T}\sum_{t=0}^{T-1} e_t \\
&+ \frac{\eta_1^2}{2}\left(L_1 + L_\Phi + 2\eta_2 L_{21}^2\right)\frac{\sigma_{\boldsymbol{w}}^2}{n} + \frac{\eta_2^2}{2}L_2\sigma_\psi^2.
\end{aligned} \tag{C.88}$$

We can further divide both sides of (C.88) by $1-\gamma$ and conclude

$$\frac{1}{T}\sum_{t=0}^{T-1} b_t \leq \frac{4L_2^2}{\mu_2^2 n^2}\frac{\epsilon_1^2 + \epsilon_2^2}{\eta_2 T} + \frac{\eta_1}{\eta_2}\frac{1}{\mu_2 n}\frac{1}{T}\sum_{t=0}^{T-1}\mathbb{E}\left\|\nabla\Phi(\overline{\mathbf{w}}_t)\right\|^2$$

$$+ \frac{\eta_1^2}{\eta_2}\frac{1}{\mu_2 n}\left(L_1 + L_\Phi + 2\eta_2 L_{21}^2\right)\frac{1}{T}\sum_{t=0}^{T-1} g_t + \frac{1}{\eta_2}\frac{2}{\mu_2 n}\left(\eta_1 L_1^2 + \eta_2 L_{21}^2\right)\frac{1}{T}\sum_{t=0}^{T-1} e_t$$
$$+ \frac{\eta_1^2}{\eta_2}\frac{1}{\mu_2 n}\left(L_1 + L_\Phi + 2\eta_2 L_{21}^2\right)\frac{\sigma_{\boldsymbol{w}}^2}{n} + \eta_2 \frac{L_2}{\mu_2 n}\sigma_\psi^2. \tag{C.89}$$

C.4.9 Proof of Lemma C.9

We begin by noting the result of Proposition C.1 in which we showed the following bound on e_t

$$e_t \leq C_1 \sum_{l=t_c+1}^{t-1} e_l + C_2 \sum_{l=t_c+1}^{t-1} g_l + C_3, \tag{C.90}$$

where we defined the coefficients C_1, C_2, C_3 in (C.68) and recall here for more convenient:

$$\begin{aligned} C_1 &\coloneqq 16\eta_1^2(\tau-1)L_1^2 \\ C_2 &\coloneqq 10\eta_1^2(\tau-1) \\ C_3 &\coloneqq 8\eta_1^2(\tau-1)^2\rho^2 + 4\eta_1^2(\tau-1)(n+1)\frac{\sigma_{\boldsymbol{w}}^2}{n}. \end{aligned} \tag{C.91}$$

Next, we apply this bound to each iteration $t = 0, \cdots, T-1$ as follows

$$e_0 = 0$$
$$\begin{cases} e_1 &= 0 \\ e_2 &\leq C_1 e_1 + C_2 g_1 + C_3 \\ \vdots & \\ e_\tau &\leq C_1\left(e_1 + \cdots + e_{\tau-1}\right) + C_2\left(g_1 + \cdots + g_{\tau-1}\right) + C_3 \end{cases}$$

$$\begin{cases} e_{\tau+1} & = 0 \\ e_{\tau+2} & \leq C_1 e_{\tau+1} + C_2 g_{\tau+1} + C_3 \\ \vdots & \\ e_{2\tau} & \leq C_1 \left(e_{\tau+1} + \cdots + e_{2\tau-1}\right) + C_2 \left(g_{\tau+1} + \cdots + g_{2\tau-1}\right) + C_3 \end{cases}$$

$$\vdots$$

$$\begin{cases} e_{T_c+1} & = 0 \\ e_{T_c+2} & \leq C_1 e_{T_c+1} + C_2 g_{T_c+1} + C_3 \\ \vdots & \\ e_{T-1} & \leq C_1 \left(e_{T_c+1} + \cdots + e_{T-2}\right) + C_2 \left(g_{T_c+1} + \cdots + g_{T-2}\right) + C_3, \end{cases} \tag{C.92}$$

where $T_c = \left\lfloor \frac{T}{\tau} \right\rfloor \tau$ denote the index of the most recent communication between the workers and the server before iteration T. Summing the above inequalities yields that

$$\sum_{t=0}^{T-1} e_t \leq C_1(\tau - 1) \sum_{t=0}^{T-1} e_t + C_2(\tau - 1) \sum_{t=0}^{T-1} g_t + C_3 T. \tag{C.93}$$

Now if we assume that $C_1(\tau - 1) = 16\eta_1^2(\tau - 1)^2 L_1^2 \leq \frac{1}{2}$, the the claim is concluded by rearranging the terms in (C.93):

$$\frac{1}{T} \sum_{t=0}^{T-1} e_t \leq 2C_2(\tau - 1) \frac{1}{T} \sum_{t=0}^{T-1} g_t + 2C_3. \tag{C.94}$$

C.5 Proof of Theorem 4.3

Fix a distribution $\tilde{P}$ and consider

$$\max_{\Lambda,\delta} \mathbb{E}_{\tilde{P}}[\ell(f_{\mathbf{w}}(\Lambda \mathbf{x} + \delta))] - \lambda \|\delta\|_2^2 - \lambda \|\Lambda - I\|_F^2 \tag{C.95}$$

Assuming a 1-Lipschitz loss ℓ with 1-Lipschitz gradient, based on [103]'s Lemma 7 the above function's gradient with respect to δ has a Lipschitz constant bounded by

$$\mathrm{Lip}(\nabla f_{\mathbf{w}}) := \big(\prod_{i=1}^{L} \|\mathbf{w}_i\|_\sigma\big) \sum_{i=1}^{l} \prod_{j=1}^{i} \|\mathbf{w}_j\|_\sigma .$$

Similarly, the expected loss's derivative with respect to Λ will also be Lipschitz in the spectral norm with a Lipschitz constant upper-bounded by

$$B\,\mathrm{Lip}(\nabla f_{\mathbf{w}}) = B\big(\prod_{i=1}^{L} \|\mathbf{w}_i\|_\sigma\big) \sum_{i=1}^{l} \prod_{j=1}^{i} \|\mathbf{w}_j\|_\sigma .$$

Given weights in $\mathbf{w}$, we denote the optimal solution for δ and Λ by $\delta_{\mathbf{w}}$ and $\Lambda_{\mathbf{w}}$, respectively. To apply the Pac-Bayes generalization analysis, we need to bound the change in $\delta_{\mathbf{w}}, \Lambda_{\mathbf{w}}$ caused by perturbing $\mathbf{w}$ to $\mathbf{w}+\boldsymbol{u}$. Note that since $\lambda > (1+B)\,\mathrm{Lip}(\nabla f_{\mathbf{w}})$, the maximization problem for optimizing $\Lambda_{\mathbf{w}}, \delta_{\mathbf{w}}$ is maximizing a strongly-concave objective whose solutions will satisfy:

$$\delta_{\mathbf{w}} = \frac{1}{\lambda}\mathbb{E}[\nabla\ell \circ f_{\mathbf{w}}(\Lambda_{\mathbf{w}}\mathbf{x} + \delta_{\mathbf{w}})],$$
$$\Lambda_{\mathbf{w}} - I = \frac{1}{\lambda}\mathbb{E}[(\nabla\ell \circ f_{\mathbf{w}}(\Lambda_{\mathbf{w}}\mathbf{x} + \delta_{\mathbf{w}}))\mathbf{X}^\top]$$

which are norm-bounded by $\frac{\mathrm{Lip}(\ell\circ f_{\mathbf{w}})}{\lambda} \le \frac{\prod_{i=1}^{d}\|\mathbf{w}_i\|_\sigma}{\lambda}$ and $B\frac{\mathrm{Lip}(\ell\circ f_{\mathbf{w}})}{\lambda} \le B\frac{\prod_{i=1}^{d}\|\mathbf{w}_i\|_\sigma}{\lambda}$, respectively. Therefore, for a norm-bounded perturbation $\boldsymbol{u}$ where $\|\boldsymbol{u}_i\|_\sigma \le \frac{1}{L}\|\mathbf{w}_i\|_\sigma$ we can write

$$\begin{aligned}
&\big\|\delta_{\mathbf{w}+\boldsymbol{u}} - \delta_{\mathbf{w}}\big\|_2 + \big\|\Lambda_{\mathbf{w}+\boldsymbol{u}} - \Lambda_{\mathbf{w}}\big\|_\sigma \\
&= \big\|\frac{1}{\lambda}\mathbb{E}[\nabla\ell(f_{\mathbf{w}+\boldsymbol{u}}(\Lambda_{\mathbf{w}+\boldsymbol{u}}\mathbf{X} + \delta_{\mathbf{w}+\boldsymbol{u}}))] - \frac{1}{\lambda}\mathbb{E}[\nabla\ell(f_{\mathbf{w}}(\Lambda_{\mathbf{w}}\mathbf{X} + \delta_{\mathbf{w}}))]\big\|_2 \\
&\quad + \big\|\frac{1}{\lambda}\mathbb{E}[\nabla\ell(f_{\mathbf{w}+\boldsymbol{u}}(\Lambda_{\mathbf{w}+\boldsymbol{u}}\mathbf{X} + \delta_{\mathbf{w}+\boldsymbol{u}}))\mathbf{X}^\top] - \frac{1}{\lambda}\mathbb{E}[\nabla\ell(f_{\mathbf{w}}(\Lambda_{\mathbf{w}}\mathbf{X} + \delta_{\mathbf{w}}))\mathbf{X}^\top]\big\|_\sigma
\end{aligned}$$

$$
\begin{aligned}
&= \big\| \frac{1}{\lambda}\mathbb{E}[\nabla \ell(f_{\mathbf{w}+\boldsymbol{u}}(\Lambda_{\mathbf{w}+\boldsymbol{u}}\mathbf{X} + \delta_{\mathbf{w}+\boldsymbol{u}})) - \nabla \ell(f_{\mathbf{w}}(\Lambda_{\mathbf{w}}\mathbf{X} + \delta_{\mathbf{w}}))]\big\|_2 \\
&\qquad + \big\| \frac{1}{\lambda}\mathbb{E}[(\nabla \ell(f_{\mathbf{w}+\boldsymbol{u}}(\Lambda_{\mathbf{w}+\boldsymbol{u}}\mathbf{X} + \delta_{\mathbf{w}+\boldsymbol{u}})) - \nabla \ell(f_{\mathbf{w}}(\Lambda_{\mathbf{w}}\mathbf{X} + \delta_{\mathbf{w}})))\mathbf{X}^\top]\big\|_\sigma \\
&\le \big\| \frac{1}{\lambda}\mathbb{E}[\nabla \ell(f_{\mathbf{w}+\boldsymbol{u}}(\Lambda_{\mathbf{w}+\boldsymbol{u}}\mathbf{X} + \delta_{\mathbf{w}+\boldsymbol{u}})) - \nabla \ell(f_{\mathbf{w}}(\Lambda_{\mathbf{w}+\boldsymbol{u}}\mathbf{X} + \delta_{\mathbf{w}+\boldsymbol{u}}))]\big\|_2 \\
&\qquad + \big\| \frac{1}{\lambda}\mathbb{E}[\nabla \ell(f_{\mathbf{w}}(\Lambda_{\mathbf{w}+\boldsymbol{u}}\mathbf{X} + \delta_{\mathbf{w}+\boldsymbol{u}})) - \nabla \ell(f_{\mathbf{w}}(\Lambda_{\mathbf{w}}\mathbf{X} + \delta_{\mathbf{w}+\boldsymbol{u}}))]\big\|_2 \\
&\qquad + \big\| \frac{1}{\lambda}\mathbb{E}[\nabla \ell(f_{\mathbf{w}}(\Lambda_{\mathbf{w}}\mathbf{X} + \delta_{\mathbf{w}+\boldsymbol{u}})) - \nabla \ell(f_{\mathbf{w}}(\Lambda_{\mathbf{w}}\mathbf{X} + \delta_{\mathbf{w}}))]\big\|_2 \\
&\qquad + \big\| \frac{1}{\lambda}\mathbb{E}[(\nabla \ell(f_{\mathbf{w}+\boldsymbol{u}}(\Lambda_{\mathbf{w}+\boldsymbol{u}}\mathbf{X} + \delta_{\mathbf{w}+\boldsymbol{u}})) - \nabla \ell(f_{\mathbf{w}}(\Lambda_{\mathbf{w}+\boldsymbol{u}}\mathbf{X} + \delta_{\mathbf{w}+\boldsymbol{u}})))\mathbf{X}^\top]\big\|_\sigma \\
&\qquad + \big\| \frac{1}{\lambda}\mathbb{E}[(\nabla \ell(f_{\mathbf{w}}(\Lambda_{\mathbf{w}+\boldsymbol{u}}\mathbf{X} + \delta_{\mathbf{w}+\boldsymbol{u}})) - \nabla \ell(f_{\mathbf{w}}(\Lambda_{\mathbf{w}}\mathbf{X} + \delta_{\mathbf{w}+\boldsymbol{u}})))\mathbf{X}^\top]\big\|_\sigma \\
&\qquad + \big\| \frac{1}{\lambda}\mathbb{E}[(\nabla \ell(f_{\mathbf{w}}(\Lambda_{\mathbf{w}}\mathbf{X} + \delta_{\mathbf{w}+\boldsymbol{u}})) - \nabla \ell(f_{\mathbf{w}}(\Lambda_{\mathbf{w}}\mathbf{X} + \delta_{\mathbf{w}})))\mathbf{X}^\top]\big\|_\sigma \\
&\le \frac{(B+1)\,\mathrm{lip}(\ell \circ f_{\mathbf{w}})}{\lambda}\big(\|\delta_{\mathbf{w}+\boldsymbol{u}} - \delta_{\mathbf{w}}\|_2 + \|\Lambda_{\mathbf{w}+\boldsymbol{u}} - \Lambda_{\mathbf{w}}\|_\sigma\big) \\
&\qquad + (B+1)e^2(\prod_{i=1}^{L}\|\mathbf{w}_i\|_\sigma)\sum_{i=1}^{d}\Big[\frac{\|\boldsymbol{u}_i\|_\sigma}{\|\mathbf{w}_i\|_\sigma} + B(\prod_{j=1}^{i}\|\mathbf{w}_j\|_\sigma)\sum_{j=1}^{i}\frac{\|\boldsymbol{u}_j\|_\sigma}{\|\mathbf{w}_j\|_\sigma}\Big],
\end{aligned}
$$

where the last inequality follows from Lemma 3 in [103]. As a result,

$$
\begin{aligned}
&\big\|\delta_{\mathbf{w}+\boldsymbol{u}} - \delta_{\mathbf{w}}\big\|_2 + \big\|\Lambda_{\mathbf{w}+\boldsymbol{u}} - \Lambda_{\mathbf{w}}\big\|_\sigma \\
&\le \frac{\lambda}{\lambda - (B+1)\,\mathrm{lip}(\ell \circ f_{\mathbf{w}})}\Big[(B+1)e^2(\prod_{i=1}^{L}\|\mathbf{w}_i\|_\sigma)\sum_{i=1}^{d}\big[\frac{\|\boldsymbol{u}_i\|_\sigma}{\|\mathbf{w}_i\|_\sigma} + B(\prod_{j=1}^{i}\|\mathbf{w}_j\|_\sigma)\sum_{j=1}^{i}\frac{\|\boldsymbol{u}_j\|_\sigma}{\|\mathbf{w}_j\|_\sigma}\big]\Big].
\end{aligned}
$$

Then, we can bound the change in the loss function caused by perturbing $\mathbf{w}$ at any $\|\mathbf{x}\|_2 \le B$ with any norm-bounded $\|\boldsymbol{u}_i\|_\sigma \le \frac{1}{L}\|\mathbf{w}_i\|_\sigma$:

$$
\begin{aligned}
&\big\|f_{\mathbf{w}+\boldsymbol{u}}(\Lambda_{\mathbf{w}+\boldsymbol{u}}\mathbf{X} + \delta_{\mathbf{w}+\boldsymbol{u}}) - f_{\mathbf{w}}(\Lambda_{\mathbf{w}}\mathbf{X} + \delta_{\mathbf{w}})\big\|_2 \\
&\le \big\|f_{\mathbf{w}+\boldsymbol{u}}(\Lambda_{\mathbf{w}+\boldsymbol{u}}\mathbf{X} + \delta_{\mathbf{w}+\boldsymbol{u}}) - f_{\mathbf{w}}(\Lambda_{\mathbf{w}+\boldsymbol{u}}\mathbf{X} + \delta_{\mathbf{w}+\boldsymbol{u}})\big\|_2 \\
&\qquad + \big\|f_{\mathbf{w}}(\Lambda_{\mathbf{w}+\boldsymbol{u}}\mathbf{X} + \delta_{\mathbf{w}+\boldsymbol{u}}) - f_{\mathbf{w}}(\Lambda_{\mathbf{w}}\mathbf{X} + \delta_{\mathbf{w}+\boldsymbol{u}})\big\|_2 \\
&\qquad + \big\|f_{\mathbf{w}}(\Lambda_{\mathbf{w}}\mathbf{X} + \delta_{\mathbf{w}+\boldsymbol{u}}) - f_{\mathbf{w}}(\Lambda_{\mathbf{w}}\mathbf{X} + \delta_{\mathbf{w}})\big\|_2
\end{aligned}
$$

$$\leq eB\big(\prod_{i=1}^{L}\|\mathbf{w}_i\|_\sigma\big)\sum_{i=1}^{L}\frac{\|\boldsymbol{u}_i\|_2}{\|\mathbf{w}_i\|_2} + (1+B)\big(\prod_{i=1}^{d}\|\mathbf{w}_i\|_\sigma\big)$$
$$\frac{e^2}{\lambda-(B+1)\,\mathrm{Lip}(\nabla f_{\mathbf{w}})}\sum_{i=1}^{L}\big[\frac{\|\boldsymbol{u}_i\|_\sigma}{\|\mathbf{w}_i\|_\sigma} + B(\prod_{j=1}^{i}\|\mathbf{w}_j\|_\sigma)\sum_{j=1}^{i}\frac{\|\boldsymbol{u}_j\|_\sigma}{\|\mathbf{w}_j\|_\sigma}\big].$$

Now, for a fixed weight vector $\tilde{\mathbf{w}}$ we consider a multivariate Gaussian distribution Q with zero-mean and diagonal covaraince matrix for perturbation $\boldsymbol{u}$ where each entry $\boldsymbol{u}_i$ has standard deviation $\kappa_i = \frac{\|\tilde{\mathbf{w}}_i\|_\sigma}{\sqrt[L]{\prod_{i=1}^{L}\|\tilde{\mathbf{w}}_i\|_\sigma}}\kappa$ with κ chosen as

$$\kappa = \frac{\gamma}{8e^5 L\sqrt{2d\log(4dL)}B\big(\prod_{i=1}^{L}\|\tilde{\mathbf{w}}_i\|_\sigma\big)\big(1+\frac{\lambda}{\lambda-(1+B)\overline{\mathrm{Lip}}(\nabla f_{\mathbf{w}})}\sum_{i=1}^{L}\prod_{j=1}^{i}\|\tilde{\mathbf{w}}_j\|_\sigma\big)}. \quad \text{(C.96)}$$

Also, for any $\mathbf{w}$ which satisfies $|\|\mathbf{w}_i\|_\sigma - \|\tilde{\mathbf{w}}_i\|_\sigma| \leq \frac{\eta}{4L}\|\tilde{\mathbf{w}}_i\|_\sigma$, we have $\overline{\mathrm{Lip}}(\ell \circ f_{\mathbf{w}}) \leq e^{\eta/2}\lambda(1-\eta) \leq (1-\eta/2)\lambda$. Therefore,

$$\begin{aligned}
&\mathrm{KL}(P_{\mathbf{w}+\boldsymbol{u}}\|Q)\\
&\leq \sum_{i=1}^{d}\frac{\|\mathbf{w}_i\|_F^2}{2\kappa_i^2}\\
&\leq O\Bigg(L^2B^2 d\log(dL)\frac{\big(\prod_{i=1}^{L}\|\tilde{\mathbf{w}}_i\|_\sigma^2\big)\big(1+\frac{1}{\lambda-(1+B)\overline{\mathrm{Lip}}(\nabla f_{\mathbf{w}})}\sum_{i=1}^{L}\prod_{j=1}^{i}\|\tilde{\mathbf{w}}_j\|_\sigma\big)^2}{\gamma^2}\sum_{i=1}^{d}\frac{\|\mathbf{w}_i\|_F^2}{\|\tilde{\mathbf{w}}_i\|_\sigma^2}\Bigg)\\
&\leq O\Bigg(L^2B^2 d\log(dL)\frac{\big(\prod_{i=1}^{L}\|\mathbf{w}_i\|_\sigma^2\big)\big(1+\frac{1}{\lambda-(1+B)\overline{\mathrm{Lip}}(\nabla f_{\mathbf{w}})}\sum_{i=1}^{L}\prod_{j=1}^{i}\|\mathbf{w}_j\|_\sigma\big)^2}{\gamma^2}\sum_{i=1}^{d}\frac{\|\mathbf{w}_i\|_F^2}{\|\mathbf{w}_i\|_\sigma^2}\Bigg)
\end{aligned}$$

Now we plug the above result into [103]'s Lemma 1, implying that given a fixed underlying distribution P and any $\xi > 0$ with probability at least $1-\xi$ for any $\mathbf{w}$ satisfying $|\|\mathbf{w}_i\|_\sigma -$

$\|\tilde{\mathbf{w}}_i\|_\sigma| \le \frac{\eta}{4L}\|\tilde{\mathbf{w}}_i\|_\sigma$ we have

$$\mathcal{L}^{\text{adv}}_{0-1}(\mathbf{w}) - \hat{\mathcal{L}}^{\text{adv}}_{\gamma}(\mathbf{w}) \le \mathcal{O}\left(\sqrt{\frac{B^2L^2d\log(Ld)\lambda^2\big(\prod_{i=1}^L\|\mathbf{w}_i\|_\sigma\sum_{i=1}^L\frac{\|\mathbf{w}_i\|_F^2}{\|\mathbf{w}_i\|_\sigma^2}\big)^2 + \log\frac{m}{\xi}}{m\gamma^2(\lambda-(1+B)\operatorname{Lip}(\nabla f_{\mathbf{w}}))^2}}\right). \tag{C.97}$$

Now we use a cover of size $O(\frac{L}{\eta}\log M)$ points where for any feasible $\|\mathbf{w}_i\|_\sigma$ we can find a point a_i in the cover such that $|\|\mathbf{w}_i\|_\sigma - a_i| \le \frac{\eta}{4L}a_i$. As a result, we can cover the space of feasible $\mathbf{w}_i$'s with $O\big((\frac{L}{\eta}\log M))^L L\big)$ number of points. This proves that for a fixed underlying distribution for every $\xi > 0$, with probability at least $\xi > 0$ for any feasible norm-bounded $\mathbf{w}$ we have

$$\begin{aligned}&\mathcal{L}^{\text{adv}}_{0-1}(\mathbf{w}) - \hat{\mathcal{L}}^{\text{adv}}_{\gamma}(\mathbf{w})\\ &\le \mathcal{O}\left(\sqrt{\frac{B^2L^2d\log(Ld)\lambda^2\big(\prod_{i=1}^L\|\mathbf{w}_i\|_\sigma\sum_{i=1}^L\frac{\|\mathbf{w}_i\|_2^2}{\|\mathbf{w}_i\|_\sigma^2}\big)^2 + L\log\frac{mL\log(M)}{\eta\xi}}{m\gamma^2(\lambda-(1+B)\operatorname{Lip}(\nabla\ell\circ f_{\mathbf{w}}))^2}}\right).\end{aligned} \tag{C.98}$$

To apply the result to the network of n nodes, we apply a union bound to have the bound hold simultaneously for the distribution of every node, which proves for every $\xi > 0$ with probability at least $1-\xi$ the average worst-case loss of the nodes satisfies the following margin-based bound:

$$\begin{aligned}&\mathcal{L}^{\text{adv}}_{0-1}(\mathbf{w}) - \hat{\mathcal{L}}^{\text{adv}}_{\gamma}(\mathbf{w})\\ &\le \mathcal{O}\left(\sqrt{\frac{B^2L^2d\log(Ld)\lambda^2\big(\prod_{i=1}^L\|\mathbf{w}_i\|_\sigma\sum_{i=1}^L\frac{\|\mathbf{w}_i\|_F^2}{\|\mathbf{w}_i\|_\sigma^2}\big)^2 + L\log\frac{nmL\log(M)}{\eta\xi}}{m\gamma^2(\lambda-(1+B)\operatorname{Lip}(\nabla f_{\mathbf{w}}))^2}}\right).\end{aligned} \tag{C.99}$$

Therefore, the proof is complete.

C.6 Proof of Theorem 4.4

Define random vector $\mathbf{U} = \Lambda\mathbf{X} + \delta$. According to the definition of optimal transport cost $W_c(P_{\mathbf{X}}, P_{\mathbf{U}})$ for quadratic $c(\mathbf{x}, \mathbf{u}) = \frac{1}{2}\|\mathbf{x} - \mathbf{u}\|_2^2$,

$$W_c(P_{\mathbf{X}}, P_{\mathbf{U}}) := \min_{P_{\mathbf{X},\mathbf{U}} \in \Pi(P_{\mathbf{X}}, P_{\mathbf{U}})} \mathbb{E}\big[\frac{1}{2}\|\mathbf{X} - \mathbf{U}\|_2^2\big] \tag{C.100}$$

where $\Pi(P_{\mathbf{X}}, P_{\mathbf{U}})$ contains any joint distribution $P_{\mathbf{X},\mathbf{U}}$ with marginals $P_{\mathbf{X}}, P_{\mathbf{U}}$. One distribution in $\Pi(P_{\mathbf{X}}, P_{\mathbf{U}})$ is the joint distribution of $(\mathbf{X}, \Lambda\mathbf{X} + \delta)$ implying that

$$\begin{aligned}
W_c(P_{\mathbf{X}}, P_{\mathbf{U}}) &\le \frac{1}{2}\mathbb{E}\big[\|\mathbf{X} - \Lambda\mathbf{X} - \delta\|_2^2\big] \\
&= \frac{1}{2}\mathbb{E}\big[\|(I - \Lambda)\mathbf{X} - \delta\|_2^2\big] \\
&\overset{(a)}{\le} \mathbb{E}\big[\|(I - \Lambda)\mathbf{X}\|_2^2\big] + \|\delta\|_2^2 \\
&\overset{(b)}{\le} \operatorname{Tr}\big((I - \Lambda)(I - \Lambda)^\top \mathbb{E}[\mathbf{X}\mathbf{X}^\top]\big) + \|\delta\|_2^2 \\
&\overset{(c)}{\le} \lambda \operatorname{Tr}\big((I - \Lambda)(I - \Lambda)^\top\big) + \|\delta\|_2^2 \\
&\overset{(d)}{\le} \lambda\|I - \Lambda\|_F^2 + \|\delta\|_2^2 \\
&\le \max\{\lambda, 1\}\big(\|I - \Lambda\|_F^2 + \|\delta\|_2^2\big).
\end{aligned}$$

In the above, (a) holds since for every two vectors $\mathbf{u}_1, \mathbf{u}_2$ we have $\|\mathbf{u}_1 + \mathbf{u}_2\|_2^2 = \|\mathbf{u}_1\|_2^2 + \|\mathbf{u}_2\|_2^2 + 2\mathbf{u}_1^\top \mathbf{u}_2 \le 2(\|\mathbf{u}_1\|_2^2 + \|\mathbf{u}_2\|_2^2)$. (b) follows from the fact that $\mathbb{E}[\|(I - \Lambda)\mathbf{X}\|_2^2] = \mathbb{E}[\operatorname{Tr}((I - \Lambda)\mathbf{X}\mathbf{X}^\top(I - \Lambda)^\top) = \operatorname{Tr}\big((I - \Lambda)(I - \Lambda)^\top\mathbb{E}[\mathbf{X}\mathbf{X}^\top]\big)$. (c) holds because of the theorem's assumption implying that $\mathbb{E}[\mathbf{X}\mathbf{X}^\top] \le \lambda I$. Last, (d) holds because we have $\operatorname{Tr}(AA^\top) = \|A\|_F^2$ for every A. Therefore, the proof is complete.

Appendix D

Supplements to Chapter 5

D.1 Proof of Theorem 5.1

In this section, we prove the asymptotic optimality of HCMM as claimed in Theorem 5.1.

Consider the HCMM load assignment in (5.7). Let the random variable $T_{\texttt{HCMM}}$ denote the finish time associated to this load allocation, i.e. the waiting time to receive at least r inner products from the workers. Let T_{max} be the random variable denoting the finish time of all the workers for the HCMM load assignment.

First, we show that

$$\mathbb{E}[T_{\texttt{HCMM}}] \leq t^* + o(1).$$

Let us define two events $\mathcal{E}_1$ and $\mathcal{E}_2$ as follows:

$$\mathcal{E}_1 = \{T_{\mathsf{max}} > \Theta(n)\} \text{ and } \mathcal{E}_2 = \{T_{\texttt{HCMM}} > t^*\}.$$

Conditioning on these events, we can write

$$\begin{aligned}\mathbb{E}[T_{\texttt{HCMM}}] = \mathbb{E}[T_{\texttt{HCMM}}|\mathcal{E}_1]\Pr[\mathcal{E}_1] + \mathbb{E}[T_{\texttt{HCMM}}|{\mathcal{E}_1}^c \cap \mathcal{E}_2]\Pr[{\mathcal{E}_1}^c \cap \mathcal{E}_2] \\ + \mathbb{E}[T_{\texttt{HCMM}}|{\mathcal{E}_1}^c \cap {\mathcal{E}_2}^c]\Pr[{\mathcal{E}_1}^c \cap {\mathcal{E}_2}^c]. \qquad \text{(D.1)}\end{aligned}$$

We can write the second term in RHS of (D.1) as follows:

$$\begin{aligned}&\mathbb{E}[T_{\texttt{HCMM}}|{\mathcal{E}_1}^c \cap \mathcal{E}_2]\Pr[{\mathcal{E}_1}^c \cap \mathcal{E}_2] \\ &= \mathbb{E}[T_{\texttt{HCMM}}|T_{\mathsf{max}} \leq \Theta(n), T_{\texttt{HCMM}} > t^*]\Pr[T_{\mathsf{max}} \leq \Theta(n), T_{\texttt{HCMM}} > t^*] \\ &\leq \mathbb{E}[T_{\mathsf{max}}|T_{\mathsf{max}} \leq \Theta(n), T_{\texttt{HCMM}} > t^*]\Pr[T_{\texttt{HCMM}} > t^*] \\ &\overset{(a)}{\leq} \Theta(n) \cdot o\left(\frac{1}{n}\right) \\ &= o(1). \qquad \text{(D.2)}\end{aligned}$$

To prove (a), we note that `HCMM` returns r inner products by time $T_{\texttt{HCMM}}$. Moreover, the aggregate return is increasing in time. Therefore,

$$\Pr[T_{\texttt{HCMM}} > t^*] \leq \Pr[X^*(t^*) < r] = o\left(\frac{1}{n}\right).$$

Furthermore, we have

$$\begin{aligned}&\mathbb{E}[T_{\mathsf{max}}|T_{\mathsf{max}} \leq \Theta(n), T_{\texttt{HCMM}} > t^*] \\ &= \frac{1}{\Pr[T_{\mathsf{max}} \leq \Theta(n), T_{\texttt{HCMM}} > t^*]} \int_{t_1=0}^{\Theta(n)} \int_{t_2=t^*}^{\infty} t_1 d\Pr[T_{\mathsf{max}} \leq t_1, T_{\texttt{HCMM}} \leq t_2] \\ &\leq \frac{\Theta(n)}{\Pr[T_{\mathsf{max}} \leq \Theta(n), T_{\texttt{HCMM}} > t^*]} \int_{t_1=0}^{\Theta(n)} \int_{t_2=t^*}^{\infty} d\Pr[T_{\mathsf{max}} \leq t_1, T_{\texttt{HCMM}} \leq t_2] \\ &= \Theta(n).\end{aligned}$$

Moreover, the third term in RHS of (D.1) can be written as

$$\begin{aligned}
&\mathbb{E}[T_{\texttt{HCMM}}|{\mathcal{E}_1}^c \cap {\mathcal{E}_2}^c] \Pr[{\mathcal{E}_1}^c \cap {\mathcal{E}_2}^c] \\
&= \mathbb{E}[T_{\texttt{HCMM}}|T_{\mathsf{max}} \leq \Theta(n), T_{\texttt{HCMM}} \leq t^*] \Pr[T_{\mathsf{max}} \leq \Theta(n), T_{\texttt{HCMM}} \leq t^*] \\
&\leq \mathbb{E}[T_{\texttt{HCMM}}|T_{\mathsf{max}} \leq \Theta(n), T_{\texttt{HCMM}} \leq t^*] \\
&\overset{(b)}{\leq} t^*, \qquad \text{(D.3)}
\end{aligned}$$

where proof of (b) is similar to proof of (a) in (D.2). Regarding the first term in RHS of (D.1), we have

$$\begin{aligned}
\mathbb{E}[T_{\texttt{HCMM}}|\mathcal{E}_1] \Pr[\mathcal{E}_1] &= \mathbb{E}[T_{\texttt{HCMM}}|T_{\mathsf{max}} > \Theta(n)] \Pr[T_{\mathsf{max}} > \Theta(n)] \\
&\leq \mathbb{E}[T_{\mathsf{max}}|T_{\mathsf{max}} > \Theta(n)] \Pr[T_{\mathsf{max}} > \Theta(n)] \\
&= \int_{\Theta(n)}^{\infty} t f_{\mathsf{max}}(t)\, dt \\
&\overset{(c)}{\leq} \int_{\Theta(n)}^{\infty} tnk_1 e^{-k_1 t} \left(1 - e^{-k_1 t}\right)^{n-1} dt \\
&\leq \int_{\Theta(n)}^{\infty} nk_1 t e^{-k_1 t}\, dt \\
&\leq \int_{\Theta(n)}^{\infty} \frac{1}{t^2}\, dt = o(1), \qquad \text{(D.4)}
\end{aligned}$$

for some $k_1 = \Theta(1)$ and large enough n. To derive inequality (c), we find a stochastic upper bound on T_{max} by considering n i.i.d. copies of the worker run-times with largest shift and smallest straggling parameters that are also $\Theta(1)$, and use the PDF of the maximum of n i.i.d. exponential random variables. As we later use in the proof of Theorem 5.3, one can similarly write for the shifted Weibull distribution:

$$\mathbb{E}[T_{\texttt{HCMM}}|\mathcal{E}_1] \Pr[\mathcal{E}_1] \leq \int_{\Theta(n)}^{\infty} t f_{\mathsf{max}}(t)\, dt$$

$$\leq \int_{\Theta(n)}^{\infty} nk_1k_2t^{k_2}e^{-k_1t^{k_2}} \left(1-e^{-k_1t^{k_2}}\right)^{n-1} dt$$
$$\leq \int_{\Theta(n)}^{\infty} nk_1k_2t^{k_2}e^{-k_1t^{k_2}}\, dt$$
$$\leq \int_{\Theta(n)}^{\infty} \frac{1}{t^2}\, dt = o(1), \tag{D.5}$$

for some constants k_1 and k_2. Therefore, using (D.2), (D.3) and (D.4) (or (D.5) for the shifted Weibull model) in (D.1) we have

$$\mathbb{E}[T_{\texttt{HCMM}}] \leq t^* + o(1).$$

Let $\boldsymbol{\ell}_{\mathsf{OPT}} = (\ell_{\mathsf{OPT},1}, \cdots, \ell_{\mathsf{OPT},n})$ denote the optimal load allocation corresponding to $\mathcal{P}_{\text{main}}$ in (5.3) and $X_{\mathsf{OPT}}(\cdot)$ represent the aggregate return under load allocation $\boldsymbol{\ell}_{\mathsf{OPT}}$. Now we prove the following lower bound on the average completion time of the optimum algorithm:

$$\mathbb{E}[T_{\mathsf{OPT}}] \geq t^* - o(1).$$

To this end, we show the following two inequalities,

$$\mathbb{E}[T_{\mathsf{OPT}}] \overset{(d)}{\geq} \tau - \delta_1 \overset{(e)}{\geq} t^* - \delta_2 - \delta_1,$$

where $\delta_1 = \Theta\left(\frac{\log n}{\sqrt{n}}\right)$, $\delta_2 = \Theta\left(\frac{\log n}{\sqrt{n}}\right)$ and τ is the solution to $\mathbb{E}[X_{\mathsf{OPT}}(\tau)] = r$. We have

$$\begin{aligned} r - \mathbb{E}[X_{\mathsf{OPT}}(\tau-\delta_1)] &= \sum_{i=1}^{n} \ell_{\mathsf{OPT},i}\big(\Pr[T_i < \tau] - \Pr[T_i < \tau - \delta_1]\big) \\ &= \sum_{i=1}^{n} \ell_{\mathsf{OPT},i}\left(\frac{d}{d\tau}\Pr[T_i<\tau]\delta_1 + \mathcal{O}\left(\delta_1^2\right)\right) \\ &= \Theta(n\delta_1) + \mathcal{O}\left(n\delta_1^2\right) = \Theta(n\delta_1), \end{aligned}$$

where we used the fact that $\ell_{\mathsf{OPT},i} = \Theta(1)$[1]. By McDiarmid's inequality (see Appendix for its description), we have

$$\begin{aligned}\Pr[X_{\mathsf{OPT}}(\tau-\delta_1) \geq r] &= \Pr[X_{\mathsf{OPT}}(\tau-\delta_1) - \mathbb{E}[X_{\mathsf{OPT}}(\tau-\delta_1)] \geq r - \mathbb{E}[X_{\mathsf{OPT}}(\tau-\delta_1)]] \\ &\leq \exp\left(-\frac{2\big(\mathbb{E}[X_{\mathsf{OPT}}(\tau-\delta_1)] - r\big)^2}{\sum_{i=1}^{n} \ell_{\mathsf{OPT},i}^2}\right) \\ &= e^{-\Theta\left(n\delta_1^2\right)} = o\left(\frac{1}{n}\right),\end{aligned}$$

which implies inequality (d). We proceed to prove (e) by showing the following two inequalities,

$$\tau \geq \tau^*, \tag{D.6}$$

$$\tau^* \geq t^* - \delta_2, \tag{D.7}$$

where τ^* is obtained in (??). Given the fact that `HCMM` maximizes the expected aggregate return, we have

$$\mathbb{E}[X^*(t)] \geq \mathbb{E}[X_{\mathsf{OPT}}(t)],$$

for every feasible t, which implies (D.6). Moreover, Lemma 5.1 proves (D.7). All in all, we have

$$t^* - o(1) \leq \mathbb{E}[T_{\mathsf{OPT}}] \leq \mathbb{E}[T_{\mathtt{HCMM}}] \leq t^* + o(1),$$

which yields $\lim_{n\to\infty} \mathbb{E}[T_{\mathtt{HCMM}}] = \lim_{n\to\infty} \mathbb{E}[T_{\mathsf{OPT}}]$ and the claim is concluded.

[1]We argue that the allocated loads in the optimum coded scheme are all $\Theta(1)$. Without loss of generality, suppose $\ell_{\mathsf{OPT},1} > \Theta(1)$ which implies $\lim_{n\to\infty} \Pr[T_1 < t] = 0$ for any $t = \Theta(1)$. We have already implemented `HCMM`, a (sub-)optimal algorithm achieving computation time $\tau^* = \Theta(1)$, therefore the optimal scheme should have a better finishing time $\tau \leq \Theta(1)$. Now assume the load of machine 1 is replaced by $\tilde{\ell}_{\mathsf{OPT},1} = \Theta(1)$. Clearly, for any time $t = \Theta(1)$, the aggregate return for the new set of loads is larger than the former one by any $\Theta(1)$ time, almost surely. This is in contradiction to optimality assumption.

D.2 Proof of Theorem 5.2

This section provides the proof of Theorem 5.2 by comparing the performance of `HCMM` to uncoded scheme. In an uncoded scheme, the redundancy factor is 1; thus, the master node has to wait for the results from all the worker nodes in order to complete the computation.

We start by characterizing the expected run-time of the best uncoded scheme. Particularly, we show that

$$\mathbb{E}[T_{\mathsf{UC}}] = \Theta\big(\log n\big),$$

where T_{UC} denotes the completion time of the optimum uncoded distributed matrix multiplication algorithm. To do so, we start by showing that

$$\mathbb{E}[T_{\mathsf{UC}}] \geq c \log n,$$

for a constant c independent of n. For a set of machines with parameters $\{(a_i, \mu_i)\}_{i=1}^{n}$, let $\tilde{a} = \min_i a_i$ and $\tilde{\mu} = \max_i \mu_i$. Now, consider another set of n machines in which every machine is replaced with a faster machine with parameters $(\tilde{a}, \tilde{\mu})$. Since the computation times of the new set of machines are i.i.d., one can show that the optimal load allocation for these machines is uniform, i.e.,

$$\widetilde{\ell}_i^* = \frac{r}{n},$$

for every machine $i \in [n]$. Let $\{\widetilde{T}_i\}_{i=1}^{n}$ represent the i.i.d. shifted exponential random variables denoting the execution times for the new set of machines where each machine is loaded by $\widetilde{\ell}_i^* = \frac{r}{n}$. Therefore, the CDF of the completion time of each new machine can be written as

$$\Pr[\widetilde{T}_i \leq t] = 1 - e^{-\frac{\tilde{\mu}}{\widetilde{\ell}_i^*}\left(t - \tilde{a}\widetilde{\ell}_i^*\right)} = 1 - e^{-\tilde{\mu}\frac{n}{r}\left(t - \tilde{a}\frac{r}{n}\right)},$$

for $t \geq \frac{\tilde{a}r}{n}$ and the expected computation time can be written as

$$\mathbb{E}[\widetilde{T}_i] = \frac{r}{n}\left(\tilde{a} + \frac{1}{\tilde{\mu}}\right),$$

for all $i \in [n]$. Since the master needs to wait for all of the machines to return their results, the total run-time is $\widetilde{T}_{\mathsf{UC}} = \max_{i \in [n]} \widetilde{T}_i$. Therefore,

$$\mathbb{E}[\widetilde{T}_{\mathsf{UC}}] = \mathbb{E}[\max_{i \in [n]} \widetilde{T}_i] = \frac{\tilde{a}r}{n} + \frac{rH_n}{n\tilde{\mu}}, \tag{D.8}$$

where $H_n = 1 + \frac{1}{2} + \frac{1}{3} + \cdots + \frac{1}{n}$ is the sum of the harmonic series. We can further bound (D.8) using the fact that

$$\frac{\tilde{a}r}{n} + \frac{rH_n}{n\tilde{\mu}} \geq \frac{\tilde{a}r}{n} + \frac{r}{n\tilde{\mu}} \log(n+1) \geq c \log n,$$

for a constant c independent of n, since $r = \Theta(n)$, $\tilde{a} = \Theta(1)$, and $\tilde{\mu} = \Theta(1)$ for all $i \in [n]$. All in all, we have the following lower bound on the optimal uncoded scheme:

$$\mathbb{E}[T_{\mathsf{UC}}] \geq \mathbb{E}[\widetilde{T}_{\mathsf{UC}}] \geq c \log n. \tag{D.9}$$

Now consider another set of n machines, where each machine is replaced with a slower one with parameters $(\hat{a}, \hat{\mu})$ for $\hat{a} = \max_i a_i$ and $\hat{\mu} = \min_i \mu_i$. By an argument similar to the one employed the lower bound, we can write

$$\mathbb{E}[T_{\mathsf{UC}}] \leq \frac{\hat{a}r}{n} + \frac{r}{n\hat{\mu}} H_n \leq C \log n, \tag{D.10}$$

for another constant C. From (D.9) and (D.10), one can conclude that

$$\mathbb{E}[T_{\mathsf{UC}}] = \Theta\big(\log n\big). \tag{D.11}$$

Further, by Theorem 5.1 and Lemma 5.1, we find that

$$\mathbb{E}[T_{\texttt{HCMM}}] = \Theta(1). \tag{D.12}$$

Comparing (D.11) to (D.12) demonstrates that `HCMM` outperforms the best uncoded scheme by a factor of $\Theta(\log n)$, i.e.,

$$\frac{\mathbb{E}[T_{\mathsf{UC}}]}{\mathbb{E}[T_{\texttt{HCMM}}]} = \Theta\big(\log n\big).$$

D.3 Proof of Lemma 5.1

Let us first state a useful inequality which we will use to prove Lemma 5.1.

McDiarmid's Inequality: Let $X_1, \cdots, X_n$ be independent random variables taking values in $\mathcal{X}$. Further, let the function $f : \mathcal{X}^n \to \mathbb{R}$ be L_i-Lipschitz for all $i \in [n]$, that is

$$|f(x_1, \cdots, x_i, \cdots, x_n) - f(x_1, \cdots, x_i', \cdots, x_n)| \leq L_i,$$

for any $x_1, \cdots, x_n, x_i' \in \mathcal{X}$ and $i \in [n]$. Then, for any $\epsilon > 0$,

$$\Pr\Big[f(X_1, \cdots, X_n) - \mathbb{E}[f(X_1, \cdots, X_n)] \geq \epsilon\Big] \leq \exp\left(-\frac{2\epsilon^2}{\sum_{i=1}^n L_i^2}\right),$$
$$\Pr\Big[\mathbb{E}[f(X_1, \cdots, X_n)] - f(X_1, \cdots, X_n) \geq \epsilon\Big] \leq \exp\left(-\frac{2\epsilon^2}{\sum_{i=1}^n L_i^2}\right).$$

For each i, the aggregate return at time t satisfies $X_i(t) \in \{0, \ell_i\}$. Therefore, we can use

McDiarmid's inequality as follows:

$$\Pr\left[X(t) - \mathbb{E}[X(t)] \geq \epsilon\right] \leq \exp\left(-\frac{2\epsilon^2}{\sum_{i=1}^{n} \ell_i^2}\right),$$
$$\Pr\left[\mathbb{E}[X(t)] - X(t) \geq \epsilon\right] \leq \exp\left(-\frac{2\epsilon^2}{\sum_{i=1}^{n} \ell_i^2}\right),$$

for any $\epsilon > 0$. Now, we proceed to the proof of Lemma 5.1.

Let $t = \tau^* + \delta$ for some $\delta = \Theta\left(\frac{\log n}{\sqrt{n}}\right)$ and $\epsilon = \delta^2$. The claim is that $\Pr\left[X^*(t) \leq r - \epsilon\right] = o\left(\frac{1}{n}\right)$. From McDiarmid's inequality, we have

$$\begin{aligned}
\Pr\left[X^*(t) \leq r - \epsilon\right] &\leq \exp\left(-\frac{2\left(\mathbb{E}[X^*(t)] - r + \epsilon\right)^2}{\sum_i \ell_i^{*2}(t)}\right) \\
&= \exp\left(-\frac{2\left(ts - r + \epsilon\right)^2}{\sum_i \ell_i^{*2}(t)}\right) \\
&= \exp\left(-\frac{2\delta^2 s^2 + 2\delta^4 + 4\delta^3 s}{\left((\frac{r}{s})^2 + \delta^2 + 2\delta\frac{r}{s}\right)\sum_i \lambda_i^2}\right) \\
&\overset{(g)}{=} e^{-\Theta\left(n\delta^2\right)} = o\left(\frac{1}{n}\right).
\end{aligned}$$

In above, equality (g) follows from the fact that $r = \Theta(n)$, $s = \Theta(n)$, $\lambda_i = \Theta(1)$, $\delta = \Theta\left(\frac{\log n}{\sqrt{n}}\right)$, and therefore $\sum_i \lambda_i^2 = \Theta(n)$ and $s^2 = \Theta(n^2)$. Moreover, if $t^* < \tau^*$, with a positive probability there are less than r equations at the master node by time t^* which is a contradiction. Therefore,

$$\tau^* \leq t^* \leq \tau^* + \delta.$$

D.4 Proof of Lemma 5.3

We first argue that if the budget-constrained problem defined in $\mathcal{P}_{\text{main-constrained}}$ is feasible, then HCMM determines the asymptotically optimal load allocation. Consider a set of N machines and assume that M of them are assigned non-zero loads in the optimal budget-constrained scheme. Now, one can run `HCMM` load allocation over the set of these M machines and according to asymptotic optimality results, `HCMM` asymptotically attains the optimal run-time while satisfying the budget constraint.

Now assume that n_k number of type $k \in [K]$ machine is used. Then, by assigning the loads obtained from `HCMM` and the result of Theorem 5.1, the induced expected cost (for large number of machines) can be written as

$$\begin{aligned}
\text{cost}\big(\texttt{HCMM}(n_1, \cdots, n_K)\big) &= \tau^* \sum_{k=1}^{K} n_k c_k \\
&= \frac{r}{s} \sum_{k=1}^{K} n_k c_k \\
&= \frac{r}{\sum_{k=1}^{K} \frac{n_k \mu_k}{1+\mu_k \lambda_k}} \sum_{k=1}^{K} n_k \kappa \mu_k^{\gamma} \\
&= \kappa r x_\xi \frac{\sum_{k=1}^{K} n_k \mu_k^{\gamma}}{\sum_{k=1}^{K} n_k \mu_k}, \qquad \text{(D.13)}
\end{aligned}$$

where $x_\xi = 1 + \mu_k \lambda_k$ is the solution to the equation $e^{x_\xi - \xi - 1} = x_\xi$ for all machine type $k \in [K]$. In another scenario, assume that we remove one machine of type K (the fastest machine type) and run `HCMM` accordingly, i.e. n_k of type $k \in [K-1]$ and $n_K - 1$ of type K. The expected cost of this scenario can be written as follows:

$$\begin{aligned}
\text{cost}\big(\texttt{HCMM}(n_1, \cdots, n_K - 1)\big) &= \kappa r x_\xi \frac{\sum_{k=1}^{K-1} n_k \mu_k^{\gamma} + (n_K - 1)\mu_K^{\gamma}}{\sum_{k=1}^{K-1} n_k \mu_k + (n_K - 1)\mu_K} \\
&\overset{(f)}{\leq} \kappa r x_\xi \frac{\sum_{k=1}^{K} n_k \mu_k^{\gamma}}{\sum_{k=1}^{K} n_k \mu_k} \\
&= \text{cost}\big(\texttt{HCMM}(n_1, \cdots, n_K)\big), \qquad \text{(D.14)}
\end{aligned}$$

where inequality (f) can be easily verified given that $\mu_1 \leq \cdots \leq \mu_K$. We can iteratively apply the same argument and conclude that the minimum expected cost is achieved when only the slowest machines are used, that is

$$C_{\min} \coloneqq \mathrm{cost}\big(\texttt{HCMM}(n_1, 0, \cdots, 0)\big) = \kappa r x_\xi \mu_1^{\gamma-1}, \tag{D.15}$$

for any $1 \leq n_1 \leq N_1$. Similar to (D.14), one can show that reducing the number of participating slowest machines increases the induced expected cost of `HCMM`, that is

$$\mathrm{cost}\big(\texttt{HCMM}(n_1 - 1, \cdots, n_K)\big) \geq \mathrm{cost}\big(\texttt{HCMM}(n_1, \cdots, n_K)\big). \tag{D.16}$$

Therefore, applying (D.16) iteratively shows that the maximum expected cost occurs when only the fastest machines are employed, that is

$$C_{\max} \coloneqq \mathrm{cost}\big(\texttt{HCMM}(0, \cdots, 0, n_K)\big) = \kappa r x_\xi \mu_K^{\gamma-1},$$

for any $1 \leq n_K \leq N_K$.

Appendix E

Supplements to Chapter 6

E.1 Pseudo-code for Computation Allocation Subroutine

Algorithm E.1: Computation Allocation (`CompAlloc`)

```
Input: dataset $\mathcal{D}$, $n$ workers, straggler toleration $s$, computation matrix
 $\mathbf{B} = [\mathbf{b}_1; \cdots; \mathbf{b}_n] \in \mathbb{R}^{n \times k}$
Output: data set allocation $\{\mathcal{D}_{(1)}, \cdots, \mathcal{D}_{(n)}\}$ for $n$ workers
Procedure CompAlloc($\mathcal{D}, \mathbf{B}$)
    uniformly partition $\mathcal{D} = \cup_{\kappa=1}^{k} \mathcal{D}_\kappa$
    for worker $i \leftarrow 1$ to $n$ do
        $\mathcal{D}_{(i)} \leftarrow \cup_{\kappa=1}^{k} b_{i\kappa} \mathcal{D}_\kappa$        % $\mathcal{D}_{(i)}$ is assigned to worker $W_i$
    end
end
```

Algorithm E.2: `CodedReduce`

```
Input: dataset $\mathcal{D}$, $(n, L)$–regular tree $T$, straggler toleration $s$ (per parent),
 model $\mathbf{w}^{(t)}$
Output: gradient $\mathbf{g}_{\mathcal{D}} = \sum_{\mathbf{x}\in\mathcal{D}} \nabla\ell(\mathbf{w}^{(t)}; \mathbf{x})$ aggregated at the master
Procedure CR.Allocate
    GC generates $\mathbf{B}$ specified by $n, s$ for $l \leftarrow 1$ to $L$ do
        for $i \leftarrow 1$ to $n^{l-1}$ do
            $\{\mathcal{D}^{T(l,n(i-1)+1)}, \cdots, \mathcal{D}^{T(l,ni)}\} = \texttt{CompAlloc}(\mathcal{D}_{T(l-1,i)}, \mathbf{B})$
        end
        for $i \leftarrow 1$ to $n^l$ do
            pick $r_{\texttt{CR}} \cdot d$ data points of $\mathcal{D}^{T(l,i)}$ as $\mathcal{D}(l, i)$
            $\mathcal{D}_{T(l,i)} \leftarrow \mathcal{D}^{T(l,i)} \setminus \mathcal{D}(l, i)$
        end
    end
end
Procedure CR.Execute
    GC generates $\mathbf{A}$ from $\mathbf{B}$
    all the workers compute their local partial gradients $\mathbf{g}_{\mathcal{D}(l,i)}$
    for $l \leftarrow L - 1$ to $1$ do
        for $i \leftarrow 1$ to $n^l$ do
            worker nodes $(l, i)$:
              receives $[\mathbf{m}_{(l+1,n(i-1)+1)}; \cdots; \mathbf{m}_{(l+1,ni)}]$ from its children
              uploads $\mathbf{m}_{(l,i)} = \mathbf{a}_{f(l,i)}[\mathbf{m}_{(l+1,n(i-1)+1)}; \cdots; \mathbf{m}_{(l+1,ni)}] + \mathbf{g}_{\mathcal{D}(l,i)}$ to its
             parent
        end
    end
    master node:
      receives $[\mathbf{m}_{(1,1)}; \cdots; \mathbf{m}_{(l,n)}]$ from its children
      recovers $\mathbf{g} = \mathbf{a}_{f(0,1)}[\mathbf{m}_{(1,1)}; \cdots; \mathbf{m}_{(1,n)}]$
end
```

E.2 Pseudo-code for `CodedReduce` Scheme

E.3 Proof of Theorem 6.1

Achievability: According to the data allocation described in Algorithm E.2, to be robust to any s straggling children of the master, the data set $\mathcal{D}$ is redundantly assigned to sub-trees $T(1, 1), \cdots, T(1, n)$ such that each data point is placed in $s + 1$ sub-trees, which yields

$$|\mathcal{D}^{T(1,i)}| = \left(\frac{s+1}{n}\right) d, \quad \forall i \in [n]. \tag{E.1}$$

Then, nodes in layer $l = 1$ pick $r_{\mathtt{CR}} d$ data points as their corresponding data sets and similarly distribute the remaining among their children which together with (E.1) yields

$$\begin{aligned} |\mathcal{D}^{T(2,i)}| &= \left(\frac{s+1}{n}\right)\left(\left(\frac{s+1}{n}\right) d - r_{\mathtt{CR}} d\right) \\ &= \left(\frac{s+1}{n}\right)\left(\left(\frac{s+1}{n}\right) - r_{\mathtt{CR}}\right) d, \quad \forall i \in [n^2]. \end{aligned}$$

By the same argument for each layer, we have

$$|\mathcal{D}^{T(L,i)}| = \left(\frac{s+1}{n}\right)\left(\left(\frac{s+1}{n}\right)^{L-1} - \left(\frac{s+1}{n}\right)^{L-2} r_{\mathtt{CR}} - \cdots - \left(\frac{s+1}{n}\right) r_{\mathtt{CR}} - r_{\mathtt{CR}}\right) d, \tag{E.2}$$

for all $i \in [n^L]$. Putting (E.2) together with $|\mathcal{D}^{T(L,i)}| = r_{\mathtt{CR}} d$ yields

$$r_{\mathtt{CR}} = \frac{1}{\left(\frac{n}{s+1}\right) + \cdots + \left(\frac{n}{s+1}\right)^L}.$$

Optimality: In an α–resilient scheme, the master node is able to recover from any $s = \alpha n$ straggling sub-trees $T(1,1), \cdots, T(1,n)$. Therefore, each data point has to be placed in at least $s+1$ of such sub-trees, which yields

$$|\mathcal{D}^{T(1,1)}| + \cdots + |\mathcal{D}^{T(1,n)}| \geq (s+1)d, \tag{E.3}$$

where the equality is achieved only if each data point is assigned to only $s+1$ sub-trees. Hence, we can assume the optimal scheme satisfies (E.3) with equality. Moving to the second layer, the following claim bounds the required redundancy assigned to sub-trees $T(2,1), \cdots, T(2,n)$. Similar claim holds for any other group of the siblings in this layer.

Claim E.1 *The following inequality holds:*

$$|\mathcal{D}^{T(2,1)}| + \cdots + |\mathcal{D}^{T(2,n)}| \geq (s+1)\left(|\mathcal{D}^{T(1,1)}| - rd\right).$$

Proof: [Proof of Claim E.1] First, note that $|\mathcal{D}^{T(1,1)} \setminus \mathcal{D}(1,1)| \geq |\mathcal{D}^{T(1,1)}| - rd$. If the claim does not hold, then there exists data point $\mathbf{x} \in \mathcal{D}^{T(1,1)} \setminus \mathcal{D}(1,1)$ such that $\mathbf{x}$ is placed in at most s sub-trees rooting in the node $(1,1)$, e.g. $T(2,1), \cdots, T(2,s)$. Note that besides sub-tree $T(1,1)$, $\mathbf{x}$ is placed in only s more sub-trees, e.g. $T(1,2), \cdots, T(1,s+1)$. Now consider a straggling pattern where $T(1,2), \cdots, T(1,s+1)$ and $T(2,1), \cdots, T(2,s)$ fail to return their results. Therefore, $\mathbf{x}$ is missed at the master and fails the aggregation recovery. ■

By the same logic used in the above proof, Claim E.1 holds for any parent node and its children, i.e. for any layer $l \in [L]$ and $i \in [n^{l-1}]$,

$$|\mathcal{D}^{T(l,n(i-1)+1)}| + \cdots + |\mathcal{D}^{T(l,ni)}| \geq (s+1)\left(|\mathcal{D}^{T(l-1,i)}| - rd\right). \tag{E.4}$$

Specifically applying (E.4) to layer L and noting that $|\mathcal{D}^{T(L,i)}| = |\mathcal{D}(L,i)| = rd$ for any i, we conclude that

$$rd\left(\left(\frac{n}{s+1}\right) + 1\right) \geq |\mathcal{D}^{T(L-1,1)}|.$$

We can then use the above inequality and furthermore write (E.4) for layer $L-1$ which results in

$$rd\left(\left(\frac{n}{s+1}\right)^2 + \left(\frac{n}{s+1}\right) + 1\right) \geq |\mathcal{D}^{T(L-2,1)}|.$$

By deriving the above inequality recursively up to the master node, we get

$$rd\left(\left(\frac{n}{s+1}\right)^{L-1}+\cdots+\left(\frac{n}{s+1}\right)+1\right)\geq\frac{s+1}{n}d,$$

which concludes the optimality in Theorem 6.1.

E.4 Proof of Theorem 6.2

Let us begin with the lower bound

$$\mathbb{E}\left[T_{\texttt{CR}}\right]\geq\frac{r_{\texttt{CR}}d}{\mu}\log\left(\frac{1}{\alpha}\right)+ar_{\texttt{CR}}d+\big(n(1-\alpha)-o(n)+L-1\big)\left(\big(1-o(1)\big)\,t_c+o(1)\right.$$

Consider the group of siblings[1] placed in layer L whose result reaches their parent nodes first. Let $\widehat{T}$ denote the time at which the parent of such group is able to recover the partial gradient from its fastest children's computations, i.e. fastest $n-s$ of them. We also denote by $T_1,\cdots,T_n$ the partial gradient computation times for the siblings. According to the random computation time model described in the paper and the computation load of CR, each T_i is shifted exponential with the shift parameter $ad_i=ar_{\texttt{CR}}d$ and the rate parameter $\frac{\mu}{d_i}=\frac{\mu}{r_{\texttt{CR}}d}$. Since CR is robust to any s stragglers per parent, the partial gradient computation time for any group of siblings is $T_{(n-s)}$, i.e. the $(n-s)$'th order statistics of $\{T_1,\cdots,T_n\}$. In [128], authors consider coded computation scenarios in a master-worker topology where the master only needs to wait for results of the first α fraction of the workers. However, as in the scenario here, the limited bandwidth at the master only allows for one transmission at the time. From the latency analysis in [128], we have the following.

[1]A group of siblings refers to n nodes with the same parent.

Lemma E.1 (Theorem 2, [128]) *With probability* $1-o(1)$, *we have*

$$\widehat{T} \geq T_{(n-s)} + \big(n\,(1-\alpha) - o(n)\big)\, t_c. \tag{E.5}$$

Now, conditioned on the event in (E.5) we can write

$$\begin{aligned}
\mathbb{E}\left[T_{\mathtt{CR}}\right] &\geq \Big(\mathbb{E}\left[T_{(n-s)}\right] + \big(n\,(1-\alpha) - o(n)\big)\, t_c\Big)\, \big(1-o(1)\big) + \Big(\mathbb{E}\left[T_{(n-s)}\right] + Lt_c\Big)\, o(1) \\
&\geq \mathbb{E}\left[T_{(n-s)}\right] + \big(n(1-\alpha) - o(n) + L - 1\big)\, \big(1-o(1)\big)\, t_c \\
&\overset{(a)}{\geq} \frac{r_{\mathtt{CR}} d}{\mu} \log\left(\frac{1}{\alpha}\right) + a r_{\mathtt{CR}} d + \big(n(1-\alpha) - o(n) + L - 1\big)\, \big(1-o(1)\big)\, t_c + o(1),
\end{aligned}$$

where inequality (a) uses the fact that $\mathbb{E}\left[T_{(n-s)}\right] = \frac{r_{\mathtt{CR}} d}{\mu}\left(H_n - H_s\right) + a r_{\mathtt{CR}} d$ and $\log(i) < H_i = 1 + \frac{1}{2} + \cdots + \frac{1}{i} < \log(i+1)$ for any positive integer i.

To derive upper bound on $\mathbb{E}[T_{\mathtt{CR}}]$, that is

$$\mathbb{E}\left[T_{\mathtt{CR}}\right] \leq \frac{r_{\mathtt{CR}} d}{\mu} \log\left(\frac{1}{\alpha}\right) + a r_{\mathtt{CR}} d + n\, \big(1-o(1)\big)\, L t_c + o(1),$$

we prove the following concentration inequality on the computation time for any group of siblings.

Lemma E.2 *Let* $T_1, \cdots, T_n$ *denote i.i.d. exponential random variables with constant rate* $\lambda = \Theta(1)$. *For* $\varepsilon = \Theta\left(\frac{1}{n^{1/4}}\right)$ *and constant* $\alpha = \frac{s}{n}$, *we have the following concentration bound for the order statistics* $T_{(n-s)}$:

$$\Pr\Big[T_{(n-s)} - \mathbb{E}\left[T_{(n-s)}\right] \geq \varepsilon\Big] \leq e^{-\Theta\left(\sqrt{n}\right)}. \tag{E.6}$$

Proof: [Proof of Lemma E.2] Given i.i.d. exponentials $T_1, \cdots, T_n \sim \exp(\lambda)$, we can write the successive differences of order statistics as independent exponentials. That is, we have

$$
\begin{aligned}
T_{(1)} &= E_1 \sim \exp\left(\frac{\lambda}{n}\right), \\
T_{(2)} - T_{(1)} &= E_2 \sim \exp\left(\frac{\lambda}{n-1}\right), \\
&\vdots \\
T_{(n-s)} - T_{(n-s-1)} &= E_{n-s} \sim \exp\left(\frac{\lambda}{s+1}\right), \\
&\vdots \\
T_{(n)} - T_{(n-1)} &= E_n \sim \exp\left(\lambda\right),
\end{aligned}
$$

where E_i's are independent. Thus, $T_{(n-s)} = \sum_{i=1}^{n-s} E_i$. We have the following for independent exponentials E_i's and $\lambda = \Theta(1)$:

$$
\begin{aligned}
\mathbb{E}\left[|E_i|^k\right] &= \mathbb{E}\left[E_i^k\right] \\
&= \left(\frac{\lambda}{n-i+1}\right)^k k! \\
&= \frac{1}{2}\mathbb{E}\left[E_i^2\right]\left(\frac{\lambda}{n-i+1}\right)^{k-2} k! \\
&\leq \frac{1}{2}\mathbb{E}\left[E_i^2\right] B^{k-2} k!,
\end{aligned}
$$

for $B = \frac{\lambda}{s} = \frac{\lambda}{\alpha n} = \Theta\left(\frac{1}{n}\right)$. Moreover,

$$
\begin{aligned}
\sum_{i=1}^{n-s} \mathbb{E}\left[E_i^2\right] &= 2\lambda^2\left(\frac{1}{n^2} + \cdots + \frac{1}{(s+1)^2}\right) \\
&\leq 2\lambda^2 \cdot \frac{n-s}{s^2} \\
&= \frac{2\lambda^2(1-\alpha)}{\alpha^2} \cdot \frac{1}{n} \\
&= \Theta\left(\frac{1}{n}\right).
\end{aligned}
$$

According to Bersterin's Lemma (See Lemma E.3), for $\varepsilon = \Theta\left(\frac{1}{n^{1/4}}\right)$ we have

$$\begin{aligned}
\Pr\left[T_{(n-s)} - \mathbb{E}\left[T_{(n-s)}\right] \geq \varepsilon\right] &\leq \exp\left(-\frac{\varepsilon^2}{2\left(\sum_{i=1}^{n-s}\mathbb{E}\left[E_i^2\right] + \varepsilon B\right)}\right) \\
&\leq \exp\left(-\frac{\varepsilon^2}{2\left(\Theta\left(\frac{1}{n}\right) + \varepsilon\Theta\left(\frac{1}{n}\right)\right)}\right) \\
&= e^{-\Theta\left(\sqrt{n}\right)}.
\end{aligned}$$

■

As described in Section 6.3.1, in the proposed CR scheme all the worker nodes start their assigned partial gradient computations simultaneously; each parent waits for enough number of children to receive their results; combines with its partial computation and sends the result up to its parent. To upper bound the total aggregation time $T_{\texttt{CR}}$, one can separate all the local computations from the communications. Let T_{comp} denote the time at which enough number of workers have executed their local gradient computations and no more local computation is needed for the final gradient recovery. Moreover, we assume that all the communications from children to parent are pipe-lined. Hence, we have $\mathbb{E}\left[T_{\texttt{CR}}\right] \leq \mathbb{E}\left[T_{\text{comp}}\right] + L(n-s)t_c$. To bound the computation time T_{comp}, consider the following event which keeps the local computation times for *all* the N/n groups of siblings concentrated below their average deviated by $\varepsilon = \Theta\left(\frac{1}{n^{1/4}}\right)$:

$$\mathcal{E}_1 := \left\{T_{(n-s)}^{gr} \leq \mathbb{E}\left[T_{(n-s)}^{gr}\right] + \varepsilon \text{ for all the } N/n \text{ groups } gr\right\},$$

where a group gr is a collection of n children with the same parent, i.e. there are N/n groups in the (n, L)–regular tree. For a group gr, $\{T_1^{gr}, \cdots, T_n^{gr}\}$ denote the random run-times of the nodes in the group and $T^{gr}_{(n-s)}$ represents its $(n-s)$'th order statistics. Clearly,

$$\mathbb{E}\left[T_{\text{comp}}|\mathcal{E}_1\right] \leq \mathbb{E}\left[T_{(n-s)}\right] + o(1). \tag{E.7}$$

Now let $\widetilde{T}$ denote the computation time corresponding to the slowest group of siblings, i.e.

$$\widetilde{T} \coloneqq \max_{\text{over all } N/n \text{ groups } gr} T^{gr}_{(n-s)}.$$

Consider the following event:

$$\mathcal{E}_2 \coloneqq \left\{\widetilde{T} > \Theta(\log n)\right\}.$$

We can write

$$\mathbb{E}\left[T_{\text{comp}}|\mathcal{E}_1^c \cap \mathcal{E}_2^c\right] \leq \Theta(\log n), \tag{E.8}$$

and

$$\begin{aligned}
\mathbb{E}\left[T_{\text{comp}}|\mathcal{E}_1^c \cap \mathcal{E}_2\right] \Pr\left[\mathcal{E}_2\right] &\leq \mathbb{E}\left[\widetilde{T}|\mathcal{E}_1^c \cap \mathcal{E}_2\right] \Pr\left[\mathcal{E}_2\right] \\
&= \mathbb{E}\left[\widetilde{T}|\widetilde{T} \leq \Theta(\log n)\right] \Pr\left[\widetilde{T} \leq \Theta(\log n)\right] \\
&\leq \mathbb{E}\left[\widetilde{T}\right] \\
&\leq \mathbb{E}\left[T_{\max}\right] \\
&= \frac{r_{\texttt{CR}} d}{\mu} H_N + a r_{\texttt{CR}} d \\
&= \Theta\left(\log N\right)
\end{aligned}$$

$$= L\Theta\left(\log n\right). \tag{E.9}$$

In the above derivation, $T_{\max}$ denotes the largest computation time over all the N nodes. Putting (E.8) and (E.9) together, we can write

$$\begin{aligned}\mathbb{E}\left[T_{\text{comp}}|\mathcal{E}_1^c\right] &= \mathbb{E}\left[T_{\text{comp}}|\mathcal{E}_1^c \cap \mathcal{E}_2\right]\Pr\left[\mathcal{E}_2\right] + \mathbb{E}\left[T_{\text{comp}}|\mathcal{E}_1^c \cap \mathcal{E}_2^c\right]\Pr\left[\mathcal{E}_2^c\right] \\ &\leq \Theta\left(\log n\right). \end{aligned} \tag{E.10}$$

Moreover, using union bound on the N/n groups of workers, we derive the following inequality.

$$\begin{aligned}\Pr\left[\mathcal{E}_1^c\right] &\leq \frac{N}{n}\Pr\left[T_{(n-s)} \geq \mathbb{E}\left[T_{(n-s)}\right] + \varepsilon\right] \\ &\leq \Theta\left(n^{L-1}\right)e^{-\Theta\left(\sqrt{n}\right)}. \end{aligned} \tag{E.11}$$

Putting (E.7), (E.10) and (E.11) together, we have

$$\begin{aligned}\mathbb{E}\left[T_{\text{comp}}\right] &= \mathbb{E}\left[T_{\text{comp}}|\mathcal{E}_1\right]\Pr\left[\mathcal{E}_1\right] + \mathbb{E}\left[T_{\text{comp}}|\mathcal{E}_1^c\right]\Pr\left[\mathcal{E}_1^c\right] \\ &\leq \mathbb{E}\left[T_{(n-s)}\right] + \varepsilon + \Theta\left(\log n\right)\Theta\left(n^{L-1}\right)e^{-\Theta\left(\sqrt{n}\right)} \\ &= \mathbb{E}\left[T_{(n-s)}\right] + o(1) \\ &= \frac{r_{\texttt{CR}}d}{\mu}\left(H_n - H_s\right) + ar_{\texttt{CR}}d + o(1).\end{aligned}$$

Therefore,

$$\begin{aligned}\mathbb{E}\left[T_{\texttt{CR}}\right] &\leq \mathbb{E}\left[T_{\text{comp}}\right] + Ln(1-\alpha)t_c \\ &= \frac{r_{\texttt{CR}}d}{\mu}\left(H_n - H_s\right) + ar_{\texttt{CR}}d + Ln(1-\alpha)t_c + o(1)\end{aligned}$$

$$\leq \frac{r_{\mathtt{CR}} d}{\mu} \log\left(\frac{1}{\alpha}\right) + a r_{\mathtt{CR}} d + n \big(1 - o(1)\big) L t_c + o(1),$$

which completes the proof.

Lemma E.3 (Bernstein's Inequality) *Suppose $E_1, \cdots, E_m$ are independent random variables such that*

$$\mathbb{E}\left[|E_i|^k\right] \leq \frac{1}{2}\mathbb{E}\left[E_i^2\right] B^{k-2} k!,$$

for some $B > 0$ and every $i = 1, \cdots, m$, $k \geq 2$. Then, for $\varepsilon > 0$,

$$\Pr\left[\sum_{i=1}^{m} E_i - \sum_{i=1}^{m} \mathbb{E}\left[E_i\right] \geq \varepsilon\right] \leq \exp\left(-\frac{\varepsilon^2}{2\left(\sum_{i=1}^{m} \mathbb{E}\left[E_i^2\right] + \varepsilon B\right)}\right).$$

Appendix F

Supplements to Chapter 7

F.1 Proof of Lemma 7.1

Before proving Lemma 7.1, we first present the following lemma that will be used in our proof.

Lemma F.1 *For random variables $\{P_i\}_{i=1}^r$ defined in (7.8), their moment generating functions for $s' > 0$ can be bounded by*

$$\mathbb{E}\left[e^{s'P_i}\right] \leq (pe^{2s'} + 1 - p)^{\tilde{g}/2}.$$

Proof: Consider a generic random variable of the form (7.8)

$$P = \sum_{j=1}^{\tilde{g}} E_j,$$

where E_j's are Bern(p) and possibly dependent. However, although E_j's may not be all independent, but dependency is restricted to pairs of E_j's. In other words, for all $1 \leq j \leq \tilde{g}$, E_j is either independent of all $E_{[\tilde{g}]\setminus\{j\}}$, or is equal to E_ℓ for some $\ell \in [\tilde{g}] \setminus \{j\}$

and independent of all $E_{[\tilde{g}]\setminus\{j,\ell\}}$. By merging dependent pairs, we can write

$$P = \sum_{j=1}^{\tilde{g}-J} F_j,$$

where

(i) F_j's are independent,

(ii) $\tilde{g} - 2J$ of F_j's are Bern(p),

(iii) J of F_j's are $2 \times$ Bern(p),

for some integer $0 \leq J \leq \lfloor \frac{\tilde{g}}{2} \rfloor$. Now, we can bound the moment generating function of P. For $s' > 0$,

$$\begin{aligned}
\mathbb{E}\Big[e^{s'P}\Big] &= \mathbb{E}\Big[e^{s' \sum_{j=1}^{J} F_j}\Big] \\
&= \prod_{j=1}^{\tilde{g}-J} \mathbb{E}\big[e^{s'F_j}\big] \\
&= \big(pe^{s'} + 1 - p\big)^{\tilde{g}-2J} \big(pe^{2s'} + 1 - p\big)^{J} \\
&= \Big[\big(pe^{s'} + 1 - p\big)^2\Big]^{\tilde{g}/2-J} \big(pe^{2s'} + 1 - p\big)^{J} \\
&\overset{(a)}{\leq} \big(pe^{2s'} + 1 - p\big)^{\tilde{g}/2-J} \big(pe^{2s'} + 1 - p\big)^{J} \\
&= \big(pe^{2s'} + 1 - p\big)^{\tilde{g}/2},
\end{aligned}$$

where inequality (a) is obtained using Lemma F.2 (proof available in Appendix F.6). ■

We now complete the proof of Lemma 7.1. For any $s' > 0$, we can write

$$\begin{aligned}
e^{s'\mathbb{E}[Q]} &\leq \mathbb{E}\big[e^{s'Q}\big] \\
&= \mathbb{E}\Big[\max_{i=1,\cdots,r} e^{s'P_i}\Big]
\end{aligned}$$

$$\begin{aligned}
&\le \mathbb{E}\Big[\sum_{i=1}^{r} e^{s'P_i}\Big] \\
&= \sum_{i=1}^{r} \mathbb{E}\big[e^{s'P_i}\big] \\
&\le r(pe^{2s'} + 1 - p)^{\tilde{g}/2},
\end{aligned}$$

where the last inequality follows from Lemma F.1. Taking logarithm from both sides yields

$$\mathbb{E}[Q] \le \frac{1}{s'}\log(r) + \frac{\tilde{g}}{2s'}\log(pe^{2s'} + 1 - p). \tag{F.1}$$

Let us substitute $s = 2s'$ in (F.1). Then,

$$\mathbb{E}[Q] \le \frac{1}{s}\log(r^2) + \frac{\tilde{g}}{s}\log(pe^{s} + 1 - p), \tag{F.2}$$

for any $s > 0$. Let $\bar{p} = 1 - p$ and pick

$$s_* = 2\sqrt{\frac{\log(r)}{\tilde{g}p\bar{p}}}.$$

We proceed with evaluation of the right hand side (RHS) of (F.2) at $s = s_*$. We first recall the following Taylor series

$$\begin{aligned}
\log(1+x) &= x - \frac{x^2}{2} + \frac{x^3}{3} - \cdots, \quad \text{for } x \in (-1, 1], \\
e^x &= 1 + x + \frac{x^2}{2} + \frac{x^3}{3!} + \cdots, \qquad \text{for } x \in \mathbb{R}.
\end{aligned}$$

Let $x = p(e^{s_*} - 1)$. It is easy to check that for $p = \omega(\frac{1}{n^2})$, we have $x \to 0$ and $s_* \to 0$ as

$n \to \infty$. Therefore, for $n \to \infty$ we can write

$$\begin{aligned}
\log(pe^{s_*} + 1 - p) &= \log(x+1) \\
&= x - \frac{x^2}{2} + \frac{x^3}{3} - \cdots \\
&= p(e^{s_*} - 1) - \frac{p^2(e^{s_*}-1)^2}{2} + \frac{p^3(e^{s_*}-1)^3}{3} - \cdots \\
&= p\big(s_* + \frac{{s_*}^2}{2} + \frac{{s_*}^3}{3!} + \cdots\big) - \frac{p^2}{2}\big(s_* + \frac{{s_*}^2}{2} + \frac{{s_*}^3}{3!} + \cdots\big)^2 \\
&\quad + \frac{p^3}{3}\big(s_* + \frac{{s_*}^2}{2} + \frac{{s_*}^3}{3!} + \cdots\big)^3 - \cdots \\
&= ps_* + \frac{p\bar{p}}{2}s_*^2 + o(ps_*^2).
\end{aligned}$$

Putting everything together, we have

$$\begin{aligned}
\mathbb{E}[Q] &\leq \frac{1}{s_*}\log(r^2) + \frac{\tilde{g}}{s_*}\log(pe^{s_*} + 1 - p) \\
&= \frac{1}{s_*}\log(r^2) + \frac{\tilde{g}}{s_*}\big(ps_* + \frac{p\bar{p}}{2}s_*^2 + o(ps_*^2)\big) \\
&= \frac{1}{s_*}\log(r^2) + \tilde{g}p + \frac{\tilde{g}p\bar{p}}{2}s_* + o(\tilde{g}ps_*) \\
&= \tilde{g}p + 2\sqrt{\tilde{g}p\bar{p}\log(r)} + o\left(\sqrt{\tilde{g}p}\right).
\end{aligned}$$

Recall that $\tilde{g} = \frac{n^2}{K\binom{K}{r}}$ which is a deterministic function of n. Therefore, we choose $p = \omega(\frac{1}{n^2})$ to have $\tilde{g}p = \omega(1)$ and thus $\sqrt{\tilde{g}p\bar{p}\log(r)} = \Theta\left(\sqrt{\tilde{g}p}\right) = o\left(\tilde{g}p\right)$. Therefore, $\mathbb{E}[Q] \leq p\tilde{g} + o(p\tilde{g})$, as $n \to \infty$.

F.2 Proof of Claim E.1

(i) If $\mathcal{S} = \{k\}$, for any $k \in [K]$ and graph $\mathcal{G}$ we have $H(X_{\mathcal{S}}|Y^{\mathcal{G}}_{\mathcal{S}^c}) \geq 0$. Therefore,

$$\mathbb{E}_{\mathcal{G}}\Big[H(X_{\mathcal{S}}|Y^{\mathcal{G}}_{\mathcal{S}^c})\Big] \geq 0 = pT\sum_{j=1}^{1} a^{1,\mathcal{S}}_{\mathcal{M}} \frac{n}{K}\frac{1-1}{1}.$$

(ii) Assume that claim (7.10) holds for all subsets of size S_0. For any subset $\mathcal{S} \subseteq [K]$ of size S_0+1, the following steps hold:

$$\begin{aligned} H(X_{\mathcal{S}}|Y^{\mathcal{G}}_{\mathcal{S}^c}) &= \frac{1}{|S|}\sum_{k\in S} H(X_{\mathcal{S}}, X_k|Y^{\mathcal{G}}_{\mathcal{S}^c}) \\ &= \frac{1}{|S|}\sum_{k\in S}(H(X_{\mathcal{S}}|X_k, Y^{\mathcal{G}}_{\mathcal{S}^c}) + H(X_k|Y^{\mathcal{G}}_{\mathcal{S}^c})) \qquad \text{(F.3)} \\ &\geq \frac{1}{|S|}\sum_{k\in S} H(X_{\mathcal{S}}|X_k, Y^{\mathcal{G}}_{\mathcal{S}^c}) + \frac{1}{|S|}H(X_{\mathcal{S}}|Y^{\mathcal{G}}_{\mathcal{S}^c}). \qquad \text{(F.4)} \end{aligned}$$

where (F.4) follows from (F.3) using chain rule and conditional entropy relations. Simplifying (F.4) and using $|S|-1 = S_0$, we have the following:

$$H(X_{\mathcal{S}}|Y^{\mathcal{G}}_{\mathcal{S}^c}) \geq \frac{1}{S_0}\sum_{k\in\mathcal{S}} H(X_{\mathcal{S}}|V^{\mathcal{G}}_{:,\mathcal{M}_k}, Y^{\mathcal{G}}_{\mathcal{S}^c}). \qquad \text{(F.5)}$$

Moreover,

$$H(X_{\mathcal{S}}|V^{\mathcal{G}}_{:,\mathcal{M}_k}, Y^{\mathcal{G}}_{\mathcal{S}^c}) = H(V^{\mathcal{G}}_{\mathcal{R}_k,:}|V^{\mathcal{G}}_{:,\mathcal{M}_k}, Y^{\mathcal{G}}_{\mathcal{S}^c}) + H(X_{\mathcal{S}}|V^{\mathcal{G}}_{:,\mathcal{M}_k}, V^{\mathcal{G}}_{\mathcal{R}_k,:}, Y^{\mathcal{G}}_{\mathcal{S}^c}). \qquad \text{(F.6)}$$

We can lower bound expected value of the first RHS term in (F.6) as follows

$$\mathbb{E}_{\mathcal{G}}\Big[H(V^{\mathcal{G}}_{\mathcal{R}_k,:}|V^{\mathcal{G}}_{:,\mathcal{M}_k}, Y^{\mathcal{G}}_{\mathcal{S}^c})\Big] = \mathbb{E}_{\mathcal{G}}\left[\sum_{v\in\mathcal{R}_k} H(V^{\mathcal{G}}_{\{v\},:}|V^{\mathcal{G}}_{\{v\},\mathcal{M}_k\cup\mathcal{M}_{\mathcal{S}^c}})\right]$$

$$
\begin{aligned}
&= \mathbb{E}_{\mathcal{G}}\left[\sum_{v\in\mathcal{R}_k} |\mathcal{N}(v)| - |\mathcal{N}(v)\cap(\mathcal{M}_k\cup\mathcal{M}_{\mathcal{S}^c})|\right] \\
&= \frac{n}{K} pT \sum_{j=0}^{S_0} a_{\mathcal{M}}^{j,\mathcal{S}\backslash\{k\}} \\
&\geq \frac{n}{K} pT \sum_{j=1}^{S_0} a_{\mathcal{M}}^{j,\mathcal{S}\backslash\{k\}}. \qquad \text{(F.7)}
\end{aligned}
$$

Expected value of the second term in RHS of (F.6) can be lower bounded from the induction assumption:

$$
\begin{aligned}
\mathbb{E}_{\mathcal{G}}\Big[H(X_{\mathcal{S}}|V^{\mathcal{G}}_{:,\mathcal{M}_k}, V^{\mathcal{G}}_{\mathcal{R}_k,:}, Y^{\mathcal{G}}_{\mathcal{S}^c})\Big] &= \mathbb{E}_{\mathcal{G}}\Big[H(X_{\mathcal{S}\backslash\{k\}}|Y^{\mathcal{G}}_{\mathcal{S}\backslash\{k\}})\Big] \\
&\geq pT \sum_{j=1}^{S_0} a_{\mathcal{M}}^{j,\mathcal{S}\backslash\{k\}} \frac{n}{K}\frac{S_0-j}{j}. \qquad \text{(F.8)}
\end{aligned}
$$

Putting (F.5), (F.6), (F.7), and (F.8) together, we have

$$
\begin{aligned}
\mathbb{E}_{\mathcal{G}}\Big[H(X_{\mathcal{S}}|Y^{\mathcal{G}}_{\mathcal{S}^c})\Big] &\geq \frac{1}{S_0}\sum_{k\in\mathcal{S}} \mathbb{E}_{\mathcal{G}}\Big[H(X_{\mathcal{S}}|V^{\mathcal{G}}_{:,\mathcal{M}_k}, Y^{\mathcal{G}}_{\mathcal{S}^c})\Big] \\
&= \frac{1}{S_0}\sum_{k\in\mathcal{S}} \mathbb{E}_{\mathcal{G}}\Big[H(V^{\mathcal{G}}_{\mathcal{R}_k,:}|V^{\mathcal{G}}_{:,\mathcal{M}_k}, Y^{\mathcal{G}}_{\mathcal{S}^c})\Big] + \mathbb{E}_{\mathcal{G}}\Big[H(X_{\mathcal{S}}|V^{\mathcal{G}}_{:,\mathcal{M}_k}, V^{\mathcal{G}}_{\mathcal{R}_k,:}, Y^{\mathcal{G}}_{\mathcal{S}^c})\Big] \\
&\geq \frac{1}{S_0}\sum_{k\in\mathcal{S}} \left(\frac{n}{K} pT \sum_{i=1}^{S_0} a_{\mathcal{M}}^{i,\mathcal{S}\backslash\{k\}} + pT \sum_{j=1}^{S_0} a_{\mathcal{M}}^{j,\mathcal{S}\backslash\{k\}} \frac{n}{K}\frac{S_0-j}{j}\right) \\
&= pT \sum_{j=1}^{S_0} \frac{n}{K}\frac{1}{j} \sum_{k\in\mathcal{S}} a_{\mathcal{M}}^{j,\mathcal{S}\backslash\{k\}} \\
&= pT \sum_{j=1}^{S_0+1} a_{\mathcal{M}}^{j,\mathcal{S}} \frac{n}{K}\frac{S_0+1-j}{j}.
\end{aligned}
$$

(iii) Therefore, for any subset $\mathcal{S}\subseteq[K]$, claim (7.10) holds.

F.3 Achievability for the Random Bi-partite Model

In this Section, we specialize our proposed scheme in Section 7.4 for the random bi-partite model and prove the achievability of Theorem 7.3. Consider RB(n_1, n_2, q) graph $\mathcal{G} = (\mathcal{V}_1 \cup \mathcal{V}_2, \mathcal{E})$ with $n = n_1 + n_2$, $|\mathcal{V}_1| = n_1 = \Theta(n)$, and $|\mathcal{V}_2| = n_2 = \Theta(n)$ where $|n_1 - n_2| = o(n)$. The prior knowledge of the bi-partite structure of the graph implies that Reduction of vertices in $\mathcal{V}_1$ depends only on the Mappers in $\mathcal{V}_2$. Therefore, the two operations would better be assigned to the same set of servers. Inspired by that argument, we describe subgraph and Reduce allocations as follows. We divide the total K servers into two sets of $K_1 = \frac{n_1}{n}K$ and $K_2 = \frac{n_2}{n}K$ servers. Assume $n_1 \geq n_2$.

(I) Mappers in $\mathcal{V}_1$ and Reducers in $\mathcal{V}_2$ are distributedly allocated to K_1 servers according to the allocation scheme proposed in Section 7.4.1. Each of the K_1 servers Maps $n_1\frac{r}{K_1} = n\frac{r}{K}$ vertices (in $\mathcal{V}_1$) and Reduces $\frac{n_2}{K_1} = \frac{n_2}{n_1}\frac{n}{K}$ vertices (in $\mathcal{V}_2$). Note that although each server in K_1 is loaded at its capacity with $n\frac{r}{K}$ Mappers, these servers are assigned $\frac{n_2}{n_1}\frac{n}{K} \leq \frac{n}{K}$ Reducers which implies more Reducers can be assigned to these servers.

(II) Next we allocate the Mappers in $\mathcal{V}_2$ to the other set of K_2 servers similar to Mappers in $\mathcal{V}_1$. According to our pick for K_2 and the allocation scheme proposed in Section 7.4.1, each server in K_2 is assigned with $n_2\frac{r}{K_2} = n\frac{r}{K}$ vertices (in $\mathcal{V}_2$). To allocate the n_1 Reductions in $\mathcal{V}_1$ to the K_2 servers, we note that these servers can accommodate at most $K_2\frac{n}{K} = n_2$ Reductions which is less than n_1. To allocate all Reductions, we use the remaining Reduction space in the K_1 servers. More precisely, we first allocate n_2 out of the total n_1 Reductions in $\mathcal{V}_1$ to the K_2 servers.

(III) Finally, we allocate the remaining $n_1 - n_2$ vertices to the K_1 servers.

All in all, each of the K servers is now assigned with nr/K Mappers and n/K Reducers.

We denote this allocation by $\tilde{A} \in \mathcal{A}(r)$. Moreover, coded Shuffling applies the coded scheme proposed in Section 7.4.1 for Reducing functions in phases (I) and (II) separately. We also allow uncoded communications for enabling Reductions required in phase (III).

Now, we evaluate the communication load of each of the above phases. Let $\bar{L}_{\tilde{A}}^{\mathsf{C1}}$, $\bar{L}_{\tilde{A}}^{\mathsf{C2}}$ denote the average normalized communication loads for phases (I) and (II); and $\bar{L}_{\tilde{A}}^{\mathsf{UC3}}$ denote the average normalized communication load regarding phase (III). From the achievability result in Theorem 7.1, for $q = \omega(\frac{1}{n^2})$, we have

$$\bar{L}_{\tilde{A}}^{\mathsf{C1}} \leq \frac{1}{r} q \frac{n_1 n_2}{n^2} \left(1 - \frac{r}{K_1}\right) + o(q),$$

and

$$\bar{L}_{\tilde{A}}^{\mathsf{C2}} \leq \frac{1}{r} q \frac{n_2^2}{n^2} \left(1 - \frac{r}{K_2}\right) + o(q).$$

As mentioned before, Reduction of the remaining $n_1 - n_2$ vertices in phase (III) is carried out uncoded, which induces the average normalized communication load as follows:

$$\bar{L}_{\tilde{A}}^{\mathsf{UC3}} = q \frac{n_2(n_1 - n_2)}{n^2}.$$

Putting all together, the proposed achievable scheme has the total average normalized communication load $\bar{L}_{\tilde{A}}$ as follows:

$$\begin{aligned}\bar{L}_{\tilde{A}} &= \bar{L}_{\tilde{A}}^{\mathsf{C1}} + \bar{L}_{\tilde{A}}^{\mathsf{C2}} + \bar{L}_{\tilde{A}}^{\mathsf{UC3}} \\ &\leq \frac{1}{r} q \frac{n_1 n_2}{n^2} \left(1 - \frac{r}{K_1}\right) + \frac{1}{r} q \frac{n_2^2}{n^2} \left(1 - \frac{r}{K_2}\right) + q \frac{n_2(n_1 - n_2)}{n^2} + o(q).\end{aligned}$$

Hence, the achievability claim of Theorem 7.3 can be concluded as follows:

$$\limsup_{n \to \infty} \frac{L^*(r)}{q} \leq \limsup_{n \to \infty} \frac{\bar{L}_{\tilde{A}}}{q}$$

$$
\begin{aligned}
&\leq \limsup_{n\to\infty} \frac{1}{r}\frac{n_1 n_2}{n^2}\left(1-\frac{r}{K_1}\right) + \limsup_{n\to\infty} \frac{1}{r}\frac{n_2^2}{n^2}\left(1-\frac{r}{K_1}\right) \\
&\quad + \limsup_{n\to\infty} \frac{n_2(n_1-n_2)}{n^2} \\
&= \frac{1}{2r}\left(1-\frac{2r}{K}\right). \qquad \text{(F.9)}
\end{aligned}
$$

F.4 Converse for the Random Bi-partite Model

Here we provide a lower bound on the optimal average communication load for the random bi-partite model that is within a constant factor of the upper bounds and complete the proof of Theorem 7.3. Consider $\mathcal{G} = (\mathcal{V}_1 \cup \mathcal{V}_2, \mathcal{E})$ and assume that $n_1 \geq n_2$. To derive a lower bound on $L^*(r)$, for every realization of $\mathrm{RB}(n_1, n_2, q)$ graph, we arbitrarily remove $n_1 - n_2$ vertices in $\mathcal{V}_1$ along with their corresponding edges. The new bi-partite graph represents two random ER graphs with n_2 vertices. Consider Reducing the vertices in one side of the new graph, e.g. $\mathcal{V}_2$. Clearly, this provides a lower bound on $L^*(r)$. Note that now each Mapper can benefit from a redundancy factor of $2r$. According to Theorem 7.1, Reducing $\mathcal{V}_2$ induces the (optimal) communication load of $\frac{1}{2r}q\left(1-\frac{2r}{K}\right)+o(q)$ which implies

$$
\limsup_{n\to\infty} \frac{L^*(r)}{q} \geq \limsup_{n\to\infty} \frac{1}{2r} q \frac{n_2^2}{n^2}\left(1-\frac{2r}{K}\right) + o(q) = \frac{1}{8r}\left(1-\frac{2r}{K}\right). \qquad \text{(F.10)}
$$

Hence, the proof of converse of Theorem 7.3 is complete. Furthermore, (F.9) and (F.10) together asymptotically characterize the optimal average normalized communication load $L^*(r)$ within a factor of 4.

F.5 Achievability for the Stochastic Block Model

In this Section, we specialize our proposed scheme in Section 7.4 for the stochastic block model and prove the achievability of Theorem 7.4. Consider an SBM(n_1, n_2, p, q) graph $\mathcal{G} = (\mathcal{V}_1 \cup \mathcal{V}_2, \mathcal{E}_1 \cup \mathcal{E}_2 \cup \mathcal{E}_3)$ with $n = n_1 + n_2$, $|\mathcal{V}_1| = n_1 = \Theta(n)$, and $|\mathcal{V}_2| = n_2 = \Theta(n)$. Edge subsets $\mathcal{E}_1$, $\mathcal{E}_2$ and $\mathcal{E}_3$ respectively represent intra-cluster edges among vertices in $\mathcal{V}_1$, intra-cluster edges among vertices in $\mathcal{V}_2$, and inter-cluster edges between vertices in $\mathcal{V}_1$ and $\mathcal{V}_2$. Let $\mathcal{G}_1 = (\mathcal{V}_1, \mathcal{E}_1)$ and $\mathcal{G}_2 = (\mathcal{V}_2, \mathcal{E}_2)$ be graphs induced by $\mathcal{V}_1$ and $\mathcal{V}_2$, respectively, and denote the graph of inter-cluster connections by $\mathcal{G}_3 = (\mathcal{V}_1 \cup \mathcal{V}_2, \mathcal{E}_3)$. Clearly, $\mathcal{G}_1$ and $\mathcal{G}_2$ are ER(n_1, p) and ER(n_2, p) graphs, while $\mathcal{G}_3$ is RB(n_1, n_2, q) graph.

Subgraph and Reduce allocations are described as follows. Mappers in $\mathcal{V}_1$ and Reducers in $\mathcal{V}_2$ are distributedly allocated to K servers according to the allocation scheme proposed in Section 7.4. Similarly, Mappers in $\mathcal{V}_2$ and Reducers in $\mathcal{V}_1$ are distributedly allocated to K servers according to the allocation scheme proposed in Section 7.4. Therefore, each server Maps $n_1 r/K$ vertices in $\mathcal{V}_1$ and $n_2 r/K$ vertices in $\mathcal{V}_2$, inducing the computation load r. Moreover, each server Reduces n_1/K functions in $\mathcal{V}_1$ and n_2/K functions in $\mathcal{V}_2$. We consider this allocation, denoted by $\tilde{A}$, for both uncoded and coded Shuffling schemes. In uncoded scheme, Reducing each function in $\mathcal{V}_1$ requires on average pn_1 intermediate values Mapped by vertices in $\mathcal{V}_1$ due to intra-cluster connections which introduces the average uncoded load $\bar{L}^{\mathsf{UC1}}_{\tilde{A}} = p\frac{n_1^2}{(n_1+n_2)^2}\left(1 - \frac{r}{K}\right)$. Similarly, the average uncoded load for Reducing $\mathcal{V}_2$ due to intra-cluster connections is $\bar{L}^{\mathsf{UC2}}_{\tilde{A}} = p\frac{n_2^2}{(n_1+n_2)^2}\left(1 - \frac{r}{K}\right)$. Moreover, inter-cluster connections induce an average load $\bar{L}^{\mathsf{UC3}}_{\tilde{A}} = q\frac{2n_1n_2}{(n_1+n_2)^2}\left(1 - \frac{r}{K}\right)$.

In the coded scheme, we propose to employ coded Shuffling for the ER and RB models in the regime of interest, that is $p = \omega(\frac{1}{n^2})$, $q = \omega(\frac{1}{n^2})$ and $p \geq q$. Thus, the overall communication load can be decomposed into three components. We first apply the coded Shuffling scheme described in Section 7.4.1 to ER graph $\mathcal{G}_1$ which induces the average normalized communication load

$$\bar{L}_{\tilde{A}}^{\mathsf{C1}} \leq \frac{1}{r}\bar{L}_{\tilde{A}}^{\mathsf{UC1}} + o(p) = \frac{1}{r}p\frac{n_1^2}{(n_1+n_2)^2}\left(1-\frac{r}{K}\right) + o(p).$$

Similarly, the same scheme applied to ER graph $\mathcal{G}_2$ results in the average normalized communication load

$$\bar{L}_{\tilde{A}}^{\mathsf{C2}} \leq \frac{1}{r}\bar{L}_{\tilde{A}}^{\mathsf{UC2}} + o(p) = \frac{1}{r}p\frac{n_2^2}{(n_1+n_2)^2}\left(1-\frac{r}{K}\right) + o(p).$$

Finally, we employ the same scheme twice for the two ER models constituting the RB graph $\mathcal{G}_3$ which induces the average normalized communication load

$$\bar{L}_{\tilde{A}}^{\mathsf{C3}} \leq \frac{1}{r}\bar{L}_{\tilde{A}}^{\mathsf{UC3}} + o(q) = \frac{1}{r}q\frac{2n_1n_2}{(n_1+n_2)^2}\left(1-\frac{r}{K}\right) + o(q).$$

Let us denote by $\bar{L}_{\tilde{A}}^{\mathsf{C}}$ and $\bar{L}_{\tilde{A}}^{\mathsf{UC}}$ the total average normalized communication loads of the coded and uncoded schemes, respectively. Therefore,

$$\begin{aligned}
L^*(r) &\leq \bar{L}_{\tilde{A}}^{\mathsf{C}} \\
&= \bar{L}_{\tilde{A}}^{\mathsf{C1}} + \bar{L}_{\tilde{A}}^{\mathsf{C2}} + \bar{L}_{\tilde{A}}^{\mathsf{C3}} \\
&\leq \frac{1}{r}(\bar{L}_{\tilde{A}}^{\mathsf{UC1}} + \bar{L}_{\tilde{A}}^{\mathsf{UC2}} + \bar{L}_{\tilde{A}}^{\mathsf{UC3}}) + o(p) \\
&= \frac{1}{r}\bar{L}_{\tilde{A}}^{\mathsf{UC}} + o(p) \\
&= \frac{pn_1^2 + pn_2^2 + 2qn_1n_2}{(n_1+n_2)^2}\left(1-\frac{r}{K}\right) + o(p),
\end{aligned}$$

which concludes the proof of achievability of Theorem 7.4.

F.6 Converse for the Stochastic Block Model

In this section, we provide the proof of the converse of Theorem 7.4. Consider an SBM(n_1, n_2, p, q) graph $\mathcal{G} = (\mathcal{V}_1 \cup \mathcal{V}_2, \mathcal{E}_1 \cup \mathcal{E}_2 \cup \mathcal{E}_3)$ with $n = n_1 + n_2$, $|\mathcal{V}_1| = n_1 = \Theta(n)$, and $|\mathcal{V}_2| = n_2 = \Theta(n)$. Our approach to derive a lower bound for the minimum average communication load is to randomly remove edges from the two intra-cluster edges, i.e. $\mathcal{E}_1$ and $\mathcal{E}_2$. Moreover, edges are removed such that each of those clusters are then Erdos-Renyi models with connectivity probability q (reduced from p). This can be simply verified by the following coupling-type argument. Let the Bernoulli random variable E_p denote the indicator of existence of a generic edge in an ER(n, p) graph, i.e. $\Pr[E_p = 1] = 1 - p$. Now, generate another Bernoulli E_q by randomly removing edges from the realized ER graph as follows:

$$E_q = \begin{cases} \text{if } E_p = 0 & 0 \\ \text{if } E_p = 1 & \begin{cases} 0 & \text{w.p. } 1 - q/p \\ 1 & \text{w.p. } q/p. \end{cases} \end{cases}$$

Clearly, E_q is Bernoulli(q) and the resulting graph has fewer number of edges compared to the original one (with probability 1). By doing so for the two ER components of the SBM graph, we have a larger ER graph of size $n = n_1 + n_2$ with connectivity probability q. Using the converse in Theorem 7.1, we have the following for average normalized communication load for the stochastic block model:

$$\frac{L^*(r)}{q} \geq \frac{1}{r}\left(1 - \frac{r}{K}\right).$$

Lemma F.2 *For all $p \in [0, 1]$ and $s' > 0$, we have* $\left(pe^{s'} + 1 - p\right)^2 \leq pe^{2s'} + 1 - p$.

Proof: For given $p \in [0,1]$, define $f(s') = \big(pe^{s'} + 1 - p\big)^2 - \big(pe^{2s'} + 1 - p\big)$. Clearly $f(0) = 0$. Moreover,

$$f'(s') = 2p\bar{p}(e^{s'} - e^{2s'}) < 0,$$

for $s' > 0$. Therefore, $f(s') \leq 0$ for all $s' > 0$, concluding the claim of the lemma. ■